EEL 4914C Senior Design Spring 2000

April 28, 2000
Control System for Pneuman
Scott Kanowitz

Design Team:
Scott Nortman
Scott Kanowitz

Customers:
Dr. Schwartz
Dr. Arroyo

Table of Contents

1. Project Scope

1.1 Description

The primary goal of the senior design project is the fabrication of a robot controller. The design will include the development of hardware to support a microcontroller and software for a high-level computer system.

1.2 Target Market

The target market for the system is any company involved in robotics development. This includes the defense department, the entertainment industry, and research laboratories.

2. Customer Needs

2.1 CUSTOMER NEEDS TABLE

Rank	Customer Need
5	Autonomous control
5	Precision control of the 8 DOF
4	Speech synthesizer
3	GUI control and monitor program

2.2 The customer needs were formulated from the lacking abilities of a previous project design, Omnibot 2000. After reviewing the problems associated with the older platform, Drs. Arroyo and Schwartz determined that a more robust design was needed.

In a discussion between Scott Nortman and I, the new requirements for the platform were developed. We determined the above needs table for the new control system of the improved robot, Pneuman.

3. Project Specifications

3.1 PROJECT SPECIFICATIONS TABLE

Metrics	**Vale/Range**
Autonomous Control	As long as program is running
Control of Degrees of Freedom (DOF)	Control Eight DOF
Speech	Capability to say anything
Software Manager	Real Time Operating System
GUI	Cylinder status

3.2 Metric Descriptions:

I. **Autonomous Control**: The Linux control system will be able to operate the system based only on interpreted data without any user input.

II. **Control of the Eight Degrees of Freedom**: The control system will be able to adjust eight pneumatic cylinders and maintain a requested position via a PID controller

III. **Speech Synthesizer**: The Linux control system will allow user to interact with the system without the aid of a monitor. The system will inform the user about the status of the system through spoken words.

IV. **Software Control Manager**: The software modules will be contained in a repository model. The manager will allow the user to control which modules are executed at run time, and will initialize the shared memory space.

V. **User Control and Monitor Program**: A module will be included to allow user control over all the aspects of the system. The monitor program will display system conditions and allow overrides of any executing process

4. Verification of Project Specifications

4.1 Testing Methods:

I. **Autonomous Control:** The Linux system was executed with the freedom to move the cylinders without user input. The cylinders all moved as expected and the Linux system reflected the changes

II. **Control of the Eight Degrees of Freedom:** The user can enter commands into the user monitor program. The 68HC11 microcontroller will then generate the signals needed to position and hold the pneumatic cylinders.

III. **Speech Synthesizer:** The Linux user monitor program was executed and the synthesized voice informed the user as to the state of the system

IV. **Software Control Manager:** The manager has shown the contents in shared memory. Test processes were executed via the manager and the processes communicated via shared memory

V. **User Control Monitor Program:** The operator controlled all aspects of the system via the monitor program. Various system parameters were changed and the system responded to the changes.

4.2: All of the above project specifications performed as desired. The software running on the HC11 will operate the cylinders indefinitely. The Linux system will allow the user to control all eight cylinders simultaneously while viewing the status of the system. The software modules developed for the Linux system operate seamlessly communicating through shared memory. The entire system operates without any user input to position the cylinders to various positions. The speech module informs the user as to the status of the system and actions the system is taking.

5. Project Details

5.1 Block Diagrams

The block diagram for the entire system includes the cylinders, control modules, embedded computer, HC11, level shifter, voice synthesizer, and

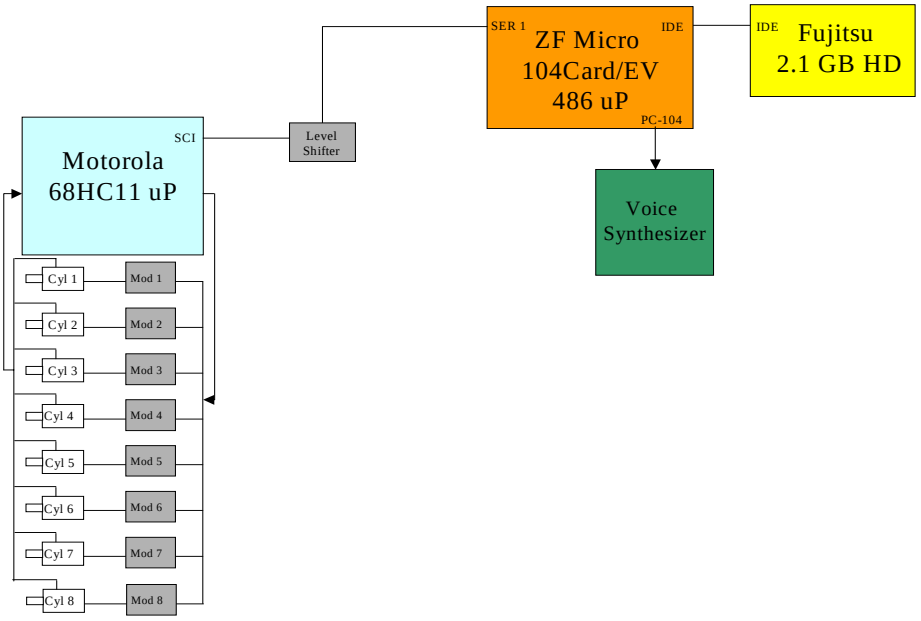

Fig. 5.1 System Block Diagram

hard drive. The main controller unit is the ZF Microsystems 486 embedded microprocessor system. The secondary devices include a Motorola 68HC11 microcontroller to generate the PWM signals, an RC Systems DoubleTalk PC104 voice synthesizer module for speech, and the valve controller modules to position the cylinders.

The ZF Microsystems PC 104 board was donated from the manufacturer. It contains an entire 486 computer in a PC-104 form factor including 16 megabytes of RAM, two serial ports, a parallel port, an IDE

controller, and a VGA monitor port. Additionally, the DoubleTalk PC 104 Voice Synthesizer module connects to the 486 processor via the PC104 bus.

The valve controller modules were used to control the airflow through the cylinder. A five-way valve and 2 two-way valves were used with a controller board to actuate the cylinders. The following is a block diagram of the valve controller module.

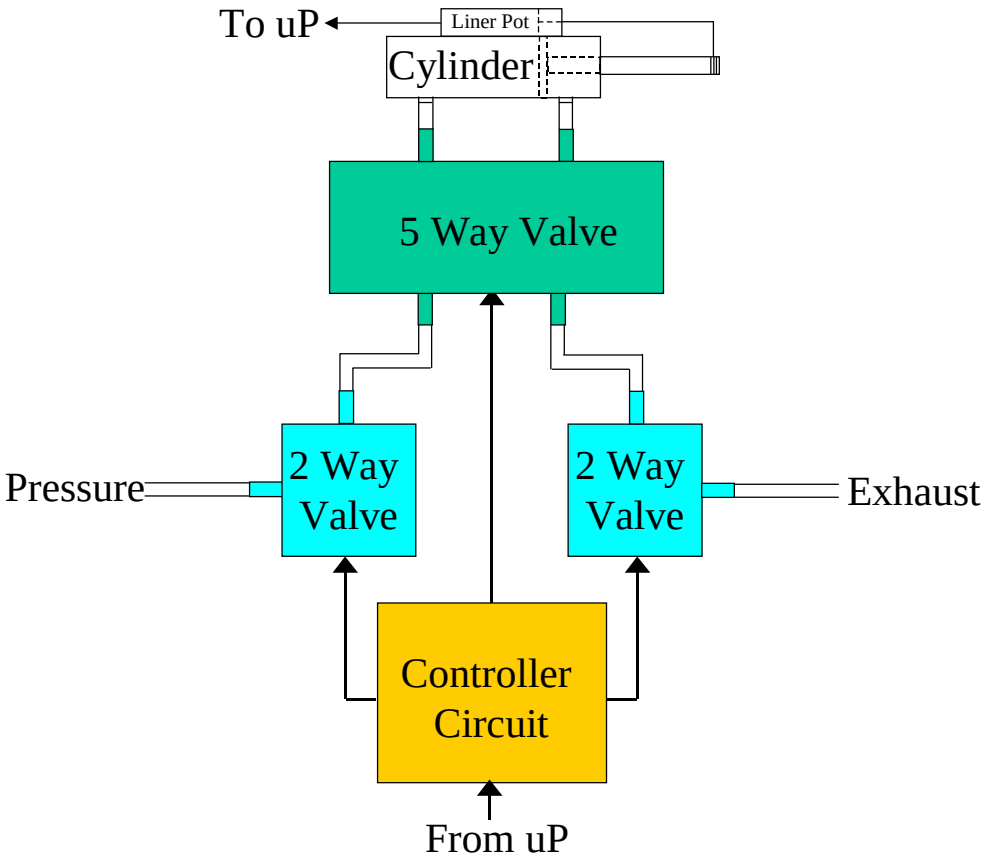

Fig. 5.2 Valve Controller Module Block Diagram

5.2 Schematics

The following is the controller board schematic for the valve controller module.

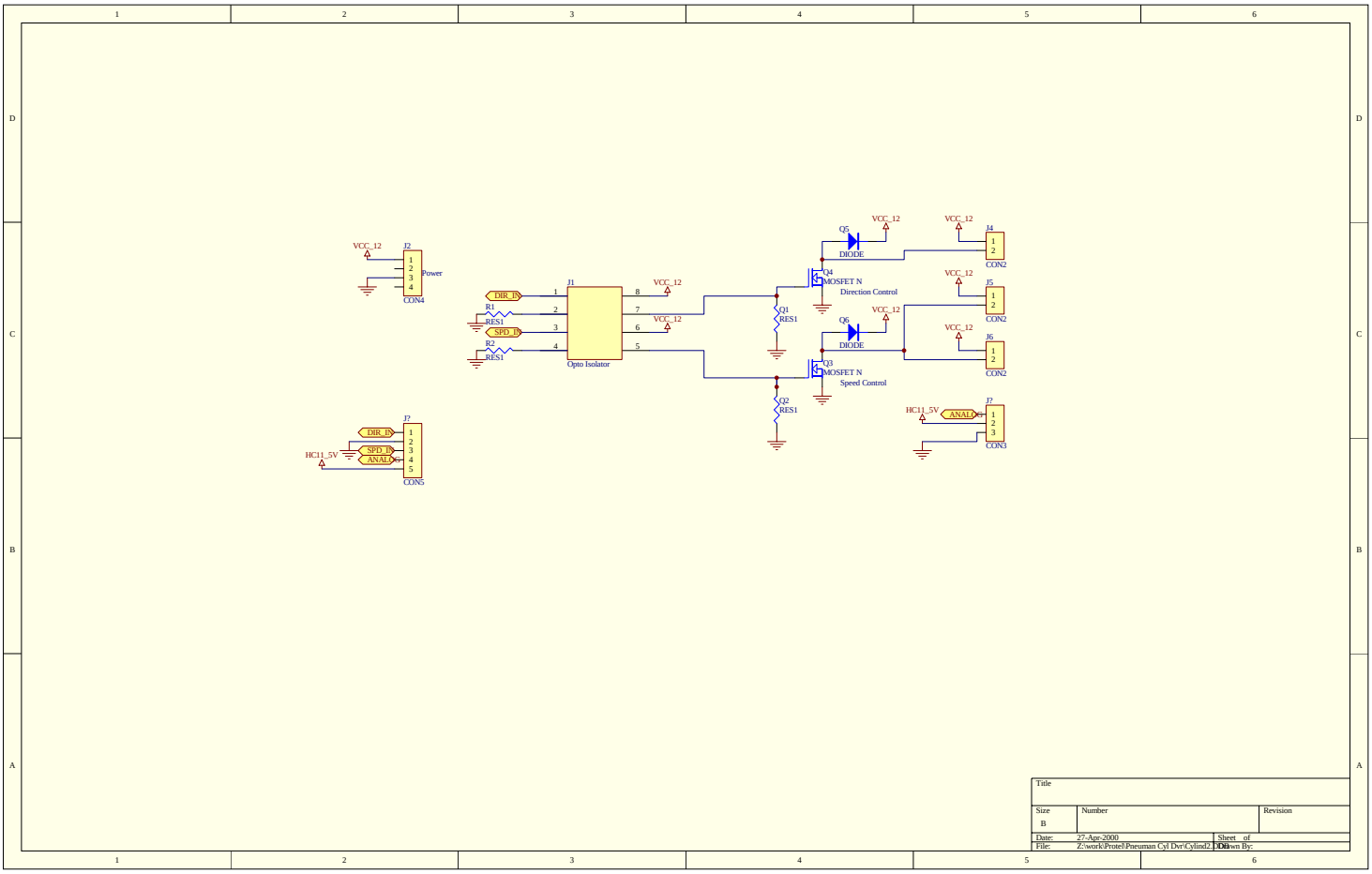

Fig. 5.3 Controller Board Schematic

The following is the printed circuit board for the valve controller module control board.

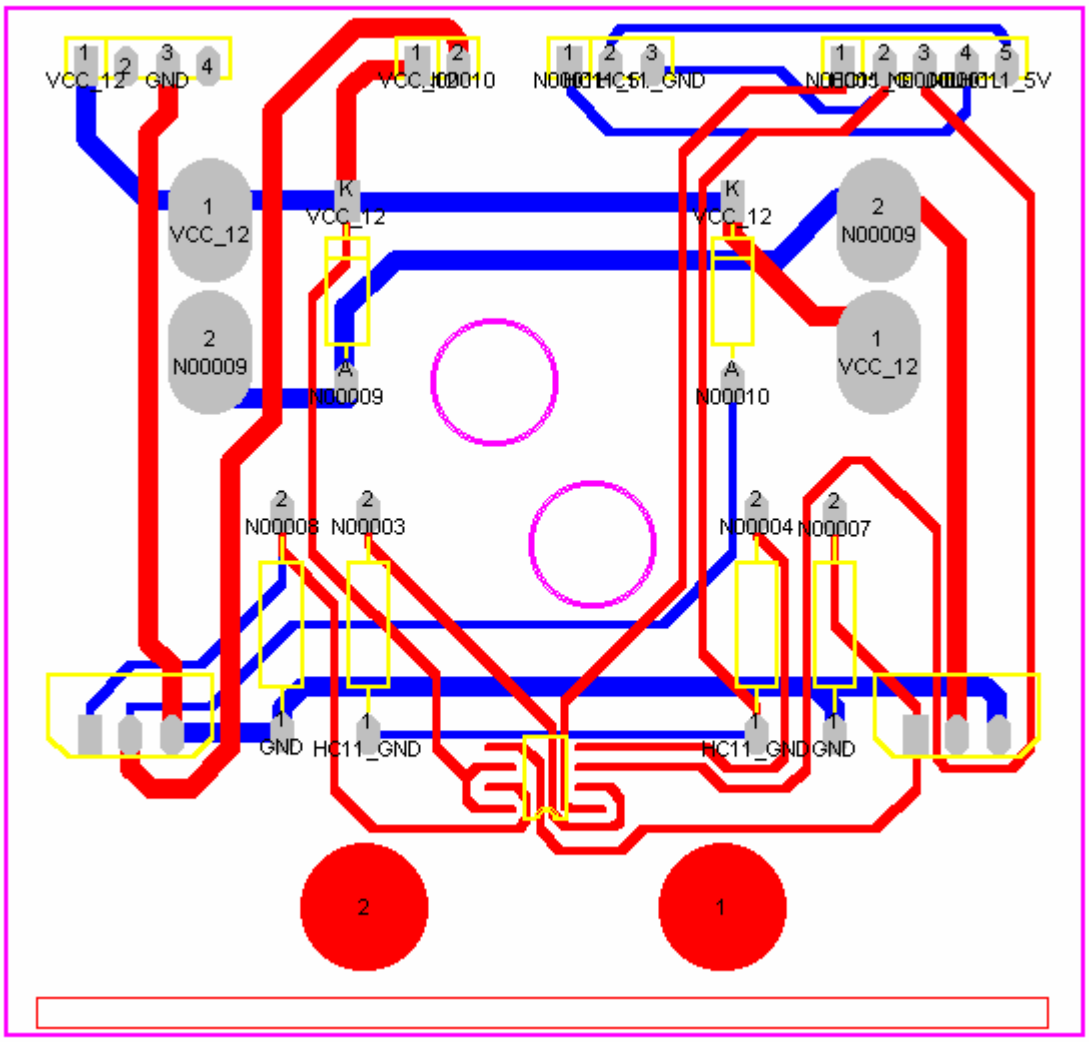

Fig. 5.4 Controller Board PCB

The Mekatronix TJ-pro board was mounted to an expansion board to allow all 8 cylinders to be operated. The following is the schematic of the expansion board.

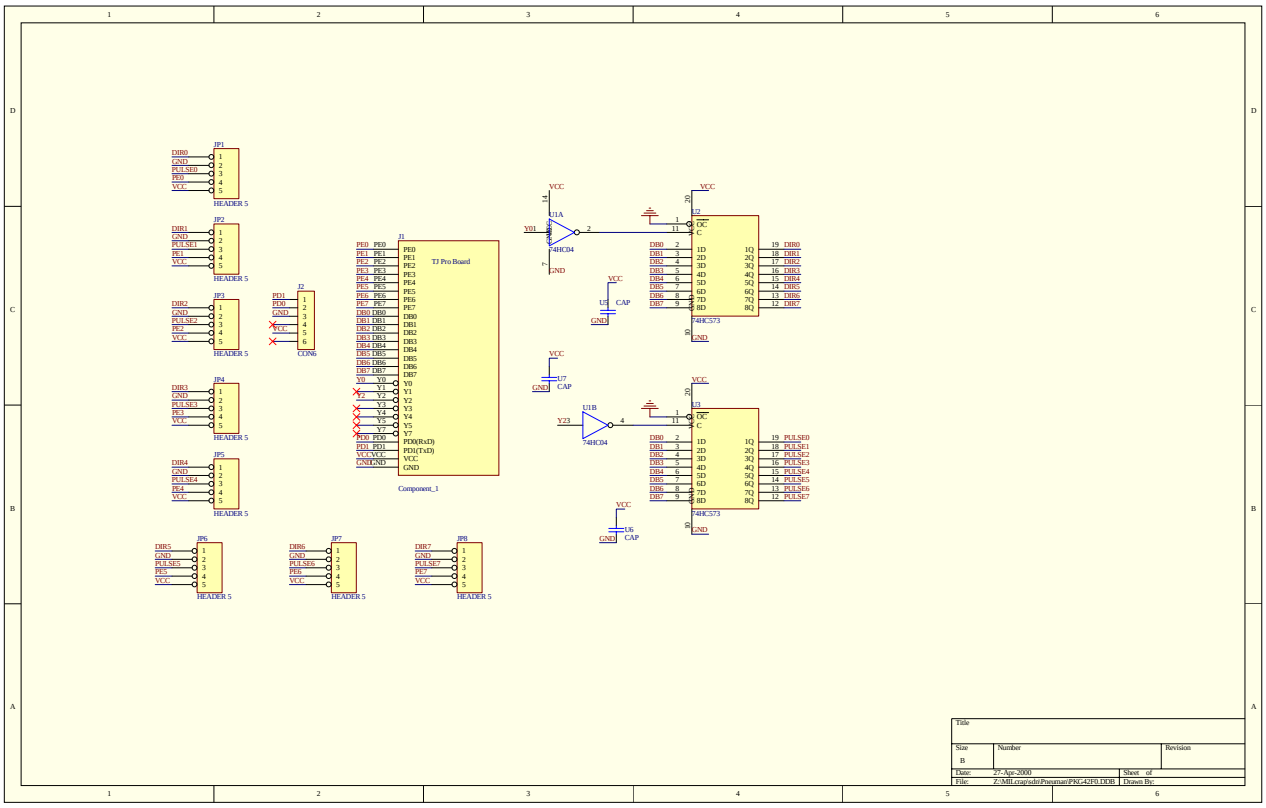

Fig. 5.5 Expansion Board Schematic

The following is the printed circuit board for the expansion board.

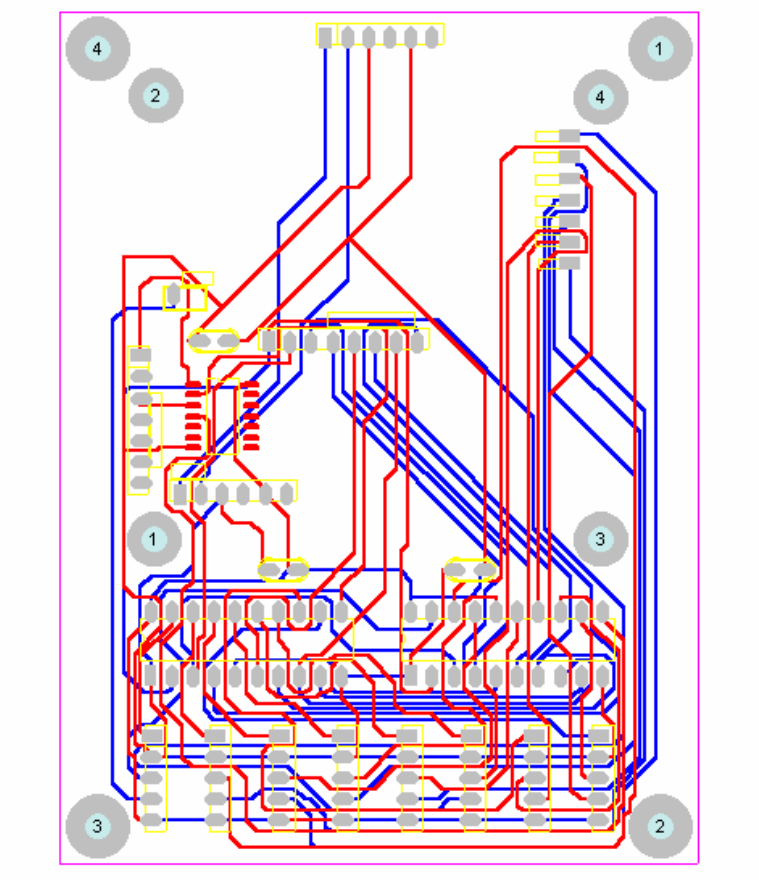

Fig. 5.6 Expansion Board PCB

5.3 Software Flow Charts

The software flow charts describe the operation of the programs. The following is the flow chart for the HC11 main code.

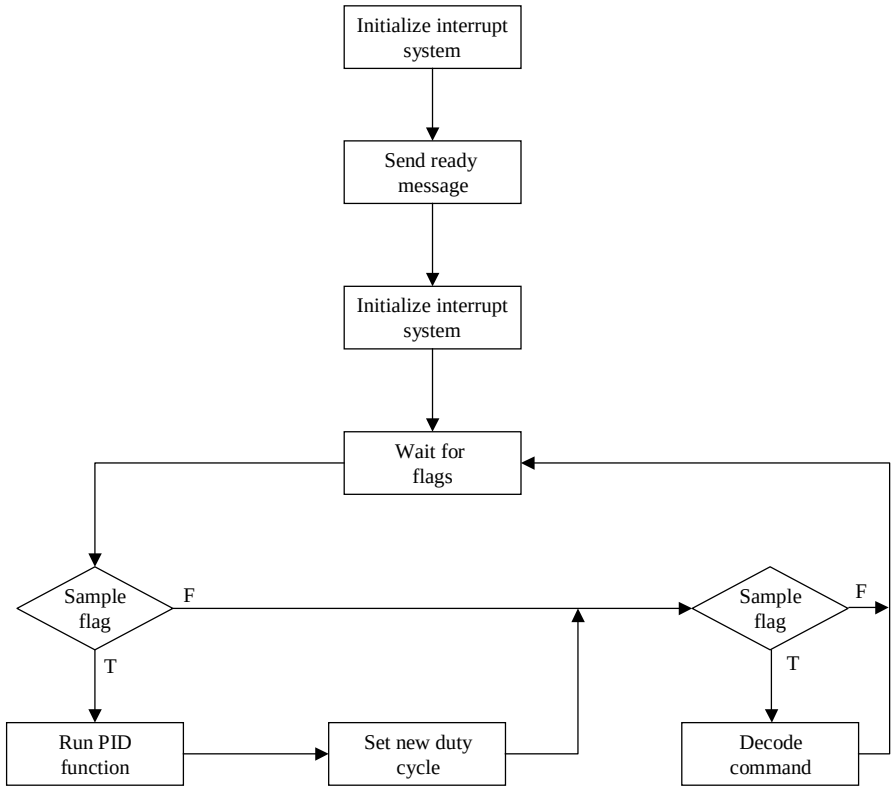

Fig. 5.7 HC11 Main Code Flow Chart

The following is the flow chart for the HC11 interrupt system.

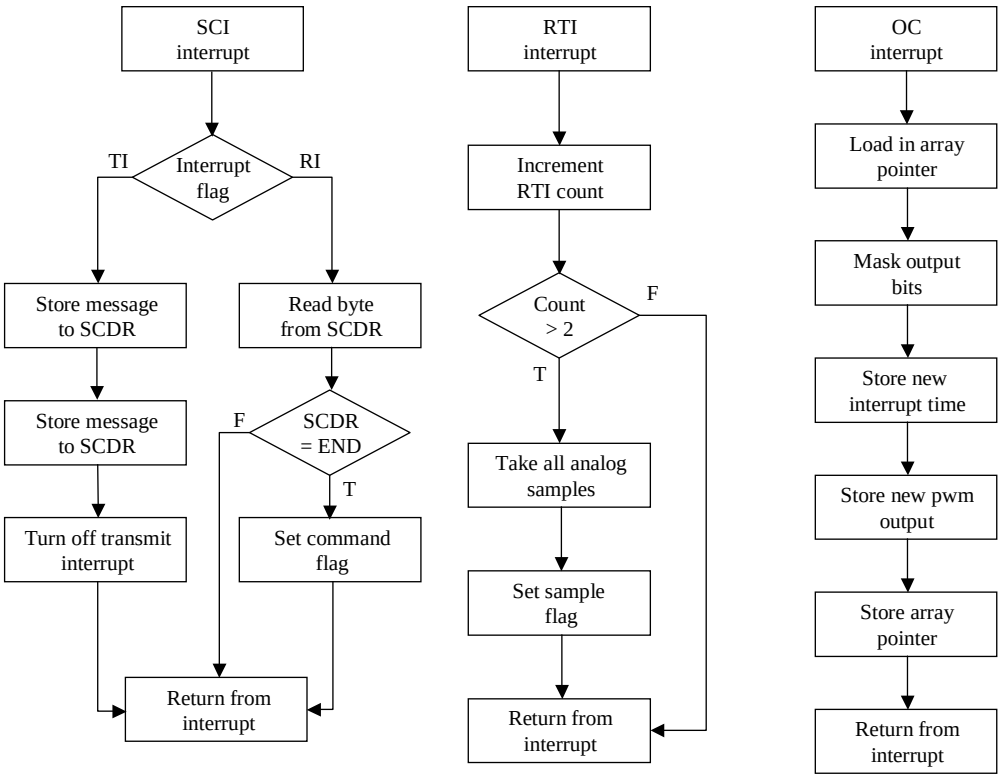

Fig. 5.8 HC11 Interrupt System Flow Chart

The following is the flow chart for the software control manager running on the Linux system.

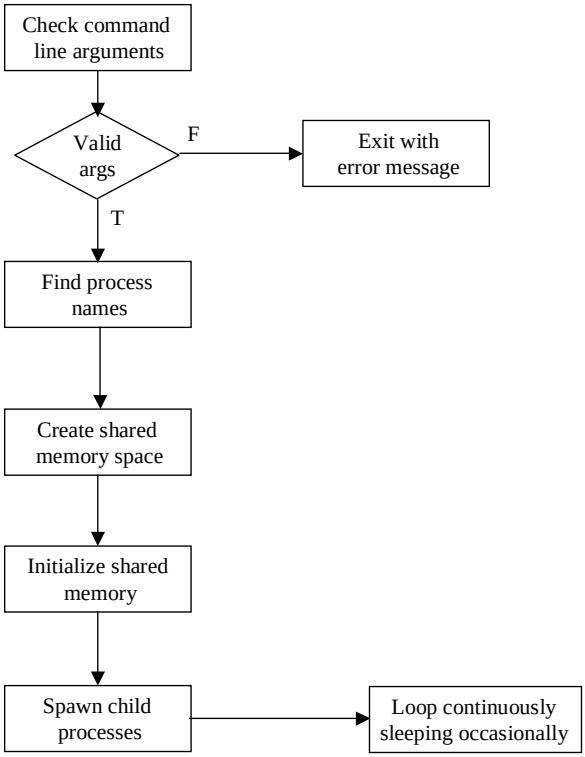

Fig. 5.9 Software Control Manager Flow Chart

The following is the flow chart for the user control monitor program.

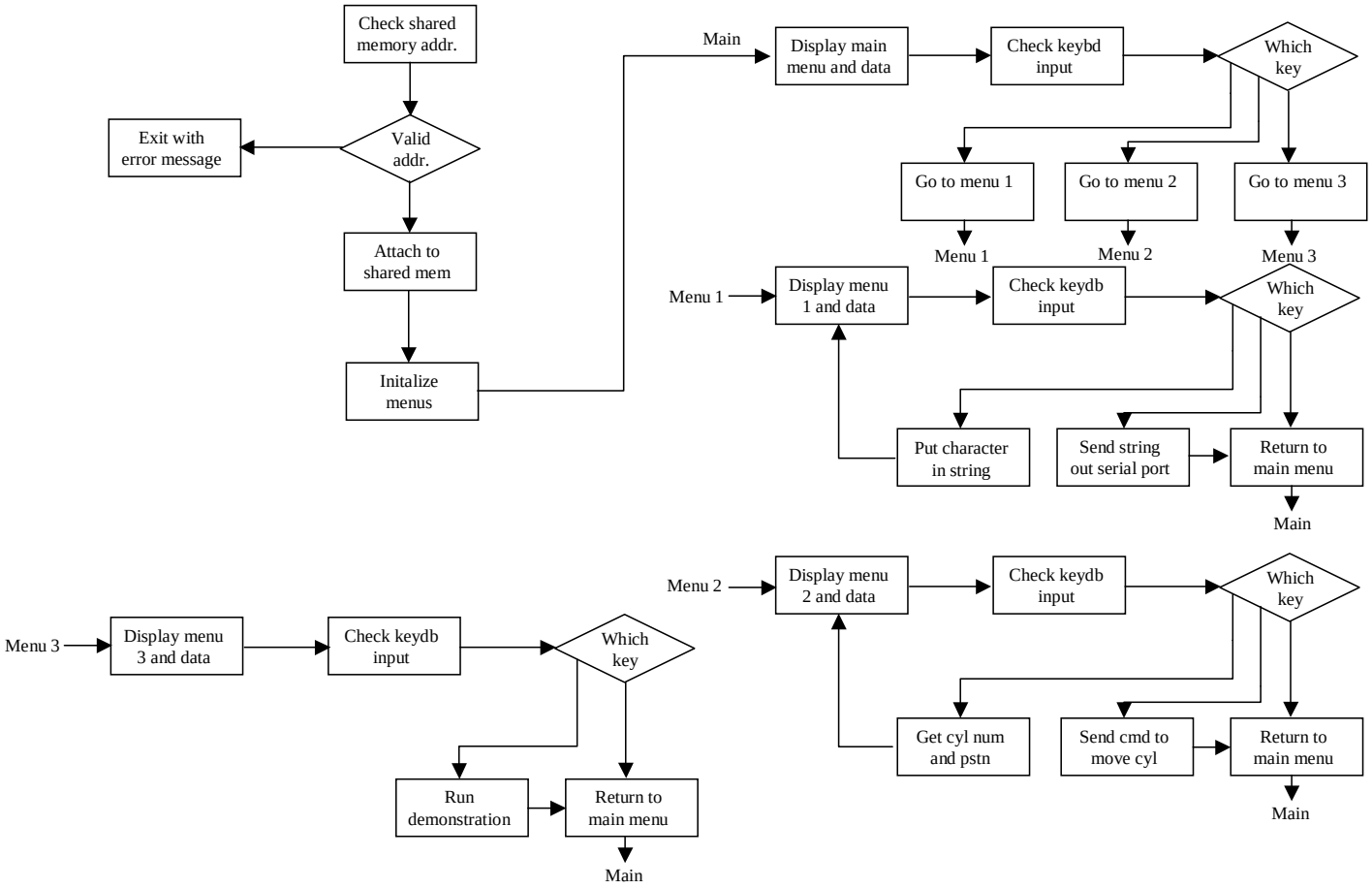

Fig. 5.10 User Control Monitor Program Flow Chart

5.4 Packaging and Assembly

The circuitry for the valve controller module was mounted directly to the valve block. The block contained all the circuitry and hardware necessary for controlling one valve including a five-way valve and 2 two-way valves. The following is a rendering of the valve controller module.

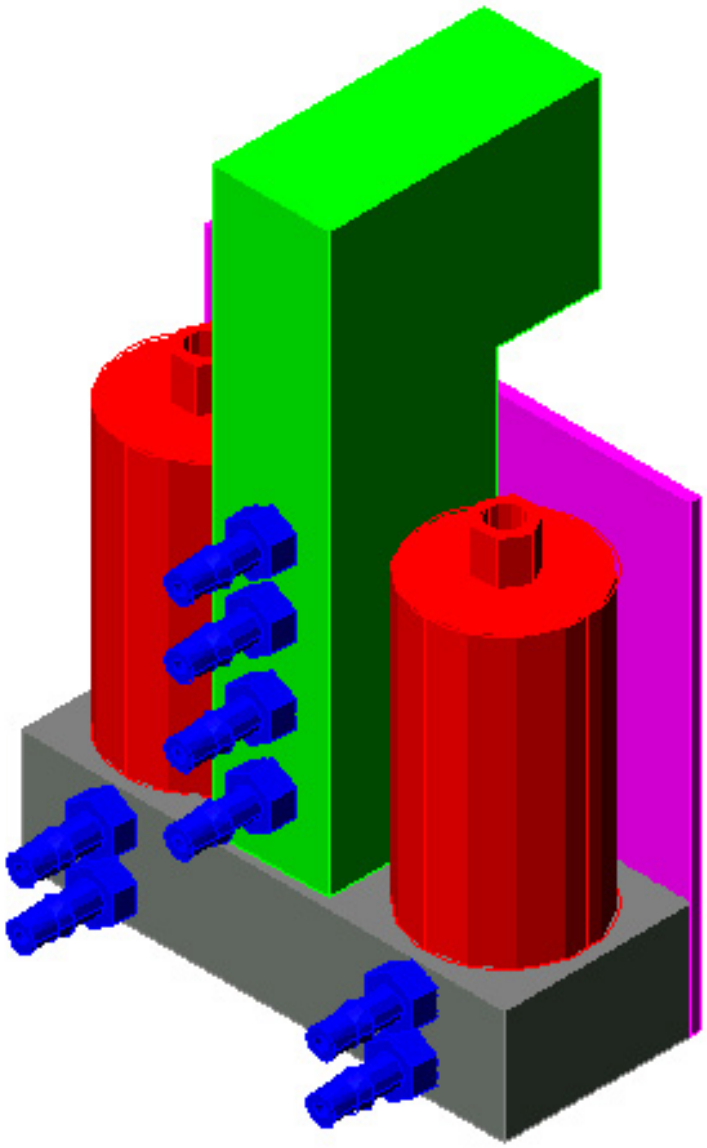

Fig. 5.11 Valve Controller Module

6. Project Cost

Part Desrciption	Actual Cost	Projected Cost
ZF Microsystems 486	donated	$550.00
Valve Modules	$102.30	$87.50
Pneumatic Cylinder	$59.00	$51.20
Speech Module	$150.09	$148.47
Linear Transducer	$9.98	$10.50
Pneumatic fittings	$28.00	$27.94
Mekatronix TJP board	$20.00	$20.00
TJP Module	$19.87	$17.90
Total Cost	**$389.24**	**$960.04**

7. Project Summary

The project proved to be challenging. It included the implementation of a variety of topics taught in our undergraduate coursework. To enable software control of the cylinder a circuit board was necessary to operate the valves. A PID controller was implemented in the software to allow for smooth positioning of the cylinder with limited overshoot. Intense C programming was needed to implement the software control manager and user control monitor programs on the Linux system. Although there are a variety of subsystems used to implement the design, they all perform correctly enabling the whole system to perform as specified in the project specifications.

Appendix A. Gantt Chart and Subproject Descriptions

A.1 GANTT CHART

	2/7	2/14	2/21	2/28	3/6	3/13	3/20	3/27	4/3	4/10
Con Mod	X	X	X	X	X					
6811 Code		X	X	X	X	X	X	X		
6811 Hard	X	X	X	X	X					
Speech Inter			X	X	X	X				
Speech Code					X	X				
Linux Code				X	X	X	X	X	X	X

A.2 Subproject Descriptions

A.2.1 Subproject #1
Control Module
The control module will house the valves and control board to interface to the cylinder air intake and the HC11 control board. The base of the module will be machined from aluminum.

A.2.2 Subproject #2
68HC11 Code Design
Code will be designed to operate on the HC11 TJ Pro boards. The code will enable precision control of the valves via the control module, and communication with the high-level computer.

A.2.3 Subproject #3
68HC11 Hardware Interfacing
The HC11s will be interfaced with the RS-232 communication module, and with each of the cylinder control modules via the controller board.

A.2.4 Subproject #4
Speech Synthesizer Interfacing
The speech synthesizer module will be interfaced with the HC11s to allow simple control of the module.

A.2.5 Subproject #5

Speech Synthesizer Code Design

Code will be designed to enable the Linux system to control the speech synthesizer module.

A.2.6 Subproject #6

Linux Code Design

Various programs will be designed to operate the high-level computer. The code will be designed for the Linux operating system. The code will be broken into many sub processes to allow ease of development.

Appendix B. Software for HC11

B.1 Demoday.c

```
/*-----------------------------------------------------*
 *Program    : demoday.c                               *
 *Version    : HC11-1.0                                *
 *Programmer : Scott Kanowitz                          *
 *Contracter : Machine Intelligence Labroatory         *
 *             University of Florida                   *
 *Project    : Pneuman                                 *
 *Date       : 3/19/2000                               *
 *-----------------------------------------------------*
 * Description : A program to run on the HC11 to      *
 *               control 8 cylinders via a PID         *
 *               controller and communicate with       *
 *               the host machine via serial line      *
 *-----------------------------------------------------*/


//#include <multvlve.h>
#include <hc11.h>
#include <mil.h>
#include <analog.h>
#include <pwmvals.h>

/* Set up interrupt vectors (also at end) */
extern void _start(void);  /* entry point in crt??.s */


#pragma interrupt_handler TOC2_isr
#pragma interrupt_handler TOC3_isr
#pragma interrupt_handler TOC4_isr
#pragma interrupt_handler TOC5_isr
#pragma interrupt_handler SCI_isr
#pragma interrupt_handler RTI_isr



#define DUMMY_ENTRY (void (*)(void))0xFFFF /*used as blank space in inpt
vectors*/
#define COMMAND_LENGTH 7 //(start char) + (5 char command(ie C1150)) + (end
char)
#define START_CHAR 0x24 //corresponds to $
#define END_CHAR 0x23 //corresponds to #

#define P 0
#define I 1
#define D 2

#define TOLERANCE 4

#define DIR_PORT *(unsigned char *)(0x4000)
/*-----------------------------------------------------*
 *Global variables                                                    *
 *
      *
 *-----------------------------------------------------*/
//char cylinderNumber = 0;

char Message = 0;
char CommandPos = 0;
```

```
char handleCommandFlag = 0;
char Command[2* COMMAND_LENGTH];

char   handleSampleFlag  = 0;

int    cyl1_pstn    = 0;
int          cyl2_pstn    = 0;
int          cyl3_pstn    = 0;
int          cyl4_pstn    = 0;
int          cyl5_pstn    = 0;
int          cyl6_pstn    = 0;
int          cyl7_pstn    = 0;
int          cyl8_pstn    = 0;

int          cyl1_des     = 100;
int          cyl2_des     = 100;
int          cyl3_des     = 100;
int          cyl4_des     = 100;
int          cyl5_des     = 100;
int          cyl6_des     = 100;
int          cyl7_des     = 100;
int          cyl8_des     = 100;

int error1[3] = {0, 0, 0};
int error2[3] = {0, 0, 0};
int error3[3] = {0, 0, 0};
int error4[3] = {0, 0, 0};
int error5[3] = {0, 0, 0};
int error6[3] = {0, 0, 0};
int error7[3] = {0, 0, 0};
int error8[3] = {0, 0, 0};

int K1[3] = {10, 0, 0};
int K2[3] = {10, 0, 0};
int K3[3] = {10, 0, 0};
int K4[3] = {10, 0, 0};
int K5[3] = {10, 0, 0};
int K6[3] = {10, 0, 0};
int K7[3] = {10, 0, 0};
int K8[3] = {10, 0, 0};


int motor1        =0;
int motor2        =0;
int motor3        =0;
int motor4        =0;
int motor5        =0;
int motor6        =0;
int motor7        =0;
int motor8        =0;

char calc_1ot    =0;
int calc_1vl     =0;
char calc_2ot  =0;
int calc_2vl   =0;
char calc_3ot  =0;
int calc_3vl   =0;

char new_motor_num   = 0;
int new_motor_spd    = 0;
int comp_motor_spd   = 0;
char first_motor     = 0;
char same_motor          = 0;
```

```
char *oc5_reg        = 0;
char oc5_1ot         = 0;
int oc5_1vl          = 0;
char oc5_2ot      = 0;
int oc5_2vl           = 0;
char oc5_3ot         = 0;
int oc5_3vl          = 0;

char *oc4_reg = 0;
char oc4_1ot  = 0;
int oc4_1vl   = 0;
char oc4_2ot  = 0;
int oc4_2vl    = 0;
char oc4_3ot  = 0;
int oc4_3vl   = 0;

char *oc3_reg = 0;
char oc3_1ot  = 0;
int oc3_1vl   = 0;
char oc3_2ot  = 0;
int oc3_2vl    = 0;
char oc3_3ot  = 0;
int oc3_3vl   = 0;

char *oc2_reg = 0;
char oc2_1ot  = 0;
int oc2_1vl   = 0;
char oc2_2ot  = 0;
int oc2_2vl    = 0;
char oc2_3ot  = 0;
int oc2_3vl   = 0;

int x_preserve = 0;

char mot_spd = 0;

int pwm_port  = 0x5000;

int wave_period = 0xEA60;

char RTIcount =0;


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

  INITIALIZATION FUNCTIONS

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

/* Set SCI for 9600, 8-n-1, TE, RE, and rx interrupt enabled, tx disabled */
void init_SCI(){
  BAUD |= 0X30;
  SCCR1 = 0X00;
  SCCR2 |= 0X2C;
  SCCR2 &= 0X7f;
}


/* Set up oc2-5 to interrupt and dissconnect from pins*/
void init_pulse()
{
       INTR_OFF();
       CLEAR_BIT(TCTL1, 0xFF);    //Disconnect all OCs from pins
       TOC2 = 3000;
```

```
       TOC3 = TOC2+3000; //offset all the initial interrupt times
       TOC4 = TOC3+3000;
       TOC5 = TOC4+3000;
       SET_BIT(TMSK1, 0x78);      //enable oc2-5
       CLEAR_BIT(PACTL, 0x04); //Enable oc5
       INTR_ON(); //turn on interrupts
}


/* Initialize RTI to interrupt every 32.768ms */
void init_RTI() {
  PACTL |= 0X03;
  TMSK2 |= 0X40;
}


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

  MULTVLVE function to set up duty_cycles

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

void multvlve(int new_duty, char mot_num)
{
new_motor_spd = new_duty;
new_motor_num = mot_num;


asm("stx _x_preserve\n"
       "ldab _new_motor_num\n"
       "cmpb #$01\n"
       "bne is_motor_2\n"
       "ldx _new_motor_spd\n"
       "stx _motor1\n"
       "ldx _motor2\n"
       "stx _comp_motor_spd\n"
       "ldab #$01\n"
       "stab _first_motor\n"
       "jmp calc_motor_times\n"
"is_motor_2:\n"
       "cmpb #$02\n"
       "bne is_motor_3\n"
       "ldx _new_motor_spd\n"
       "stx _motor2\n"
       "ldx _motor1\n"
       "stx _comp_motor_spd\n"
       "clr _first_motor\n"
       "jmp calc_motor_times");
asm("is_motor_3:\n"
       "cmpb #$03\n"
       "bne is_motor_4\n"
       "ldx _new_motor_spd\n"
       "stx _motor3\n"
       "ldx _motor4\n"
       "stx _comp_motor_spd\n"
       "ldab #$01\n"
       "stab _first_motor\n"
       "jmp calc_motor_times\n"
"is_motor_4:\n"
       "cmpb #$04\n"
       "bne is_motor_5\n"
       "ldx _new_motor_spd\n"
       "stx _motor4\n"
       "ldx _motor3\n"
```

```
       "stx _comp_motor_spd\n"
       "clr _first_motor\n"
       "jmp calc_motor_times");
asm("is_motor_5:\n"
       "cmpb #$05\n"
       "bne is_motor_6\n"
       "ldx _new_motor_spd\n"
       "stx _motor5\n"
       "ldx _motor6\n"
       "stx _comp_motor_spd\n"
       "ldab #$01\n"
       "stab _first_motor\n"
       "jmp calc_motor_times\n"
"is_motor_6:\n"
       "cmpb #$06\n"
       "bne is_motor_7\n"
       "ldx _new_motor_spd\n"
       "stx _motor6\n"
       "ldx _motor5\n"
       "stx _comp_motor_spd\n"
       "clr _first_motor\n"
       "jmp calc_motor_times");
asm("is_motor_7:\n"
       "cmpb #$07\n"
       "bne is_motor_8\n"
       "ldx _new_motor_spd\n"
       "stx _motor7\n"
       "ldx _motor8\n"
       "stx _comp_motor_spd\n"
       "ldab #$01\n"
       "stab _first_motor\n"
       "jmp calc_motor_times\n"
"is_motor_8:\n"
       "cmpb #$08\n"
       "bne end_routine\n"
       "ldx _new_motor_spd\n"
       "stx _motor8\n"
       "ldx _motor7\n"
       "stx _comp_motor_spd\n"
       "clr _first_motor\n"
       "bra calc_motor_times\n");
asm("end_routine:\n"
       "jmp multvlve_done\n"
"calc_motor_times:\n"
       "clr _same_motor\n"
       "ldx _new_motor_spd\n"
       "cpx _comp_motor_spd\n"
       "bne calc_motor_nequ\n"
       "ldaa #$01\n"
       "staa _same_motor\n"
"calc_motor_nequ:\n"
       "cpx _wave_period\n"
       "bne calc_mot_con1\n"
       "jmp clc_mot_40\n"
"calc_mot_con1:\n"
       "cpx #$0000\n"
       "bne calc_mot_con2\n"
       "jmp clc_mot_00");
asm("calc_mot_con2:\n"
       "ldaa _same_motor\n"
       "bne calc_motor_same\n"
       "cpx _comp_motor_spd\n"
       "bls clc_mot_les\n"
       "jmp clc_mot_gt\n"
```

```
"clc_mot_les:\n"
       "stx _calc_1vl\n"
       "ldd _comp_motor_spd\n"
       "subd _new_motor_spd\n"
       "std _calc_2vl\n"
       "ldd _wave_period\n"
       "cpd _comp_motor_spd\n"
       "beq clc_mot_lesc\n"
       "subd _calc_2vl\n"
       "subd _calc_1vl\n"
       "std _calc_3vl\n"
       "ldaa #0b00000011\n"
       "staa _calc_1ot\n"
       "ldaa #0b00000010\n"
       "staa _calc_2ot\n"
       "ldaa #0b00000000\n"
       "staa _calc_3ot\n"
       "jmp calc_mot_fin");
asm("clc_mot_lesc:\n"
       "subd _new_motor_spd\n"
       "subd #1000\n"
       "std _calc_3vl\n"
       "ldd #1000\n"
       "std _calc_2vl\n"
       "ldaa _first_motor\n"
       "bne is_mot_lessc\n"
       "ldaa #0b00000011\n"
       "staa _calc_1ot\n"
       "ldaa #0b00000001\n"
       "staa _calc_3ot\n"
       "staa _calc_2ot\n"
       "jmp calc_mot_fin\n"
"is_mot_lessc:\n"
       "ldaa #0b00000011\n"
       "staa _calc_1ot\n"
       "ldaa #0b00000010\n"
       "staa _calc_3ot\n"
       "staa _calc_2ot\n"
       "jmp calc_mot_fin");
asm("calc_motor_same:\n"
       "ldd _new_motor_spd\n"
       "std _calc_1vl\n"
       "ldd #1000\n"
       "std _calc_2vl\n"
       "ldd _wave_period\n"
       "subd _calc_2vl\n"
       "subd _calc_1vl\n"
       "std _calc_3vl\n"
       "ldaa #0b00000011\n"
       "staa _calc_1ot\n"
       "ldaa #$00000000\n"
       "staa _calc_3ot\n"
       "staa _calc_2ot\n"
       "jmp calc_mot_fin");
asm("clc_mot_gt:\n"
       "ldd _comp_motor_spd\n"
       "cpd #$00\n"
       "beq clc_mot_gtr0\n"
       "std _calc_1vl\n"
       "ldd _new_motor_spd\n"
       "subd _comp_motor_spd\n"
       "std _calc_2vl\n"
       "ldd _wave_period\n"
       "subd _calc_2vl\n"
```

```
        "subd _calc_1vl\n"
        "std _calc_3vl\n"
        "ldaa _first_motor\n"
        "bne is_mot_gtrc\n"
        "ldaa #0b00000011\n"
        "staa _calc_1ot\n"
        "ldaa #0b00000010\n"
        "staa _calc_2ot\n"
        "ldaa #0b00000000\n"
        "staa _calc_3ot\n"
        "jmp calc_mot_fin")
asm("is_mot_gtrc:\n"
        "ldaa #0b00000011\n"
        "staa _calc_1ot\n"
        "ldaa #0b00000001\n"
        "staa _calc_2ot\n"
        "ldaa #0b00000000\n"
        "staa _calc_3ot\n"
        "jmp calc_mot_fin\n"
"clc_mot_gtr0:\n"
        "ldd _new_motor_spd\n"
        "std _calc_1vl\n"
        "ldd #1000\n"
        "std _calc_2vl\n"
        "ldd _wave_period\n"
        "subd _calc_2vl\n"
        "subd _calc_1vl\n"
        "std _calc_3vl\n"
        "ldaa _first_motor\n"
        "bne is_mot_gtrcc\n"
        "ldaa #0b00000010\n"
        "staa _calc_1ot\n"
        "ldaa #0b00000000\n"
        "staa _calc_3ot\n"
        "staa _calc_2ot\n"
        "jmp calc_mot_fin");
asm("is_mot_gtrcc:\n"
        "ldaa #0b00000001\n"
        "staa _calc_1ot\n"
        "ldaa #0b00000000\n"
        "staa _calc_3ot\n"
        "staa _calc_2ot\n"
        "jmp calc_mot_fin\n"
"clc_mot_00:\n"
        "ldaa _same_motor\n"
        "bne both_zero\n"
        "ldd _comp_motor_spd\n"
        "cpd _wave_period\n"
        "beq clc_mot_0040\n"
        "std _calc_1vl\n"
        "ldd #1000\n"
        "std _calc_2vl\n"
        "ldd _wave_period\n"
        "subd _calc_2vl\n"
        "subd _calc_1vl\n"
        "std _calc_3vl\n"
        "ldaa _first_motor\n"
        "bne is_mot_00\n"
        "ldaa #0b00000001\n"
        "staa _calc_1ot\n"
        "ldaa #0b00000000\n"
        "staa _calc_3ot\n"
        "staa _calc_2ot\n"
        "jmp calc_mot_fin");
```

```
asm("is_mot_00:\n"
       "ldaa #0b00000010\n"
       "staa _calc_1ot\n"
       "ldaa #0b00000000\n"
       "staa _calc_3ot\n"
       "staa _calc_2ot\n"
       "jmp calc_mot_fin\n"
"both_zero:\n"
       "ldaa #0b00000000\n"
       "staa _calc_1ot\n"
       "staa _calc_2ot\n"
       "staa _calc_3ot\n"
       "ldd #1000\n"
       "std _calc_1vl\n"
       "ldd #1000\n"
       "std _calc_2vl\n"
       "ldd _wave_period\n"
       "subd _calc_2vl\n"
       "subd _calc_1vl\n"
       "std _calc_3vl\n"
       "jmp calc_mot_fin\n"
"clc_mot_0040:\n"
       "ldd #1000\n"
       "std _calc_1vl\n"
       "std _calc_2vl\n"
       "ldd _wave_period\n"
       "subd _calc_1vl\n"
       "subd _calc_2vl\n"
       "std _calc_3vl\n"
       "ldaa #0b00000010\n"
       "ldab _first_motor\n"
       "bne is_mot_0040\n"
       "ldaa #0b00000001");
asm("is_mot_0040:\n"
       "staa _calc_1ot\n"
       "staa _calc_2ot\n"
       "staa _calc_3ot\n"
       "jmp calc_mot_fin\n"
"clc_mot_40:\n"
       "ldaa _same_motor\n"
       "bne both_40000\n"
       "ldd _comp_motor_spd\n"
       "cpd #$0000\n"
       "bne clc_mot_40c\n"
       "ldd #1000\n"
"clc_mot_40c:\n"
       "std _calc_1vl\n"
       "ldd #1000\n"
       "std _calc_2vl\n"
       "ldd _wave_period\n"
       "subd _calc_2vl\n"
       "subd _calc_1vl\n"
       "std _calc_3vl\n"
       "ldaa _first_motor\n"
       "bne is_mot_40\n"
       "ldaa #0b00000011\n"
       "ldx _comp_motor_spd\n"
       "cpx #$0000\n"
       "bne clc_mot_40cc\n"
       "ldaa #0b00000010\n"
"clc_mot_40cc:\n"
       "staa _calc_1ot\n"
       "ldaa #0b00000010\n"
       "staa _calc_3ot\n"
```

```
       "staa _calc_2ot\n"
       "jmp calc_mot_fin");
asm("is_mot_40:\n"
       "ldaa #0b00000011\n"
       "ldx _comp_motor_spd\n"
       "cpx #$0000\n"
       "bne is_mot_40c\n"
       "ldaa #0b00000001\n"
"is_mot_40c:\n"
       "staa _calc_1ot\n"
       "ldaa #0b00000001\n"
       "staa _calc_3ot\n"
       "staa _calc_2ot\n"
       "jmp calc_mot_fin\n"
"both_40000:\n"
       "ldd #1000\n"
       "std _calc_1vl\n"
       "ldd #1000\n"
       "std _calc_2vl\n"
       "ldd _wave_period\n"
       "subd _calc_2vl\n"
       "subd _calc_1vl\n"
       "std _calc_3vl\n"
       "ldaa #0b00000011\n"
       "staa _calc_1ot\n"
       "staa _calc_2ot\n"
       "staa _calc_3ot\n"
       "jmp calc_mot_fin");
asm("calc_mot_fin:\n"
       "ldy #_oc2_1ot\n"
       "ldaa _new_motor_num\n"
       "cmpa #$01\n"
       "beq jmp_calc_motor_done\n"
       "cmpa #$02\n"
       "beq jmp_calc_motor_done\n"
       "cmpa #$03\n"
       "beq calc_motor_34\n"
       "cmpa #$04\n"
       "beq calc_motor_34\n"
       "cmpa #$05\n"
       "beq calc_motor_56\n"
       "cmpa #$06\n"
       "beq calc_motor_56\n"
       "bra calc_motor_78\n"
"jmp_calc_motor_done:\n"
       "jmp calc_motor_done");
asm("calc_motor_78:\n"
       "ldaa _calc_1ot\n"
       "lsla\n"
       "lsla\n"
       "lsla\n"
       "lsla\n"
       "lsla\n"
       "lsla\n"
       "staa _calc_1ot\n"
       "ldaa _calc_2ot\n"
       "lsla\n"
       "lsla\n"
       "lsla\n"
       "lsla\n"
       "lsla\n"
       "lsla\n"
       "staa _calc_2ot\n"
       "ldaa _calc_3ot\n"
```

```
        "lsla\n"
        "lsla\n"
        "lsla\n"
        "lsla\n"
        "lsla\n"
        "lsla\n"
        "staa _calc_3ot\n"
        "ldy #_oc5_1ot\n"
        "jmp calc_motor_done");
asm("calc_motor_56:\n"
        "ldaa _calc_1ot\n"
        "lsla\n"
        "lsla\n"
        "lsla\n"
        "lsla\n"
        "staa _calc_1ot\n"
        "ldaa _calc_2ot\n"
        "lsla\n"
        "lsla\n"
        "lsla\n"
        "lsla\n"
        "staa _calc_2ot\n"
        "ldaa _calc_3ot\n"
        "lsla\n"
        "lsla\n"
        "lsla\n"
        "lsla\n"
        "staa _calc_3ot\n"
        "ldy #_oc4_1ot\n"
        "jmp calc_motor_done");
asm("calc_motor_34:\n"
        "ldaa _calc_1ot\n"
        "lsla\n"
        "lsla\n"
        "staa _calc_1ot\n"
        "ldaa _calc_2ot\n"
        "lsla\n"
        "lsla\n"
        "staa _calc_2ot\n"
        "ldaa _calc_3ot\n"
        "lsla\n"
        "lsla\n"
        "staa _calc_3ot\n"
        "ldy #_oc3_1ot\n"
"calc_motor_done:\n"
        "ldx #_calc_1ot\n"
        "ldaa 0,x\n"
        "staa 0,y\n"
        "inx\n"
        "iny\n"
        "ldd 0,x\n"
        "std 0,y\n"
        "inx\n"
        "inx\n"
        "iny\n"
        "iny\n"
        "ldaa 0,x\n"
        "staa 0,y\n"
        "inx\n"
        "iny\n"
        "ldd 0,x\n"
        "std 0,y\n"
        "inx\n"
        "inx\n"
```

```
       "iny\n"
       "iny\n"
       "ldaa 0,x\n"
       "staa 0,y\n"
       "inx\n"
       "iny\n"
       "ldd 0,x\n"
       "std 0,y\n"
"multvlve_done:\n"
       "ldx _x_preserve");


}

/*******************************************************/
/*pid_controller()
       */
/*function contains all the pid routines to update          */
/*the duty cycles
       */
/*
             */
/*******************************************************/

void pid_controller(void){

       int proportion = 0;
       int integral = 0;
       int derivative = 0;
       int new_duty = 0;


       //Run pid for cylinder 1

              //shift error values
              error1[2] = error1[1];
              error1[1] = error1[0];
              error1[0] = cyl1_des - cyl1_pstn;

              //range for tolerance
              if (error1[0] < TOLERANCE & error1[0] > 0 - TOLERANCE){
                     error1[0] = 0;
                     multvlve(0, 0x01);
              }
              else { //run pid

                     //calc gains
                     proportion = K1[P] * error1[0];
                     //integral = K1[I];
                     //derivative = K1[D] * (error1[0] - error1[1]);

                     //calc new raw value
                     new_duty = proportion + integral + derivative;


                     if (error1[0] < 0) {
                            DIR_PORT &= 0xFE;
                     }
                     else {
                            DIR_PORT |= 0x01;
                     }

                     //take absolute value of new_duty
                     if (new_duty < 0){
```

```
                asm("ldd %new_duty\n"
                        "subd #1\n"
                        "coma\n"
                        "comb\n"
                        "std %new_duty\n");
        }
        //range new_duty to max value
        if (new_duty > 236){
                        new_duty = 236;
                }

        new_duty = pwm_vals[new_duty];


        multvlve(new_duty, 0x01);

    }

//Run pid for cylinder 2

    //shift error values
    error2[2] = error2[1];
    error2[1] = error2[0];
    error2[0] = cyl2_des - cyl2_pstn;

    //range for tolerance
    if (error2[0] < TOLERANCE & error2[0] > 0 - TOLERANCE){
        error2[0] = 0;
        multvlve(0, 0x02);
    }
    else { //run pid

        //calc gains
        proportion = K2[P] * error2[0];
        //integral = K2[I];
        //derivative = K2[D] * (error2[0] - error2[1]);

        //calc new raw value
        new_duty = proportion + integral + derivative;


        if (error2[0] < 0) {
            DIR_PORT &= 0xFD;
        }
        else {
            DIR_PORT |= 0x02;
        }

        //take absolute value of new_duty
        if (new_duty < 0){
            asm("ldd %new_duty\n"
                    "subd #1\n"
                    "coma\n"
                    "comb\n"
                    "std %new_duty\n");
        }
        //range new_duty to max value
        if (new_duty > 236){
                    new_duty = 236;
            }

        new_duty = pwm_vals[new_duty];
```

```
            multvlve(new_duty, 0x02);

      }


 //Run pid for cylinder 3

      //shift error values
      error3[2] = error3[1];
      error3[1] = error3[0];
      error3[0] = cyl3_des - cyl3_pstn;

      //range for tolerance
      if (error3[0] < TOLERANCE & error3[0] > 0 - TOLERANCE){
            error3[0] = 0;
            multvlve(0, 0x03);
      }
      else { //run pid

            //calc gains
            proportion = K3[P] * error3[0];
            //integral = K3[I];
            //derivative = K3[D] * (error3[0] - error3[1]);

            //calc new raw value
            new_duty = proportion + integral + derivative;


            if (error3[0] < 0) {
                  DIR_PORT &= 0xFB;
            }
            else {
                  DIR_PORT |= 0x04;
            }

            //take absolute value of new_duty
            if (new_duty < 0){
                  asm("ldd %new_duty\n"
                        "subd #1\n"
                        "coma\n"
                        "comb\n"
                        "std %new_duty\n");
            }
            //range new_duty to max value
            if (new_duty > 236){
                        new_duty = 236;
                  }

            new_duty = pwm_vals[new_duty];


            multvlve(new_duty, 0x03);

      }



//Run pid for cylinder 4

      //shift error values
      error4[2] = error4[1];
      error4[1] = error4[0];
      error4[0] = cyl4_des - cyl4_pstn;
```

```
	//range for tolerance
	if (error4[0] < TOLERANCE & error4[0] > 0 - TOLERANCE){
		error4[0] = 0;
		multvlve(0, 0x04);
	}
	else { //run pid

		//calc gains
		proportion = K4[P] * error4[0];
		//integral = K4[I];
		//derivative = K4[D] * (error4[0] - error4[1]);

		//calc new raw value
		new_duty = proportion + integral + derivative;


		if (error4[0] < 0) {
			DIR_PORT &= 0xF7;
		}
		else {
			DIR_PORT |= 0x08;
		}

		//take absolute value of new_duty
		if (new_duty < 0){
			asm("ldd %new_duty\n"
				"subd #1\n"
				"coma\n"
				"comb\n"
				"std %new_duty\n");
		}
		//range new_duty to max value
		if (new_duty > 236){
				new_duty = 236;
			}

		new_duty = pwm_vals[new_duty];


		multvlve(new_duty, 0x04);

	}


//Run pid for cylinder 5

	//shift error values
	error5[2] = error5[1];
	error5[1] = error5[0];
	error5[0] = cyl5_des - cyl5_pstn;

	//range for tolerance
	if (error5[0] < TOLERANCE & error5[0] > 0 - TOLERANCE){
		error5[0] = 0;
		multvlve(0, 0x05);
	}
	else { //run pid

		//calc gains
		proportion = K5[P] * error5[0];
		//integral = K5[I];
		//derivative = K5[D] * (error5[0] - error5[1]);

		//calc new raw value
```

```
		new_duty = proportion + integral + derivative;


		if (error5[0] < 0) {
			DIR_PORT &= 0xEF;
		}
		else {
			DIR_PORT |= 0x10;
		}

		//take absolute value of new_duty
		if (new_duty < 0){
			asm("ldd %new_duty\n"
				"subd #1\n"
				"coma\n"
				"comb\n"
				"std %new_duty\n");
		}
		//range new_duty to max value
		if (new_duty > 236){
				new_duty = 236;
			}

		new_duty = pwm_vals[new_duty];


		multvlve(new_duty, 0x05);

	}


//Run pid for cylinder 6

	//shift error values
	error6[2] = error6[1];
	error6[1] = error6[0];
	error6[0] = cyl6_des - cyl6_pstn;

	//range for tolerance
	if (error6[0] < TOLERANCE & error6[0] > 0 - TOLERANCE){
		error6[0] = 0;
		multvlve(0, 0x06);
	}
	else { //run pid

		//calc gains
		proportion = K6[P] * error6[0];
		//integral = K6[I];
		//derivative = K6[D] * (error6[0] - error6[1]);

		//calc new raw value
		new_duty = proportion + integral + derivative;


		if (error2\6[0] < 0) {
			DIR_PORT &= 0xDF;
		}
		else {
			DIR_PORT |= 0x20;
		}

		//take absolute value of new_duty
		if (new_duty < 0){
			asm("ldd %new_duty\n"
```

```
                        "subd #1\n"
                        "coma\n"
                        "comb\n"
                        "std %new_duty\n");
                }
                //range new_duty to max value
                if (new_duty > 236){
                        new_duty = 236;
                }

                new_duty = pwm_vals[new_duty];


                multvlve(new_duty, 0x06);

        }



//Run pid for cylinder 7

        //shift error values
        error7[2] = error7[1];
        error7[1] = error7[0];
        error7[0] = cyl7_des - cyl7_pstn;

        //range for tolerance
        if (error7[0] < TOLERANCE & error7[0] > 0 - TOLERANCE){
                error7[0] = 0;
                multvlve(0, 0x07);
        }
        else { //run pid

                //calc gains
                proportion = K7[P] * error7[0];
                //integral = K7[I];
                //derivative = K7[D] * (error7[0] - error7[1]);

                //calc new raw value
                new_duty = proportion + integral + derivative;


                if (error7[0] < 0) {
                        DIR_PORT &= 0xBF;
                }
                else {
                        DIR_PORT |= 0x40;
                }

                //take absolute value of new_duty
                if (new_duty < 0){
                        asm("ldd %new_duty\n"
                                "subd #1\n"
                                "coma\n"
                                "comb\n"
                                "std %new_duty\n");
                }
                //range new_duty to max value
                if (new_duty > 236){
                                new_duty = 236;
                        }

                new_duty = pwm_vals[new_duty];
```

```
                multvlve(new_duty, 0x07);

        }


    //Run pid for cylinder 8

        //shift error values
        error8[2] = error8[1];
        error8[1] = error8[0];
        error8[0] = cyl8_des - cyl8_pstn;

        //range for tolerance
        if (error8[0] < TOLERANCE & error8[0] > 0 - TOLERANCE){
                error8[0] = 0;
                multvlve(0, 0x08);
        }
        else { //run pid

                //calc gains
                proportion = K8[P] * error8[0];
                //integral = K8[I];
                //derivative = K8[D] * (error8[0] - error8[1]);

                //calc new raw value
                new_duty = proportion + integral + derivative;


                if (error8[0] < 0) {
                        DIR_PORT &= 0x7F;
                }
                else {
                        DIR_PORT |= 0x80;
                }

                //take absolute value of new_duty
                if (new_duty < 0){
                        asm("ldd %new_duty\n"
                                "subd #1\n"
                                "coma\n"
                                "comb\n"
                                "std %new_duty\n");
                }
                //range new_duty to max value
                if (new_duty > 236){
                                new_duty = 236;
                        }

                new_duty = pwm_vals[new_duty];


                multvlve(new_duty, 0x08);

        }



    handleSampleFlag = 0;
}
```

```
/*******************************************************/
/*decode_command()
      */
/*function to decode the commands recieved on the           */
/*serial port
      */
/*
            */
/*******************************************************/

void decode_command(void){

      int counter = 0;
      char cylinderNumber = 0;
      char cylinderDes = 0;


      //search through command string to find start char
      //in case of bogus data
      while(counter < COMMAND_LENGTH && Command[counter] != START_CHAR)
      {counter++;
      }

      if (counter == COMMAND_LENGTH){//bogus data so return
            handleCommandFlag = 0;
            return;
            }

      switch (Command[counter + 1]){
            case 'C': //cylinder position value
                  cylinderNumber = Command[counter + 2] & 0x0F;
                  Command[counter + 3] &= 0x0F;
                  Command[counter + 4] &= 0x0F;
                  Command[counter + 5] &= 0x0F;
                  cylinderDes = Command[counter + 3] * 100 + Command[counter
+ 4] * 10 + Command[counter + 5];
                  //store new command into global variables
                  asm("ldab %cylinderNumber\n"
                  "subb #1\n"
                  "lslb\n"
                  "ldaa %cylinderDes\n"
                  "stx _x_preserve\n"
                  "ldx #_cyl1_des\n"
                  "abx\n"
                  "inx\n"
                  "staa 0,x\n"
                  "ldx _x_preserve");

                  break ;

            //put next command case here

      }//ends switch statement

      handleCommandFlag = 0;
}

/*******************************************************/
/*send_message(byte)
      */
/*function to send a byte out the serial port               */
/*
            */
```

```
/*
             */
/*****************************************************/

void send_message(char byte)
{
       Message = byte;
    SCCR2 |= 0x80;
}

/*****************************************************/
/*ISRs
       */
/*
             */
/*
             */
/*
             */
/*****************************************************/

/* Captures incoming messages of CommandLength bytes */
/* Sends one byte messages then disables TIE */
/* SCI isr */
void SCI_isr() {
  if(SCSR & 0X20) {
    Command[CommandPos] = SCDR;
    send_message(SCDR);                 // Include to echo characters
    if (++CommandPos == COMMAND_LENGTH) {
      CommandPos = 0;
    }
    if (SCDR == END_CHAR) {
      handleCommandFlag = 1;
    }
  }
  else if(SCSR & 0X80) {
    SCDR = Message;
    SCCR2 &= 0X7F;
  }

}


/*TOC5_isr- controls motors 7 and 8*/
void TOC5_isr()
{
       TFLG1 = 0x08; //clear interrupt flag
   asm("ldy _oc5_reg\n"
       "ldaa _mot_spd\n"
       "anda #$3f\n"
       "oraa 0,y\n"
       "staa _mot_spd\n"
       "ldx _pwm_port\n"
       "staa 0,x\n"
       "iny\n"
       "ldx #$101E\n"
       "ldd 0,y\n"
       "addd 0,x\n"
       "std 0,x\n"
       "cpy #_oc5_3vl\n"
       "bne oc5_rti\n"
       "ldy #_oc5_reg\n"
"oc5_rti:\n"
       "iny\n"
```

```
       "iny\n"
       "sty _oc5_reg");
}

/*TOC4_isr - controls motors 5 and 6*/
void TOC4_isr()
{
       TFLG1 = 0X10; //clear interrupt flag
   asm("ldy _oc4_reg\n"
       "ldaa _mot_spd\n"
       "anda #$cf\n"
       "oraa 0,y\n"
       "staa _mot_spd\n"
       "ldx _pwm_port\n"
       "staa 0,x\n"
       "iny\n"
       "ldx #$101C\n"
       "ldd 0,y\n"
       "addd 0,x\n"
       "std 0,x\n"
       "cpy #_oc4_3vl\n"
       "bne oc4_rti\n"
       "ldy #_oc4_reg\n"
"oc4_rti:\n"
       "iny\n"
       "iny\n"
       "sty _oc4_reg");
}


/*TOC3_isr - controls motors 3 and 4*/
void TOC3_isr()
{
       TFLG1 = 0x20; //clear interrupt flag
   asm("ldy _oc3_reg\n"
       "ldaa _mot_spd\n"
       "anda #$f3\n"
       "oraa 0,y\n"
       "staa _mot_spd\n"
       "ldx _pwm_port\n"
       "staa 0,x\n"
       "iny\n"
       "ldx #$101A\n"
       "ldd 0,y\n"
       "addd 0,x\n"
       "std 0,x\n"
       "cpy #_oc3_3vl\n"
       "bne oc3_rti\n"
       "ldy #_oc3_reg\n"
"oc3_rti:\n"
       "iny\n"
       "iny\n"
       "sty _oc3_reg");
}



/*TOC2_isr - controls motors 1 and 2*/
void TOC2_isr()
{
       TFLG1 = 0x40; //clear interrupt flag

   asm("ldy _oc2_reg\n"
       "ldaa _mot_spd\n"
```

```
	"anda #$fc\n"
	"oraa 0,y\n"
	"staa _mot_spd\n"
	"ldx _pwm_port\n"
	"staa 0,x\n"
	"iny\n"
	"ldx #$1018\n"
	"ldd 0,y\n"
	"addd 0,x\n"
	"std 0,x\n"
	"cpy #_oc2_3vl\n"
	"bne oc2_rti\n"
	"ldy #_oc2_reg\n"
"oc2_rti:\n"
	"iny\n"
	"iny\n"
	"sty _oc2_reg");
}

/*RTI_isr - samples analog values on every third interrupt*/
/*sets a flag to call PID controller*/
void RTI_isr()
{
	TFLG2 = 0x40; //clear flag
	if (RTIcount < 2)
	{
		RTIcount++;
		return;
	}

	RTIcount = 0;

	//run sample of all 8 cylinders
	ADCTL = 0x90 //CCF=1 SCAN=0 MULT=1 CD-CA=0000
	//wait for complete samples
	asm("ldaa #21\n"
		"loop:\n"
		"deca\n"
		"bne loop");

	//now load in the analog values
	//assembly routines to change char to int

	//cyl1_pstn = ADR1;
	asm("ldaa $1031\n"
		"staa _cyl1_pstn + 1\n");

	//cyl2_pstn = ADR2;
	asm("ldaa $1032\n"
		"staa _cyl2_pstn + 1\n");

	//cyl3_pstn = ADR3;
	asm("ldaa $1033\n"
		"staa _cyl3_pstn + 1\n");

	//cyl4_pstn = ADR4;
	asm("ldaa $1034\n"
		"staa _cyl4_pstn + 1\n");

	ADCTL = 0x94 //CCF=1 SCAN=0 MULT=1 CD-CA=0100
	//wait for complete samples
	asm("ldaa #21\n"
		"loop2:\n"
		"deca\n"
```

```
            "bne loop2");

      //cyl5_pstn = ADR1;
      asm("ldaa $1031\n"
            "staa _cyl5_pstn + 1\n");

      //cyl6_pstn = ADR2;
      asm("ldaa $1032\n"
            "staa _cyl6_pstn + 1\n");

      //cyl7_pstn = ADR3;
      asm("ldaa $1033\n"
            "staa _cyl7_pstn + 1\n");

      //cyl8_pstn = ADR4;
      asm("ldaa $1034\n"
            "staa _cyl8_pstn + 1\n");

      //finished with sampling


      handleSampleFlag = 1; //set flag to run pid routine

}

/*******************************************************/
/*Begin main function
      */
/*
            */
/*
            */
/*
            */
/*******************************************************/

void main(void)
{

      int count = 1700;
      /*this must be here to initalize the pointers and
      leave them in the correct momory space*/

      oc2_reg = &oc2_1ot;
      oc3_reg = &oc3_1ot;
      oc4_reg = &oc4_1ot;
      oc5_reg = &oc5_1ot;

      init_analog();
      init_SCI();
      init_pulse(); //set up interrupts for oc2-5
      init_RTI();

  //printf("\n\n starting\n");
  //multvlve(20000, 0x01);

      while(1){ //main loop
            if(handleSampleFlag){
                  pid_controller();
            }//ends if

            if(handleCommandFlag){
                  decode_command();
            }//ends if
```

```
      }//ends main while
}



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

  Initialize interrupt vectors

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

#pragma abs_address:0xffd6
void (*interrupt_vectors[])(void) =
      {
      SCI_isr,            /* SCI */
      DUMMY_ENTRY,  /* SPI */
      DUMMY_ENTRY,  /* PAIE */
      DUMMY_ENTRY,  /* PAO */
      DUMMY_ENTRY,  /* TOF */
      TOC5_isr,           /* TOC5 */
      TOC4_isr,           /* TOC4 */
      TOC3_isr,           /* TOC3 */
      TOC2_isr,           /* TOC2 */
      DUMMY_ENTRY,  /* TOC1 */
      DUMMY_ENTRY,  /* TIC3 */
      DUMMY_ENTRY,  /* TIC2 */
      DUMMY_ENTRY,  /* TIC1 */
      RTI_isr,                  /* RTI */
      DUMMY_ENTRY,  /* IRQ */
      DUMMY_ENTRY,  /* XIRQ */
      DUMMY_ENTRY,  /* SWI */
      DUMMY_ENTRY,  /* ILLOP */
      DUMMY_ENTRY,  /* COP */
      DUMMY_ENTRY,  /* CLM */
      _start              /* RESET */
      };
#pragma end_abs_address
```

B.2 Pwmvals.h

```
/*-----------------------------------------------------*
 *Program    : pwmvals.h                               *
 *Version    : HC11-1.0                                *
 *Programmer : Scott Kanowitz                          *
 *Contracter : Machine Intelligence Labroatory         *
 *             University of Florida                   *
 *Project    : Pneuman                                 *
 *Date       : 3/25/2000                               *
 *-----------------------------------------------------*
 * Description : Header file containing all the       *
 *               possible duty cycles for demoday.c    *
 *-----------------------------------------------------*/

int pwm_vals[] = {0,
0x4268,
0x42CC,
0x4330,
0x4394,
0x43F8,
0x445C,
0x44C0,
0x4524,
0x4588,
0x45EC,
0x4650,
0x46B4,
0x4718,
0x477C,
0x47E0,
0x4844,
0x48A8,
0x490C,
0x4970,
0x49D4,
0x4A38,
0x4A9C,
0x4B00,
0x4B64,
0x4BC8,
0x4C2C,
0x4C90,
0x4CF4,
0x4D58,
0x4DBC,
0x4E20,
0x4E84,
0x4EE8,
0x4F4C,
0x4FB0,
0x5014,
0x5078,
0x50DC,
0x5140,
0x51A4,
0x5208,
0x526C,
0x52D0,
0x5334,
0x5398,
0x53FC,
0x5460,
```

```
0x54C4,
0x5528,
0x558C,
0x55F0,
0x5654,
0x56B8,
0x571C,
0x5780,
0x57E4,
0x5848,
0x58AC,
0x5910,
0x5974,
0x59D8,
0x5A3C,
0x5AA0,
0x5B04,
0x5B68,
0x5BCC,
0x5C30,
0x5C94,
0x5CF8,
0x5D5C,
0x5DC0,
0x5E24,
0x5E88,
0x5EEC,
0x5F50,
0x5FB4,
0x6018,
0x607C,
0x60E0,
0x6144,
0x61A8,
0x620C,
0x6270,
0x62D4,
0x6338,
0x639C,
0x6400,
0x6464,
0x64C8,
0x652C,
0x6590,
0x65F4,
0x6658,
0x66BC,
0x6720,
0x6784,
0x67E8,
0x684C,
0x68B0,
0x6914,
0x6978,
0x69DC,
0x6A40,
0x6AA4,
0x6B08,
0x6B6C,
0x6BD0,
0x6C34,
0x6C98,
0x6CFC,
0x6D60,
```

```
0x6DC4,
0x6E28,
0x6E8C,
0x6EF0,
0x6F54,
0x6FB8,
0x701C,
0x7080,
0x70E4,
0x7148,
0x71AC,
0x7210,
0x7274,
0x72D8,
0x733C,
0x73A0,
0x7404,
0x7468,
0x74CC,
0x7530,
0x7594,
0x75F8,
0x765C,
0x76C0,
0x7724,
0x7788,
0x77EC,
0x7850,
0x78B4,
0x7918,
0x797C,
0x79E0,
0x7A44,
0x7AA8,
0x7B0C,
0x7B70,
0x7BD4,
0x7C38,
0x7C9C,
0x7D00,
0x7D64,
0x7DC8,
0x7E2C,
0x7E90,
0x7EF4,
0x7F58,
0x7FBC,
0x8020,
0x8084,
0x80E8,
0x814C,
0x81B0,
0x8214,
0x8278,
0x82DC,
0x8340,
0x83A4,
0x8408,
0x846C,
0x84D0,
0x8534,
0x8598,
0x85FC,
0x8660,
```

```
0x86C4,
0x8728,
0x878C,
0x87F0,
0x8854,
0x88B8,
0x891C,
0x8980,
0x89E4,
0x8A48,
0x8AAC,
0x8B10,
0x8B74,
0x8BD8,
0x8C3C,
0x8CA0,
0x8D04,
0x8D68,
0x8DCC,
0x8E30,
0x8E94,
0x8EF8,
0x8F5C,
0x8FC0,
0x9024,
0x9088,
0x90EC,
0x9150,
0x91B4,
0x9218,
0x927C,
0x92E0,
0x9344,
0x93A8,
0x940C,
0x9470,
0x94D4,
0x9538,
0x959C,
0x9600,
0x9664,
0x96C8,
0x972C,
0x9790,
0x97F4,
0x9858,
0x98BC,
0x9920,
0x9984,
0x99E8,
0x9A4C,
0x9AB0,
0x9B14,
0x9B78,
0x9BDC,
0x9C40,
0x9CA4,
0x9D08,
0x9D6C,
0x9DD0,
0x9E34};
```

Appendix C. Software for Linux

C.1 Spwnproc.c

```
/*-----------------------------------------------------*
 *Program    : spwnproc.c                              *
 *Version    : PC/104-1.0                              *
 *Programmer : Scott Kanowitz                          *
 *Contracter : Machine Intelligence Labroatory         *
 *             University of Florida                   *
 *Project    : Pneuman                                 *
 *Date       : 3/29/2000                               *
 *-----------------------------------------------------*
 * Description : A program to allow user management   *
 *               of processes. Command line options   *
 *               allow user to specify which processes*
 *               to spawn. Initalizes shared memory   *
 *               and forks off new processes          *
 *-----------------------------------------------------*/


#include "spwnprocLib.h"
#include "user.h"

/*********************Global Variables*********************/
struct shrd_mem *shmem; /*pointer to shared memory*/
int shmid; /*shared memory id*/

/***********************Main Routine***********************/
int main(int argc, char **argv){

  int proc_counter = 0; /*counter for the number of processes to spawn*/
  char *spawnable_procs[argc]; /*pointer to the process names to spawn*/
  int i=0; /*loop variables*/
  int j=0;


  /*begin command line error checking*/
  if(argc < 2  || argc > NUM_OF_PROCS + 1){
    printf("parameters incorect\n");
    exit(1);
  }/*end if*/


  /*read command line args and fill spawnable_procs array*/
  proc_counter = 0;
  for(i=0; i < (argc - 1); i++){
    for(j=0; j < NUM_OF_PROCS; j++){
      if(strcmp(argv[i+1], CMD_ARRAY[j]) == 0){
       spawnable_procs[proc_counter] = PROCESS_ARRAY[j];
       proc_counter ++;
       }/*end if*/
     }/*end for2*/
  }/*end for1*/


  /*create and attach shared memory*/
  shmid = shmget(IPC_PRIVATE, sizeof *shmem, 0666);
  shmem = (struct shrd_mem *) shmat(shmid,0,0);

  /*clear out all of shared memory*/
  shared_mem_clear();
```

```
  /*spawn all procs*/
  spawn_procs(proc_counter, spawnable_procs);

  //run until killed
  for(;;)
    sleep(10);
}
```

C.2 Spwnproclib.c

```
/*-------------------------------------------------------*
 *Program    : spwnprocLib.c                             *
 *Version    : PC/104-1.0                                *
 *Programmer : Scott Kanowitz                            *
 *Contracter : Machine Intelligence Labroatory           *
 *             University of Florida                     *
 *Project    : Pneuman                                   *
 *Date       : 3/29/2000                                 *
 *-------------------------------------------------------*
 * Description : Contains library routines used by       *
 *               spwnproc.c                              *
 *-------------------------------------------------------*/

#include "spwnprocLib.h"
#include "user.h"



/*********************************************************
 *spawn_procs                                            *
 *                                                       *
 *library routine to spawn the processes given in the    *
 *spawnable_procs array. Uses fork and ececl to do the   *
 *process creation. Child process id's are stored in     *
 *the proc_id array in shared memory.                    *
 *********************************************************/
int spawn_procs(int proc_counter, char *spawnable_procs[]){

  extern struct shrd_mem *shmem;
  int i = 0;
  int cpid;
  char ashmem[10];  //string to convert shared mem address to string
  extern int shmid;


  sprintf(ashmem, "%d", shmid);
  // printf("SPWMPROCLIB: shared mem id = %s", ashmem);

   for(i = 0; i < proc_counter; i++){
     cpid = fork();

     if (!(cpid)){
       execl(spawnable_procs[i], spawnable_procs[i], ashmem, 0);
     }

     shmem->proc_ids[i] = cpid; /*store child process id's in shared mem*/
  }/*end for*/

   return 0;
}

/*********************************************************
 *shared_mem_clear                                       *
 *                                                       *
 *Library routine to clear (set equal to 0)              *
 *out all of the shared memory table locations           *
 *********************************************************/

void shared_mem_clear(void){
  extern struct shrd_mem *shmem;
```

```
  shmem-> despstn_cyl1 = 0; /*desired position of cylinder 1*/
  shmem-> despstn_cyl2 = 0; /*desired position of cylinder 2*/
  shmem-> despstn_cyl3 = 0; /*desired position of cylinder 3*/
  shmem-> despstn_cyl4 = 0; /*desired position of cylinder 4*/
  shmem-> despstn_cyl5 = 0; /*desired position of cylinder 5*/
  shmem-> despstn_cyl6 = 0; /*desired position of cylinder 6*/
  shmem-> despstn_cyl7 = 0; /*desired position of cylinder 7/
  shmem-> despstn_cyl8 = 0; /*desired position of cylinder 8*/
  shmem-> despstn_cyl9 = 0; /*desired position of cylinder 9*/
  shmem-> despstn_cyl10 = 0; /*desired position of cylinder 10*/
  shmem-> despstn_cyl11 = 0; /*desired position of cylinder 11*/
  shmem-> despstn_cyl12 = 0; /*desired position of cylinder 12*/
  shmem-> despstn_cyl13 = 0; /*desired position of cylinder 13*/

  shmem-> pstn_cyl1 = 0; /*current position of cylinder 1*/
  shmem-> pstn_cyl2 = 0; /*current position of cylinder 2*/
  shmem-> pstn_cyl3 = 0; /*current position of cylinder 3*/
  shmem-> pstn_cyl4 = 0; /*current position of cylinder 4*/
  shmem-> pstn_cyl5 = 0; /*current position of cylinder 5*/
  shmem-> pstn_cyl6 = 0; /*current position of cylinder 6*/
  shmem-> pstn_cyl7 = 0; /*current position of cylinder 7*/
  shmem-> pstn_cyl8 = 0; /*current position of cylinder 8*/
  shmem-> pstn_cyl9 = 0; /*current position of cylinder 9*/
  shmem-> pstn_cyl10 = 0; /*current position of cylinder 10*/
  shmem-> pstn_cyl11 = 0; /*current position of cylinder 11*/
  shmem-> pstn_cyl12 = 0; /*current position of cylinder 12*/
  shmem-> pstn_cyl13 = 0; /*current position of cylinder 13*/

}
```

C.3 Spwnproclib.h

```
/*-----------------------------------------------------*
 *Program    : spwnprocLib.h                           *
 *Version    : PC/104-1.0                              *
 *Programmer : Scott Kanowitz                          *
 *Contracter : Machine Intelligence Labroatory         *
 *             University of Florida                   *
 *Project    : Pneuman                                 *
 *Date       : 3/29/2000                               *
 *-----------------------------------------------------*
 * Description : Header file for spwnporocLib.c        *
 *-----------------------------------------------------*/


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define DEBUG         /*this is used to set the debug messages*/

int spawn_procs ( int proc_counter, char *spawnable_procs[] );
```

C.4 User.h

```
/*-----------------------------------------------------*
 *Program    : user.h                                  *
 *Version    : PC/104-1.0                              *
 *Programmer : Scott Kanowitz                          *
 *Contracter : Machine Intelligence Labroatory         *
 *             University of Florida                   *
 *Project    : Pneuman                                 *
 *Date       : 3/29/2000                               *
 *-----------------------------------------------------*
 * Description : Contains all the user changeable     *
 *               attributes. Contains the process     *
 *               table                                *
 *-----------------------------------------------------*/



#define NUM_OF_PROCS 2


   /*the commandline arguments referring to the processes definitions numbered
according to the */
static const char CMD_ARRAY[NUM_OF_PROCS][2] = {"m\0", "b\0"};
/* m = menu
   b = process2*/


static const char PROCESS_ARRAY[NUM_OF_PROCS][10] = {"menu\0", "process2\0"};
/*process names are limited to 10 characters*/

/*the shared memory table */
struct shrd_mem {
  int proc_ids[NUM_OF_PROCS];

  int despstn_cyl1; /*desired position of cylinder 1*/
  int despstn_cyl2; /*desired position of cylinder 2*/
  int despstn_cyl3; /*desired position of cylinder 3*/
  int despstn_cyl4; /*desired position of cylinder 4*/
  int despstn_cyl5; /*desired position of cylinder 5*/
  int despstn_cyl6; /*desired position of cylinder 6*/
  int despstn_cyl7; /*desired position of cylinder 7*/
  int despstn_cyl8; /*desired position of cylinder 8*/
  int despstn_cyl9; /*desired position of cylinder 9*/
  int despstn_cyl10; /*desired position of cylinder 10*/
  int despstn_cyl11; /*desired position of cylinder 11*/
  int despstn_cyl12; /*desired position of cylinder 12*/
  int despstn_cyl13; /*desired position of cylinder 13*/

  int pstn_cyl1; /*current position of cylinder 1*/
  int pstn_cyl2; /*current position of cylinder 2*/
  int pstn_cyl3; /*current position of cylinder 3*/
  int pstn_cyl4; /*current position of cylinder 4*/
  int pstn_cyl5; /*current position of cylinder 5*/
  int pstn_cyl6; /*current position of cylinder 6*/
  int pstn_cyl7; /*current position of cylinder 7*/
  int pstn_cyl8; /*current position of cylinder 8*/
  int pstn_cyl9; /*current position of cylinder 9*/
  int pstn_cyl10; /*current position of cylinder 10*/
  int pstn_cyl11; /*current position of cylinder 11*/
  int pstn_cyl12; /*current position of cylinder 12*/
  int pstn_cyl13; /*current position of cylinder 13*/
};
```

C.5 Menu.c

```
/*-------------------------------------------------------------------------*
 * Program     : menu.c                                                    *
 * Version     : 1.0                                                       *
 * Programmer  : Scott Kanowitz                                            *
 *-------------------------------------------------------------------------*
 * Date        : 04/4/00                                                   *
 *-------------------------------------------------------------------------*
 * Description : This program will display the current state of the        *
 *               system and allow users to change variables via menus      *
 *-------------------------------------------------------------------------*/

#include "menuLib.h"
#include "user.h"
#include "portOpsLib.h"

struct shrd_mem *shmem; //pointer to shared mem
int fd; //file descriptor for speech module
int ioport; //serial ioport

int main(int argc, char *argv[])
{

  int  retval;
  int menu, new_menu, cylinder_selected, integer_string;
  int cylinder_number;
  int select;
  int curs_pstn = 0;
  char in_string[CMD_LENGTH] = {'\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0',
'\0', '\0'};
  char output_string[CMD_LENGTH] = {'\0', '\0', '\0', '\0', '\0', '\0', '\0',
'\0', '\0', '\0'};
  char speak_buf[40];

  //check if shared memory address was passed
  if (argc < 2){
    printf("ERROR: menu.c, shared memory address not passed\n");
    exit(1);
  }

  //get shared memory address and attach
  shmem = (struct shrd_mem *) shmat(atoi(argv[1]),0,0);

  //open IO port at 9600 baud 8N1
  ioport = initPort(2, 9600, "8N1");
  flushBuf(ioport, 2);


  //open port to talk to doubletalk
  fd = open("/dev/dtlk", O_RDWR);
  if (fd == -1)
    {
      printf("ERROR: menu.c, can not open double talk\n");
      exit(1);
    }

  //initalize keyboard input
  if ( initscr() == NULL)
    perror("initscr");
  noecho();
  cbreak();
  keypad( stdscr, TRUE ) ;
```

```
//say begining of menu program
retval = write(fd, "pneuman user control and data analysis program", 46 + 1);

menu = MAIN;
new_menu = TRUE;
cylinder_selected = FALSE;

while(1){//begin main while loop

  if (menu == MAIN){
    printMainData();

    if (new_menu){
     printMain();
     retval = write(fd, ",main menu", 10 + 1);
     new_menu = FALSE;
    }

    //-------------------------------------------------//
    //main menu choices                                //
    //-------------------------------------------------//

    //read keyboard input for main menu
    select = readKeyboard();
    switch ( select ){
    case ERR:
     //do nothing
     break;

    case '1':
     //enter menu 1
     menu = MENU1;
     new_menu = TRUE;
     break;

    case '2':
     //enter menu 2
     menu = MENU2;
     new_menu = TRUE;
     break;

     case'3':
       //enter menu3
       menu = MENU3;
     new_menu = TRUE;
     break;

    }//ends main menu keyboard input switch
  }//ends if menu == main


  else if (menu == MENU1){
    printMenu1Data();

    if (new_menu) {
     printMenu1();
     retval = write(fd, ",send command string menu", 25 + 1);
     new_menu = FALSE;
       }

    //-------------------------------------------------//
    //menu 1 choices                                   //
```

```
  //-----------------------------------------------//

  //read keyboard input for menu 1
  select = readKeyboard();
  switch (select){
  case ERR:
   //do nothing
   break;

   //this should be the escape ESC case
  case ESC:
   //return to main menu
   menu = MAIN;
   new_menu = TRUE;
   curs_pstn = 0; //clear out buffer
   break;

   //this should be the return sequence
  case CR:
   //pressed return so send out string and return to main menu
   writeBuf(ioport, in_string, curs_pstn);
   menu = MAIN;
   new_menu = TRUE;
   curs_pstn = 0;
   break;


  default:
   //otherwise input the string

   in_string[curs_pstn] = (char)select;
   if (++curs_pstn == CMD_LENGTH){
     retval = write(fd, "string too long, try again", 26 + 1);
     curs_pstn = 0;
   }
   in_string[curs_pstn] = '\0';

  }// ends read keyboard for menu 1


}//ends if menu == menu1


else if (menu == MENU2){
   printMenu2Data();

  if (new_menu) {
   printMenu2();
   retval = write(fd, ",set cylinder position menu", 27 + 1);
   new_menu = FALSE;
  }

  select = readKeyboard();

  if ( curs_pstn + 1 == 5){//invalid input length
   menu = MAIN;
   new_menu = TRUE;
   select = ERR; //do this so the switch statement is ignored
   curs_pstn = 0;
   retval = write(fd, "invalid position", 16 + 1);
  }


  switch (select){
```

```
case ERR:
 //do nothing
 break;

case ESC:
 //return to main menu
 menu = MAIN;
 new_menu = TRUE;
 cylinder_selected = FALSE;
 cylinder_number = 0;
 break;

case '1':
 if (cylinder_selected == FALSE){
   cylinder_number = select;
   cylinder_selected = TRUE;
   retval = write(fd, "cylinder selected, now input position", 37 +1);
 }
 else{
   in_string[curs_pstn] = (char) select;
   in_string[++curs_pstn] = '\0';
     }
 break;

case '2':
 if (cylinder_selected == FALSE){
   cylinder_number = select;
   cylinder_selected = TRUE;
   retval = write(fd, "cylinder selected, now input position", 37 +1);
 }
 else{
   in_string[curs_pstn] = (char) select;
   in_string[++curs_pstn] = '\0';
 }
 break;

case '3':
 if (cylinder_selected == FALSE){
   cylinder_number = select;
   cylinder_selected = TRUE;
   retval = write(fd, "cylinder selected, now input position", 37 +1);
 }
 else{
   in_string[curs_pstn] = (char) select;
   in_string[++curs_pstn] = '\0';
 }
 break;

case '4':
 if (cylinder_selected == FALSE){
   cylinder_number = select;
   cylinder_selected = TRUE;
   retval = write(fd, "cylinder selected, now input position", 37 +1);
 }
 else{
   in_string[curs_pstn] = (char) select;
   in_string[++curs_pstn] = '\0';
 }
 break;

case '5':
 if (cylinder_selected == FALSE){
   cylinder_number = select;
   cylinder_selected = TRUE;
```

```
   retval = write(fd, "cylinder selected, now input position", 37 +1);
 }
 else{
   in_string[curs_pstn] = (char) select;
   in_string[++curs_pstn] = '\0';
 }
 break;

case '6':
 if (cylinder_selected == FALSE){
   cylinder_number = select;
   cylinder_selected = TRUE;
   retval = write(fd, "cylinder selected, now input position", 37 +1);
 }
 else{
   in_string[curs_pstn] = (char) select;
   in_string[++curs_pstn] = '\0';
 }
 break;

case '7':
 if (cylinder_selected == FALSE){
   cylinder_number = select;
   cylinder_selected = TRUE;
   retval = write(fd, "cylinder selected, now input position", 37 +1);
 }
 else{
   in_string[curs_pstn] = (char) select;
   in_string[++curs_pstn] = '\0';
 }
 break;

case '8':
 if (cylinder_selected == FALSE){
   cylinder_number = select;
   cylinder_selected = TRUE;
   retval = write(fd, "cylinder selected, now input position", 37 +1);
 }
 else{
   in_string[curs_pstn] = (char) select;
   in_string[++curs_pstn] = '\0';
 }
 break;

case '9':
 if (cylinder_selected == FALSE){
   retval = write(fd, "invalid cylinder number", 23 + 1);
   menu = MAIN;
   new_menu = TRUE;
   cylinder_selected = FALSE;
   cylinder_number = 0;
   break;
 }
 else{
   in_string[curs_pstn] = (char) select;
   in_string[++curs_pstn] = '\0';
 }
 break;

case '0':
 if (cylinder_selected == FALSE){//invalid cylinder number
   retval = write(fd, "invalid cylinder number", 23 + 1);
   menu = MAIN;
   new_menu = TRUE;
```

```
        cylinder_selected = FALSE;
        cylinder_number = 0;
        break;
      }
      else{
        in_string[curs_pstn] = (char) select;
        in_string[++curs_pstn] = '\0';
      }
      break;

     case CR:

      integer_string = atoi(in_string);

      if ( 0 <= integer_string && integer_string <= 255 && '1'<=
cylinder_number && cylinder_number <= '8' ){
        //run the rest of the routine

        //first store values to memory
        if (cylinder_number == '1')
          shmem-> despstn_cyl1 = integer_string;
        else if (cylinder_number == '2')
          shmem-> despstn_cyl2 = integer_string;
        else if (cylinder_number == '3')
          shmem-> despstn_cyl3 = integer_string;
        else if (cylinder_number == '4')
          shmem-> despstn_cyl4 = integer_string;
        else if (cylinder_number == '5')
          shmem-> despstn_cyl5 = integer_string;
        else if (cylinder_number == '6')
          shmem-> despstn_cyl6 = integer_string;
        else if (cylinder_number == '7')
          shmem-> despstn_cyl7 = integer_string;
        else if (cylinder_number == '8')
          shmem-> despstn_cyl8 = integer_string;

        //now send out command to HC11
        if (integer_string >= 100){//just output cylinder position
           sprintf(output_string, "$C%c%s#", (char)cylinder_number, in_string);
        }
        else if (10 <= integer_string && integer_string < 100){//append a zero
to the beginng of the position
          sprintf(output_string, "$C%c0%s#", (char)cylinder_number, in_string);
        }
        else if (integer_string < 10){//append two zeros to the begining of the
position
          sprintf(output_string, "$C%c00%s#", (char)cylinder_number,
in_string);

        }
        sprintf(speak_buf, "moving cylinder %c, to position %s",
(char)cylinder_number, in_string);
        retval = write(fd, speak_buf, strlen(speak_buf));

        writeBuf( ioport, output_string, OUTPUT_LENGTH);
      }//ends if (0<=atoi(in_string ) <= ......


      else{//invalid input so do nothing
          retval = write(fd, "invalid input", 13 + 1);
      }
      menu = MAIN;
      new_menu = TRUE;
      cylinder_selected = FALSE;
```

```
       cylinder_number = 0;
       curs_pstn = 0;
       break;


      }//ends read keyboard for menu 2

    }//ends if menu == menu2
    else if (menu == MENU3) {//run presentation
      printMenu3Data();

      if (new_menu){
       printMenu3();
       retval = write(fd, "Press return to begin demonstration or escape to
return to the main menu", 72 + 1);
       new_menu = FALSE;
      }

      select = readKeyboard();

      switch (select){
      case ERR:
       //do nothing
       break;

      case ESC:
       //return to main menu
       menu = MAIN;
       new_menu = TRUE;
       break;

      case CR:
       //run presentation
       runPresentation();
       menu = MAIN;
       new_menu = TRUE;
       break;


      }//ends keyboard input for menu3

    }//ends if menu = menu3

  }//ends main while loop

}//ends program
```

C.6 Menulib.c

```
/*-----------------------------------------------------*
 *Program    : menuLib.c                               *
 *Version    : PC/104-1.0                              *
 *Programmer : Scott Kanowitz                          *
 *Contracter : Machine Intelligence Labroatory         *
 *             University of Florida                   *
 *Project    : Pneuman                                 *
 *Date       : 4/5/2000                                *
 *-----------------------------------------------------*
 * Description : Contains library routines used by     *
 *               menu.c                                *
 *-----------------------------------------------------*/


#include "menuLib.h"
#include "user.h"
#include "portOpsLib.h"

extern struct shrd_mem *shmem;

void printMain(void){
  clear();
  move(1,0);
  printw("%s", "+-------------------------------------------------------------
----------------+\n");
  move(2,0); printw("%s", "+");
  move(2,17);
  printw("%s", "Pneuman User Control and Data Analysis Program");
  move(2,79); printw("%s", "+");
  move(3,0); printw("%s", "+");
  move(3,35);
  printw("%s", "Main Menu");
  move(3,79); printw("%s", "+");
  move(4,0);
  printw("%s", "+-------------------------------------------------------------
----------------+\n");
  move(6,2);
  printw("%s", "Command Menus:  1. Send Cmd String  2. Set Cyl Pstn  3.
Demonstration");
  move(9,2);
  printw("%s", "CYL");
  move(9,9);
  printw("%s", "DES PSTN");
  move(9,20);
  //printw("%s", "CUR PSTN");
  move(10,2); printw("%s", "1    |          |         |");
  move(11,2); printw("%s", "2    |          |         |");
  move(12,2); printw("%s", "3    |          |         |");
  move(13,2); printw("%s", "4    |          |         |");
  move(14,2); printw("%s", "5    |          |         |");
  move(15,2); printw("%s", "6    |          |         |");
  move(16,2); printw("%s", "7    |          |         |");
  move(17,2); printw("%s", "8    |          |         |");
  refresh();
}//end of printMain

void printMainData(void)
{
  move(10,9); printw("%d  ", shmem-> despstn_cyl1);
  move(11,9); printw("%d  ", shmem-> despstn_cyl2);
  move(12,9); printw("%d  ", shmem-> despstn_cyl3);
  move(13,9); printw("%d  ", shmem-> despstn_cyl4);
```

```
  move(14,9); printw("%d  ", shmem-> despstn_cyl5);
  move(15,9); printw("%d  ", shmem-> despstn_cyl6);
  move(16,9); printw("%d  ", shmem-> despstn_cyl7);
  move(17,9); printw("%d  ", shmem-> despstn_cyl8);
  move(22,1);
  // printw("%s", "\n");
  refresh();
}

void printMenu1(void){
  clear();
  move(1,0);
  printw("%s", "+-----------------------------------------------------------
----------------+\n");
  move(2,0); printw("%s", "+");
  move(2,17);
  printw("%s", "Pneuman User Control and Data Analysis Program");
  move(2,79); printw("%s", "+");
  move(3,0); printw("%s", "+");
  move(3,29);
  printw("%s", "Send Command String");
  move(3,79); printw("%s", "+");
  move(4,0);
  printw("%s", "+-----------------------------------------------------------
----------------+\n");
  move(6,2);
  printw("%s", "Input string and press return. Press ESC to return to main
menu");
  move(9,2);
  printw("%s", "CYL");
  move(9,9);
  printw("%s", "DES PSTN");
  move(9,20);
  //printw("%s", "CUR PSTN");
  move(10,2); printw("%s", "1    |          |         |");
  move(11,2); printw("%s", "2    |          |         |");
  move(12,2); printw("%s", "3    |          |         |");
  move(13,2); printw("%s", "4    |          |         |");
  move(14,2); printw("%s", "5    |          |         |");
  move(15,2); printw("%s", "6    |          |         |");
  move(16,2); printw("%s", "7    |          |         |");
  move(17,2); printw("%s", "8    |          |         |");
  refresh();
}//end of printMenu1


void printMenu1Data(void)
{
  move(10,9); printw("%d  ", shmem-> despstn_cyl1);
  move(11,9); printw("%d  ", shmem-> despstn_cyl2);
  move(12,9); printw("%d  ", shmem-> despstn_cyl3);
  move(13,9); printw("%d  ", shmem-> despstn_cyl4);
  move(14,9); printw("%d  ", shmem-> despstn_cyl5);
  move(15,9); printw("%d  ", shmem-> despstn_cyl6);
  move(16,9); printw("%d  ", shmem-> despstn_cyl7);
  move(17,9); printw("%d  ", shmem-> despstn_cyl8);
  move(22,1);
  // printw("%s", "\n");
  refresh();
}//end of print menu 1 data



void printMenu2(void){
```

```
  clear();
  move(1,0);
  printw("%s", "+-----------------------------------------------------------------
---------------+\n");
  move(2,0); printw("%s", "+");
  move(2,17);
  printw("%s", "Pneuman User Control and Data Analysis Program");
  move(2,79); printw("%s", "+");
  move(3,0); printw("%s", "+");
  move(3,28);
  printw("%s", "Set Cylinder Position");
  move(3,79); printw("%s", "+");
  move(4,0);
  printw("%s", "+-----------------------------------------------------------------
---------------+\n");
  move(6,2);
  printw("%s", "Select cylinder number then input new position");
  move(9,2);
  printw("%s", "CYL");
  move(9,9);
  printw("%s", "DES PSTN");
  move(9,20);
  //printw("%s", "CUR PSTN");
  move(10,2); printw("%s", "1    |          |         |");
  move(11,2); printw("%s", "2    |          |         |");
  move(12,2); printw("%s", "3    |          |         |");
  move(13,2); printw("%s", "4    |          |         |");
  move(14,2); printw("%s", "5    |          |         |");
  move(15,2); printw("%s", "6    |          |         |");
  move(16,2); printw("%s", "7    |          |         |");
  move(17,2); printw("%s", "8    |          |         |");
  refresh();
}//end of printMenu2


void printMenu2Data(void)
{
  move(10,9); printw("%d  ", shmem-> despstn_cyl1);
  move(11,9); printw("%d  ", shmem-> despstn_cyl2);
  move(12,9); printw("%d  ", shmem-> despstn_cyl3);
  move(13,9); printw("%d  ", shmem-> despstn_cyl4);
  move(14,9); printw("%d  ", shmem-> despstn_cyl5);
  move(15,9); printw("%d  ", shmem-> despstn_cyl6);
  move(16,9); printw("%d  ", shmem-> despstn_cyl7);
  move(17,9); printw("%d  ", shmem-> despstn_cyl8);
  move(22,1);
  // printw("%s", "\n");
  refresh();
}//end of print menu 2 data

void printMenu3(void){
  clear();
  move(1,0);
  printw("%s", "+-----------------------------------------------------------------
---------------+\n");
  move(2,0); printw("%s", "+");
  move(2,17);
  printw("%s", "Pneuman User Control and Data Analysis Program");
  move(2,79); printw("%s", "+");
  move(3,0); printw("%s", "+");
  move(3,29);
  printw("%s", "Self Demonstration");
  move(3,79); printw("%s", "+");
  move(4,0);
```

```
  printw("%s", "+-----------------------------------------------------------------
----------------+\n");
  move(6,2);
  printw("%s", "Press return to begin demonstration or ESC to return to main
menu");
  move(9,2);
  printw("%s", "CYL");
  move(9,9);
  printw("%s", "DES PSTN");
  move(9,20);
  //printw("%s", "CUR PSTN");
  move(10,2); printw("%s", "1    |          |         |");
  move(11,2); printw("%s", "2    |          |         |");
  move(12,2); printw("%s", "3    |          |         |");
  move(13,2); printw("%s", "4    |          |         |");
  move(14,2); printw("%s", "5    |          |         |");
  move(15,2); printw("%s", "6    |          |         |");
  move(16,2); printw("%s", "7    |          |         |");
  move(17,2); printw("%s", "8    |          |         |");
  refresh();
}//end of printMenu3



void printMenu3Data(void)
{
  move(10,9); printw("%d  ", shmem-> despstn_cyl1);
  move(11,9); printw("%d  ", shmem-> despstn_cyl2);
  move(12,9); printw("%d  ", shmem-> despstn_cyl3);
  move(13,9); printw("%d  ", shmem-> despstn_cyl4);
  move(14,9); printw("%d  ", shmem-> despstn_cyl5);
  move(15,9); printw("%d  ", shmem-> despstn_cyl6);
  move(16,9); printw("%d  ", shmem-> despstn_cyl7);
  move(17,9); printw("%d  ", shmem-> despstn_cyl8);
  move(22,1);
  // printw("%s", "\n");
  refresh();
}//end of print menu 3 data

int readKeyboard( void )
{
  int nbytes ;

  ioctl( 0, FIONREAD, &nbytes ) ;

  if ( nbytes == 0 )
    return( ERR ) ;
  else
    return( getch() ) ;

}  // end of readKeyboard()... //

void runPresentation(void){
  int ret;
  char buf[32];
  extern int fd;
  extern int ioport;
  extern struct shrd_mem *shmem;

  ret = write(fd, MESS2, strlen(MESS2) +1);
  ret = read(fd, buf, 1); //wait until finished talking

  sleep(1);
```

```
ret = write(fd, MESS3, strlen(MESS3) +1);
ret = read(fd, buf, 1); //wait until finished talking

sleep(1);

ret = write(fd, MESS4, strlen(MESS4) +1);
ret = read(fd, buf, 1); //wait until finished talking

sleep(1);

ret = write(fd, MESS45, strlen(MESS45) +1);
ret = read(fd, buf, 1); //wait until finished talking

sleep(1);

ret = write(fd, MESS5, strlen(MESS5) +1);
ret = read(fd, buf, 1); //wait until finished talking

writeBuf(ioport, "$C1150#", 7);//set cylinder 1 to position 150
shmem-> despstn_cyl1 = 150;
printMenu3Data();

sleep(3);

ret = write(fd, MESS6, strlen(MESS6) +1);
ret = read(fd, buf, 1); //wait until finished talking

writeBuf(ioport, "$C1100#", 7);//set cylinder 1 to position 100
shmem-> despstn_cyl1 = 100;
printMenu3Data();

sleep(3);

ret = write(fd, MESS7, strlen(MESS7) +1);
ret = read(fd, buf, 1); //wait until finished talking

writeBuf(ioport, "$C1220#", 7);//set cylinder 1 to position 220
shmem-> despstn_cyl1 = 220;
printMenu3Data();

sleep(3);

ret = write(fd, MESS8, strlen(MESS8) +1);
ret = read(fd, buf, 1); //wait until finished talking

writeBuf(ioport, "$C1050#", 7);//set cylinder 1 to position 50
shmem-> despstn_cyl1 = 50;
printMenu3Data();

ret = write(fd, MESS9, strlen(MESS9) +1);
sleep(1);//wait 1 second then,
writeBuf(ioport, "$C1200#", 7);//set cylinder 1 to position 200
shmem-> despstn_cyl1 = 200;
printMenu3Data();

sleep(3);

ret = write(fd, MESS10, strlen(MESS10) +1);
ret = read(fd, buf, 1); //wait until finished talking

sleep(2);

ret = write(fd, MESS11, strlen(MESS11) +1);
ret = read(fd, buf, 1); //wait until finished talking
```

```
  sleep(2);

  ret = write(fd, MESS12, strlen(MESS12) +1);
  ret = read(fd, buf, 1); //wait until finished talking

}
```

C.7 Menulib.h

```
/*-----------------------------------------------------*
 *Program    : menuLib.h                               *
 *Version    : PC/104-1.0                              *
 *Programmer : Scott Kanowitz                          *
 *Contracter : Machine Intelligence Labroatory         *
 *             University of Florida                   *
 *Project    : Pneuman                                 *
 *Date       : 4/5/2000                                *
 *-----------------------------------------------------*
 * Description : Header file for menuLib.c and menu.c *
 *-----------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <curses.h>
#include <ctype.h>
#include <time.h>

#define MAIN 0
#define MENU1 1
#define MENU2 2
#define MENU3 3

#define CMD_LENGTH 10
#define CR 0x0A
#define ESC 0x1B
#define OUTPUT_LENGTH 7

//presentation strings
#define MESS1 "Press return to begin autonomous control or escape to return to
main menu\0010I\r"
#define MESS2 " This is a demonstration of the control abilities of this
system\0010I\r"
#define MESS3 "Each cylinder uses a p i d controller to maintain
position\0010I\r"
#define MESS4 "there are 255 different positions available along the length of
the cylinder\0010I\r"
#define MESS45 "I can set each cylinder position depending on the needs of the
system\0010I\r"
#define MESS5 "if I want cylinder 1 at position 150 I can send it
there\0010I\r"
#define MESS6 "I can also send it back to position 100\0010I\r"
#define MESS7 "I will set the cylinder to position 220\0010I\r"
#define MESS8 "now I will set the cylinder to position 50 and change the
position while it is moving\0010I\r"
#define MESS9 "back to position 200\0010I\r"
#define MESS10 "I can control all 8 cylinders the same way and at the same
time\0010I\r"
#define MESS11 "thank you for watching my demonstration\0010I\r"
#define MESS12 "I will now return to the main menu\0010I\r"

//function declarations
void printMain(void);
void printMainData(void);
void printMenu1(void);
void printMenu1Data(void);
void printMenu2(void);
void printMenu2Data(void);
int readKeyboard(void);
```

C.8 Src Makefile

```
HOME = /home/pneuman/lincode

SRC = $(HOME)/src

INCLUDE = $(HOME)/include

LIB = $(HOME)/lib

EXE = $(HOME)/bin

CC = gcc



all: libs $(EXE)/spwnproc $(EXE)/menu

libs:
	cd $(LIB);   make;

$(EXE)/spwnproc: spwnproc.o $(LIB)/spwnprocLib.o
	$(CC) -o $(EXE)/spwnproc $(LIB)/spwnprocLib.o spwnproc.o

spwnproc.o: spwnproc.c $(INCLUDE)/spwnprocLib.h $(INCLUDE)/user.h
	$(CC) -c -I$(INCLUDE) spwnproc.c

$(EXE)/menu: menu.o $(LIB)/cursPstnLib.o $(LIB)/menuLib.o
	$(CC) -o $(EXE)/menu $(LIB)/cursPstnLib.o \
	$(LIB)/menuLib.o $(LIB)/portOpsLib.o menu.o -lncurses

menu.o: menu.c $(INCLUDE)/portOpsLib.h $(INCLUDE)/menuLib.h \
	$(INCLUDE)/cursPstnLib.h  $(INCLUDE)/user.h
	$(CC) -c -I$(INCLUDE) menu.c
```

C.9 Lib Makefile

```
HOME = /home/pneuman/lincode

INCLUDE = $(HOME)/include

LIB = $(HOME)/lib

EXE = $(HOME)/bin

CC = gcc



all:  spwnprocLib.o cursPstnLib.o menuLib.o portOpsLib.o

spwnprocLib.o: spwnprocLib.c $(INCLUDE)/spwnprocLib.h $(INCLUDE)/user.h
	$(CC) -c -I$(INCLUDE) spwnprocLib.c

cursPstnLib.o: cursPstnLib.c $(INCLUDE)/cursPstnLib.h
	$(CC) -c -I$(INCLUDE) cursPstnLib.c

menuLib.o: menuLib.c $(INCLUDE)/menuLib.h
	$(CC) -c  -I$(INCLUDE) menuLib.c

portOpsLib.o: portOpsLib.c $(INCLUDE)/portOpsLib.h
	$(CC) -c -I$(INCLUDE) portOpsLib.c
```