Automated Industrial Wind Tunnel Controller

by

Nick DeTrempe

Daniel Monahan

Advisors:

Dr. Aleksander Malinowski

Dr. Scott Post

Dr. Marin Morris

Abstract

Automated Industrial Wind Tunnel Project involves adding a microcontroller and computer to an existing wind tunnel as control devices. There are three aspects of the project. The first is the microcontroller. It implements safety and control functions for the wind tunnel as well as sending data to the adjunct server. The next part is the server. It acts as a host for remote users, allowing them to connect and collect data. It also relays control signals from users to the microcontroller. The third part is the client program. The application displays data for users to see, provides an interface for them to interpret the data and send control signals. The client also allows the data to be collected and exported into a spreadsheet. The entire system provides users with a way to remotely view and control the wind tunnel at Bradley University.

Table of Contents

Project Description

The Automated Industrial Wind Tunnel Controller project involves integrating a microcontroller into the pre-existing wind tunnel system and utilizing a server-client approach to control the wind tunnel either remotely or locally using a computer interface. Using a LabView based client, users can remotely manipulate the actuators on the wind tunnel and the wind tunnel itself to create different testing conditions. Data collected at the Wind Tunnel are relayed back to the client to be displayed to the user, and to be exported to a spreadsheet to allow for saving of the testing data. The user can also monitor the wind tunnel visually using four webcams that are placed strategically to allow the most informative viewing experience.

Previous Work Completed

Two design iterations have been made in the past. This project is founded both on information already collected from the users of the tunnel, and lessons learned from the performance of the previous designs.

2010: Benjamin Morrison and Michael Firman's Project

- Performed system analysis on the wind tunnel.
- Acquired solid state relays to control damper and blower through a microcontroller.
- Designed H-Bridge circuitry to control linear actuators through a microcontroller.

2011: Adam Green's Project

- Performed system analysis on the wind tunnel.
- Acquired solid state relays to control damper and blower through a microcontroller.
- Designed H-Bridge circuitry to control linear actuators through a microcontroller.

Changes to Previous Designs

- Removal of the National Instruments digital to analog converter.
- Microcontroller automatic shutdown to protect components.
- Electronic and manual control switching method.
- Addition of ambient air pressure sensor measurement.

Current System Concept

Our system contains 3 main parts: the microcontroller, the computer with the server, and a remote client. The microcontroller is a Silicon Labs C8051F120. It has an 8-bit 8051 processor and an 8 channel analog to digital converter. This allows us to collect data from sensors on the wind tunnel and to send control signals to the control interfaces on the wind tunnel. Communications between the microcontroller and the server take place via an RS-232 serial interface. The overall system block diagram is shown in Figure 1.

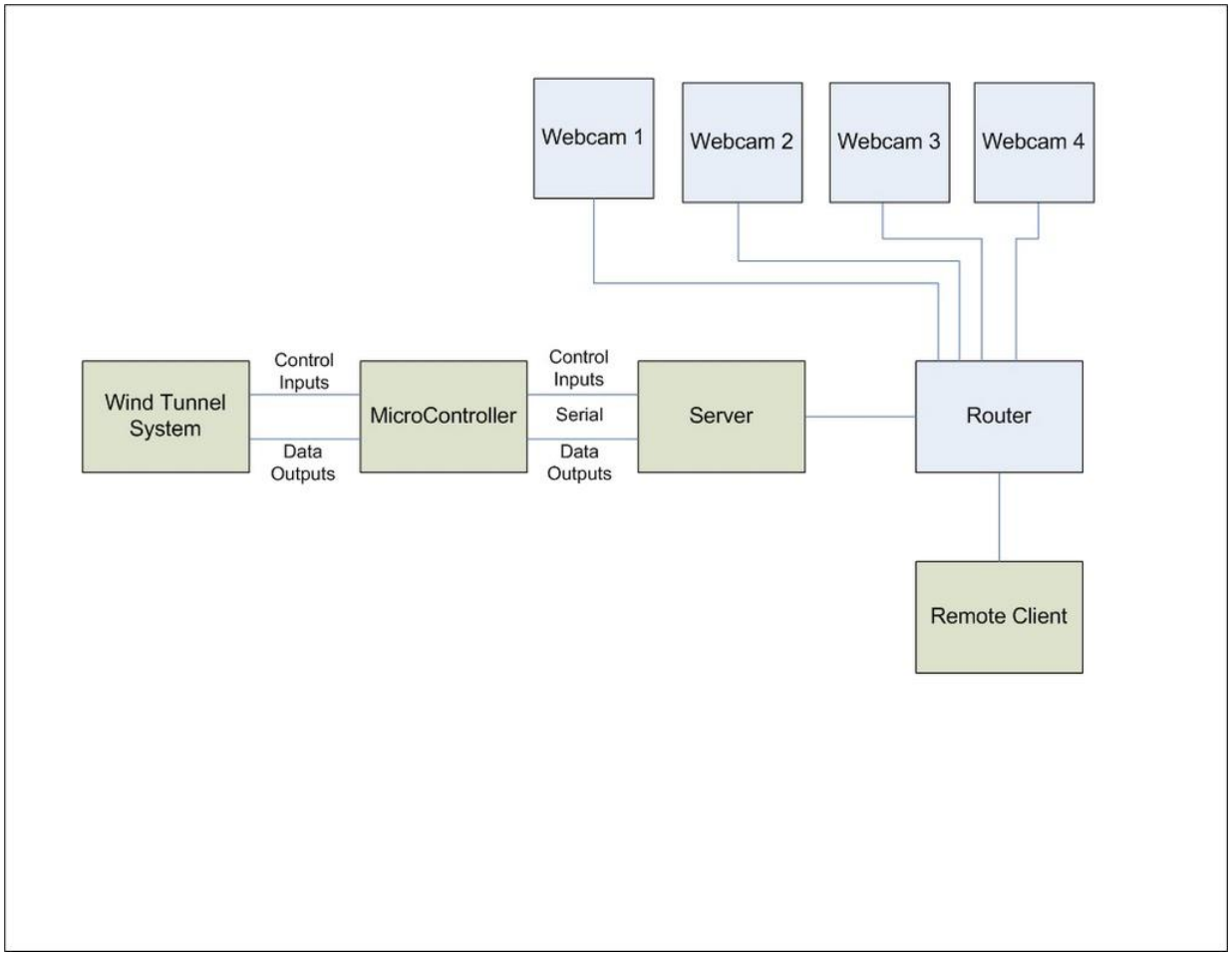


Figure 1. Overall System Block Diagram

Wind Tunnel Controls Hardware

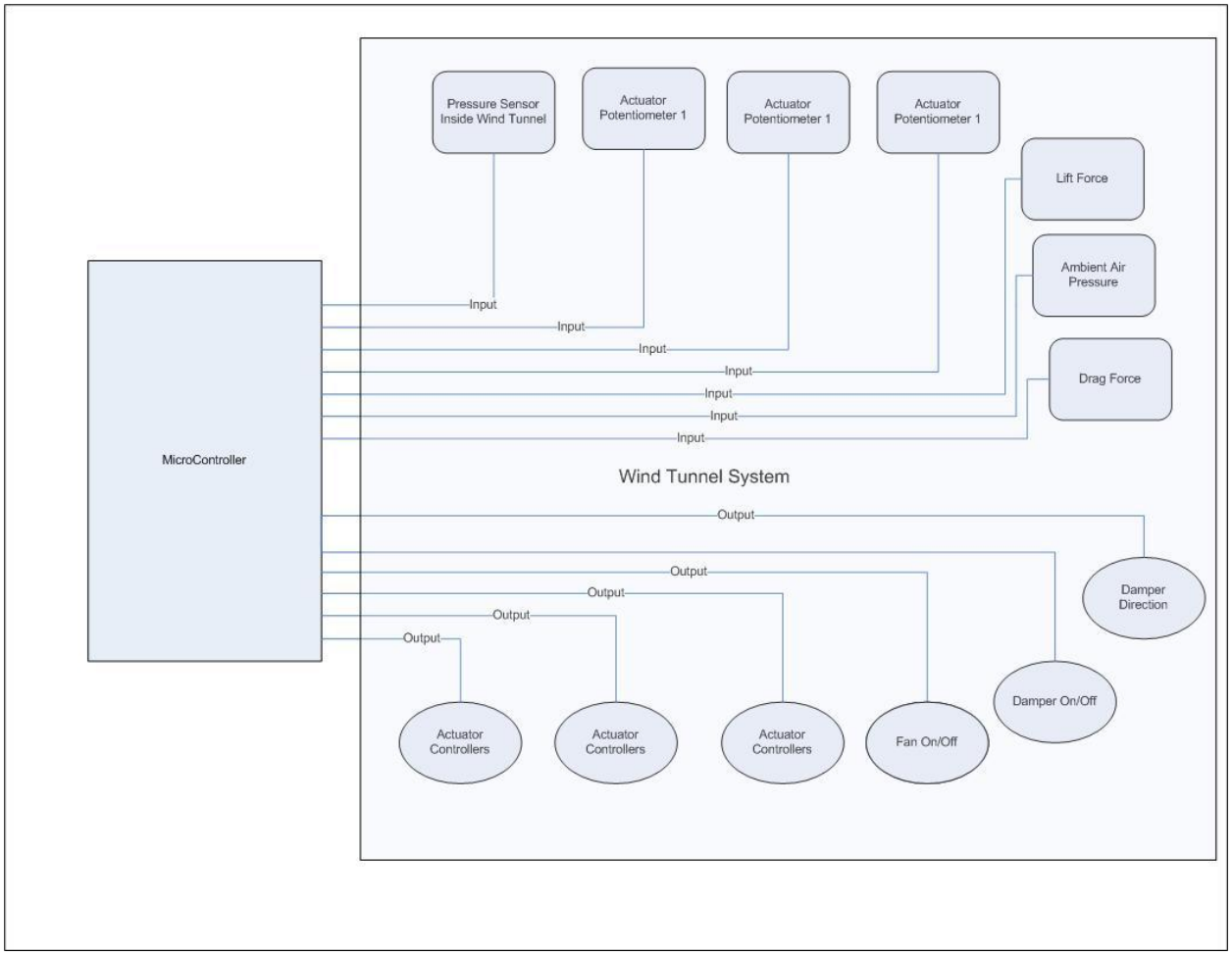


Figure 2. Block diagram for wind tunnel hardware

Figure 2 shows the block diagram of all hardware that is used for the microcontroller to interact with the wind tunnel. The outputs on the bottom of the figure are by the microcontroller to turn on the blower, adjust the position of the actuator and adjust the position of the three linear actuators. The inputs on the top of the figure are used to receive data from various sensors. Lift and drag force on the target object are received by existing sensors. The ambient air pressure and air pressure inside the tunnel are used to calculate wind speed. Each actuator has a built in potentiometer that is used to measure actuator position. In addition to what is displayed in figure 2, a 12V power supply is available for use.

The wind tunnel itself is controlled by two components: a blower and a damper. The blower is turned on by two basic single pole single throw on/off switch. The damper is used to control the wind speed. It's position is controlled by a double pole, double throw, center off switch. The double pole double

throw functionality of this switch allows for the polarity of the power delivered to the motor to be switched, giving bi-directional control. The center off functionality allows for the switch to be placed in a position where no power is delivered to the motor, leaving the damper stationary.

Electronically, the blower is controlled by an Omron G3NA-225B-DC5-24 single pole single throw relay. This relay is controlled directly by a port from the microcontroller. The damper is controlled by a Dayton 1EGX1 double pole, double throw relay. The use of this relay created a couple of issues. The first is that the relay is controlled a 10V signal and therefore cannot be directly controlled by the microcontroller. The second is that the this relay doesn't have a center off position like the manual switch does. Therefore, if this relay is directly connected to the damper motor then the motor will always be energized. The first issue was resolved with a G3NA-D210B-DC5-24 DC to DC relay. This is controlled by a port on the microcontroller and is used to route 12 volts to the double pole double throw switch. This relay is used by the microcontroller to control the direction of the damper motor. The second issue is resolved with another G3NA-225B-DC5-24 single pole single throw relay. This relay is controlled directly by the microcontroller and delivers AC power to the double pole double throw switch. This relay is used to control turn the damper motor on or off. These components were purchased in the previous project years but not implemented until this year.

At the time of the writing of this report, only one linear actuator was mounted in the wind tunnel. However, the system was designed so that three actuators could be used. Should one or two more actuators be added in the future, they would just need to be plugged into existing ports to be used. Each of these linear actuators are controlled by an LMD-18200T H-Bridge. Each H-Bridge is receiving a PWM input and a direction input from the microcontroller. This allows for the actuators to be extended and retracted by the microcontroller. The H-Bridges are powered by a 12 volt power supply that is located in the wind tunnel lab. Between the H-Bridge outputs and actuator inputs are 3 ohm 16 watt power resistors to help with power dissipation. These H-bridge circuits were designed and implement in the previous year.

The position of each actuator is measured by a potentiometer that is placed in a voltage divider. In order to read a voltage from the potentiometer, a reference voltage must be delivered to this circuit. The reference voltage that is used is 3.84V and is generated by using a voltage divider circuit which is powered by the 12V power supply. This circuit was designed and implemented in the previous year. In order to convert the measured voltage to a particular actuator length. The actuator was extended to several different lengths and the voltage was measured by the A/D converter on the microcontroller while the length was measured with a ruler. This data was plotted in Microsoft Excel and linear regression was performed to get a voltage to length function. Figure 3 shows this data.

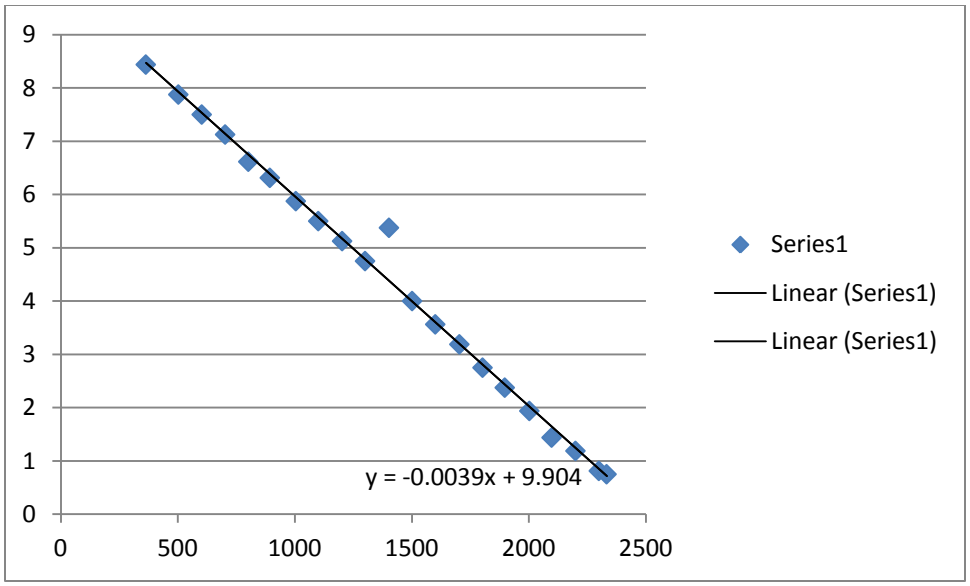


Figure 3. Linear regression of voltage to length function for linear actuators

Lift and drag are measured with two LM207 force sensors. These sensors are currently connected to conditioning modules which output measured values to a seven segment display. In addition to outputting values to a display, they each output a voltage which changes linearly with respect to the measured value. These voltages are measured by the A/D converter in the microcontroller. In order to get a voltage to force function, both sensors were given various different loads. For each load, the measured voltage and displayed force were documented. As with the actuator, the recorded data was entered into Microsoft Excel and a linear regression was used to get a voltage to force function. Figures 4 and 5 show the lift and drag functions, respectively.

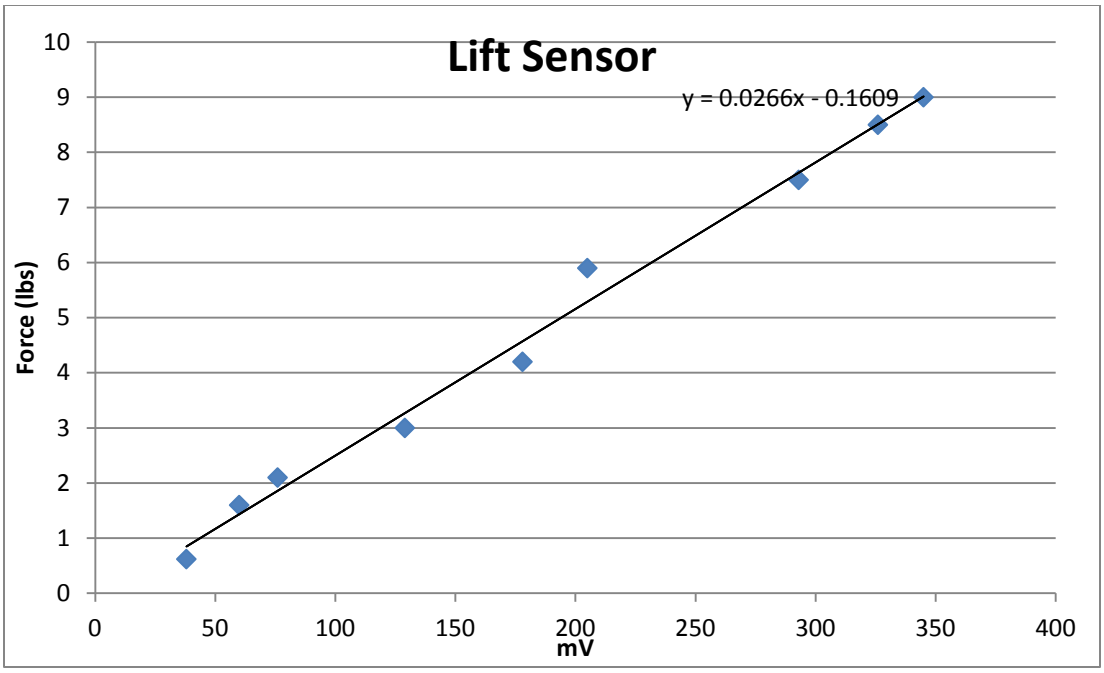


Figure 4. Lift sensor characteristics

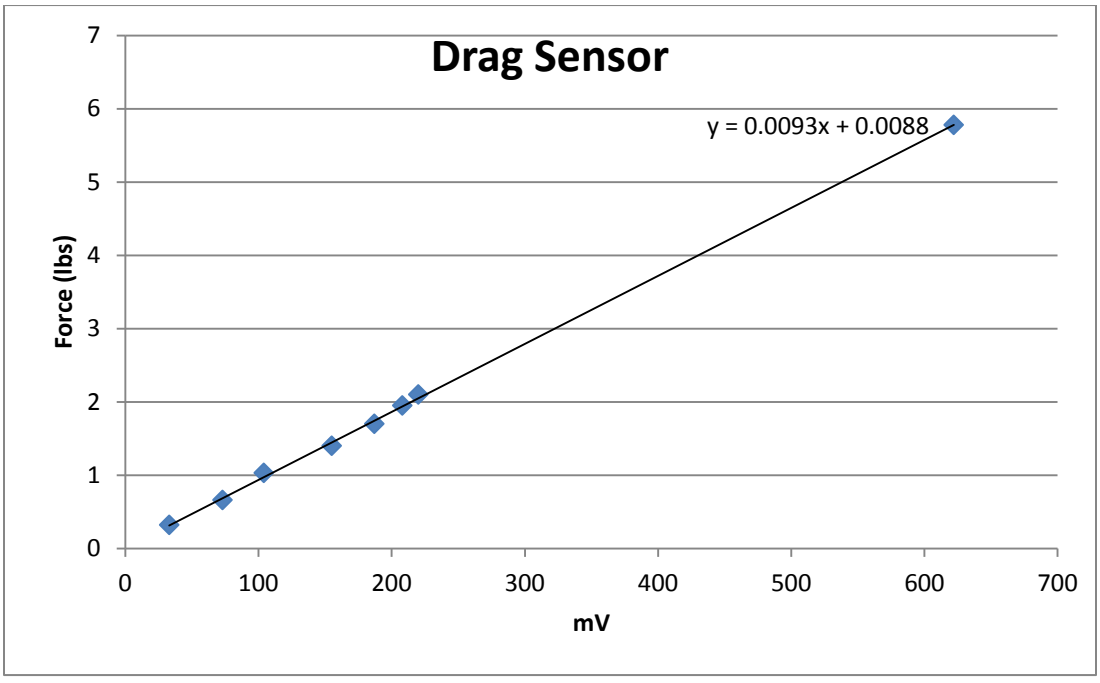


Figure 5. Drag sensor characteristics

The wind speed is measured with two Setra pressure transducers. A Setra model 276 is used to measure ambient air pressure outside of the tunnel. A Setra model 320 pressure sensor is used to measure pressure inside the tunnel. The function for voltage to pressure for the ambient air pressure was provided by the datasheet and is shown in equation 1. The voltage range for this sensor is 0.5V to 4.5V and the pressure range is 0.8 bar to 1.2 bar. These voltages could not be delivered directly to the microcontroller as it has a max range of 2.2V. This problem was solved by a simple voltage divider. The voltage divider was designed two 5% resistors: a 39kOhm and 62kOhm. Each of these resistors were measured with a Fluke multi-meter and the measured values were: 38.56kOhms and 61.63kOhms. Using these values, a transfer function of 0.385 was found. Equation 2 shows the updated voltage to pressure function for voltages measured after the resistor.

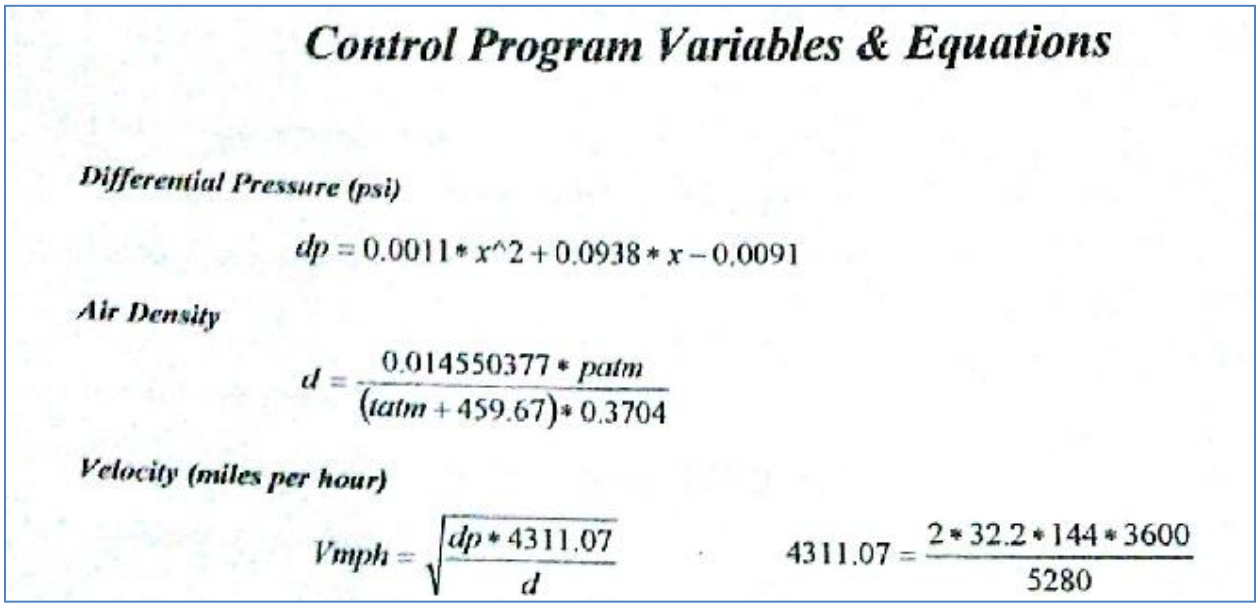
$$P_{br} = 0.06 * v + 0.794$$

Equation 1. voltage to pressure function of Setra 276 before the voltage divider.

$$P_{br} = 0.1558 * v + 0.794$$

Equation 2. voltage to pressure function of Setra 276 after the voltage divider.

The wind speed is then calculated by using the barometric pressure calculated from the Setra 276 and the raw measured voltage from the Setra 320. The equations used to convert the two input values to a wind speed are shown in figure 6. This information was provided by an earlier project performed by the mechanical engineering lab.



Control Program Variables & Equations

Differential Pressure (psi)

$$dp = 0.0011 * x\text{^}2 + 0.0938 * x - 0.0091$$

Air Density

$$d = \frac{0.014550377 * patm}{(iatm + 459.67) * 0.3704}$$

Velocity (miles per hour)

$$Vmph = \sqrt{\frac{dp * 4311.07}{d}} \qquad 4311.07 = \frac{2 * 32.2 * 144 * 3600}{5280}$$

Figure 6. Equations used to calculate wind speed

Microcontroller

A Silicon Labs C8051F120 microcontroller was used to control all relays as well as read all data from sensors. This microcontroller contains a nine channel A/D converter which allows for all of the seven sensors to be measured. The A/D converter is interrupt driven at a 600 Hz rate. Each interrupt generates a conversion on a different channel in a round robin fashion. At a 600 Hz rate for 7 channels, the sample rate for each channel is approximately 86 Hz. Only steady state data is being recorded so an 86 Hz sample rate is acceptable for this application.

All communications between the microcontroller and the server take place over an RS-232 serial port connection. These communications are sent via strings of ASCII characters. Measured values are sent from the microcontroller to the server while commands are sent from the server to the microcontroller. The measured values are sent in the following format: "xnnnn" where 'x' denotes the device that is measured and "nnnn" denotes the specific value in miliVolts that was measured.

The main loop of the microcontroller uses polling to check for updated values from the A/D converter as well as messages from the microcontroller. Figure 7 shows the flowchart for the main loop. Upon seeing a flag that a new sample is ready, the processor obtains that sample, averages it with the most recent 15 values for filtering purposes and then sends the value to the server. Upon seeing a flag that a new character has been received from the server, the processor enters a loop to continue receiving characters until an end of line character or null character is received. If an acceptable command has been received, the command is entered into a FIFO array which holds 10 commands. Figure 8 shows a list of acceptable commands. The main loop will then poll the FIFO array to see if any commands are ready to be executed, if so then it will execute the command.

Executing commands to turn on the blower or move the damper are pretty straight forward, as they only involve switching on a relay. Executing an actuator command is more difficult and time consuming. Timer 4 is used to generate an interrupt which is then used to generate a PWM signal to the H-Bridge. On top of the PWM generation, the timer 4 interrupt routine generates a counter to time how long the PWM signal is being generated. This is needed as a precaution, to ensure that the microcontroller doesn't continually attempt to remove an unresponsive actuator. Should the position of an actuator not change after three seconds or should the actuator not reach it's intended position after 90 seconds, the microcontroller will discontinue the PWM signal and consider the command complete. While the actuator is moving, the microcontroller monitors its position and stops the PWM when it has reached its commanded position.

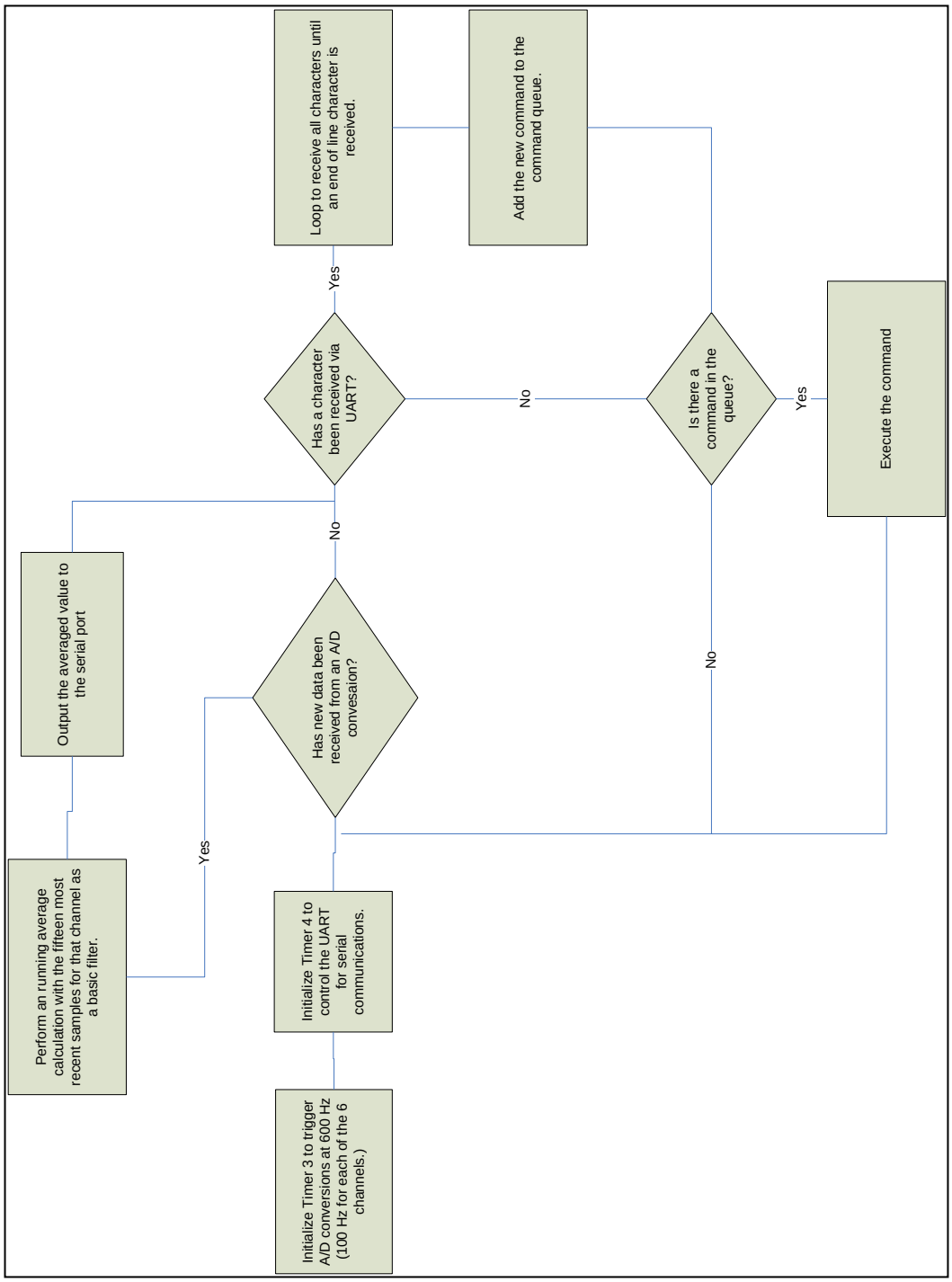


Figure 7. Main loop flowchart for microcontroller software

"BO"	Turn the blower on
"BS"	Turn the blower off
"DO"	Start opening the damper
"DC"	Start closing the damper
"DS"	Stop moving the damper

Figure 8. Acceptable microcontroller commands

Control Output	Port/Pin	Hardware Pin
Actuator 1 direction	P1_1	B-4
Actuator 1 PWM	P3_0	A-10
Actuator 2 direction	P1_2	A-4
Actuator 2 PWM	P3_1	C-9
Actuator 3 direction	P1_3	C-3
Actuator PWM	P3_2	B-9
Tunnel on/off	P3_3	A-9
Damper on/off	P3_4	C-8
Damper direction	P3_5	B-8

Figure 9 microcontroller output pins

Sensor Input	Port/Pin	Hardware Pin
Actuator 1 position	AIN0.0	A-32
Actuator 2 position	AIN0.2	B-31
Actuator 3 position	AIN0.4	C-30
Lift	AIN0.1	C-31
Drag	AIN0.3	A-31
Air pressure inside tunnel	AIN0.5	B-30
Ambient air pressure	AIN0.6	A-30

Figure 10 microcontroller input pins

Java Server

The server component for the project was written using Java. Java seemed the best solution for both capability and ease of development. The server began in C but the need for multiple threads arose and we moved to Java. The server functions as little more than a communication relay between the LabView client and the microcontroller. The server juggles TCP read and write functions as well as reading and writing to the computer's COM port. Currently the server accepts up to ten clients and will send data received from the microcontroller to all of them. Further development of the server includes flagging a particular client as the one in control.

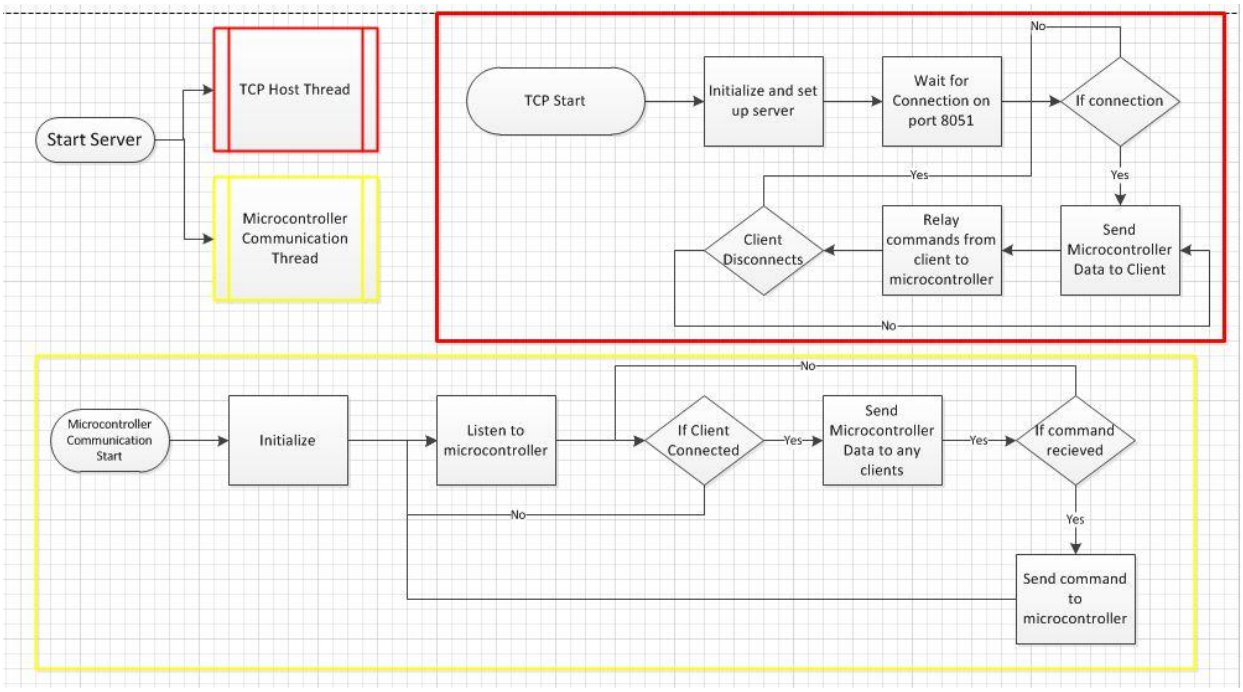


Figure 11: Server Flow Charts

LabView Client Program

The client communicates using TCP to the server on the host machine. The server is written in Java. The client currently has successfully been tested to both send and receive text from the microcontroller. Integration of graphical interfaces to both display data and send commands has also been tested successfully. The code for LabView is “written” graphically. To some degree the code flows like one main program, however using independent while loops, separate threads can essentially be created to allow for independent processes. This allows for more reliable communication.

LabView provides a library of functions that enable developers to more quickly develop more complex code. For example there are numerous TCP communication functions as well as other communication protocols. Hardware interface is another strength of LabView even though we are not utilizing it in this project. LabView also provides the ability to create custom functions, “VI” files, that allow developers to condense and simplify the reading of their Block Diagrams.

The client that has been developed using labview gives users the ability to turn the wind tunnel on and off. Adjust the wind speed manually and automatically, and adjust the actuators. It displays data including Lift and Drag forces, angle, time, and wind speed. This data can be recorded to a file for further analysis.

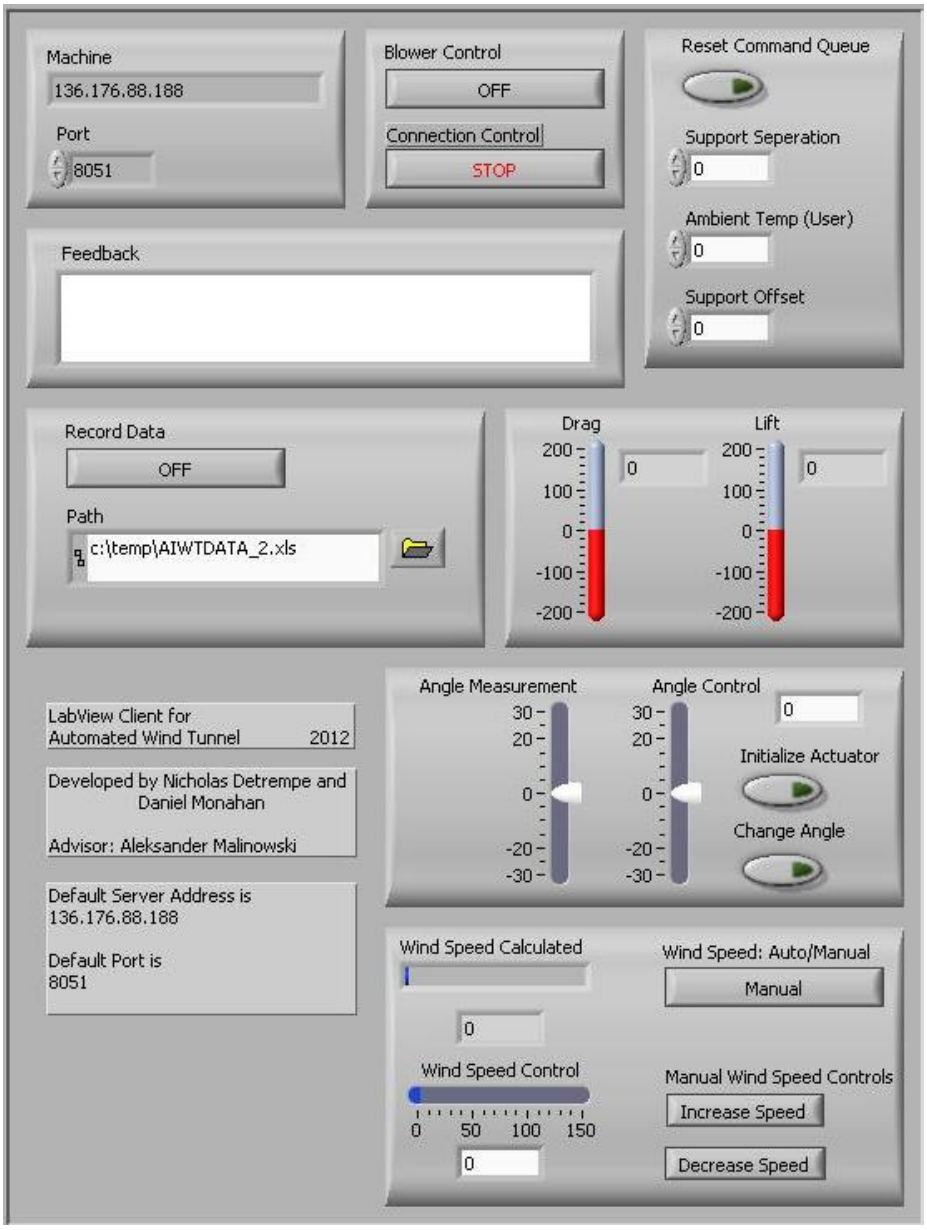


Figure 12: LabView User Interface

Further Project Development

There are various ways that this project can be expanded upon. Further development of each feature is possible. Some key feature improvements are noted below.

- LabView Front end configuration of angle equation: If the target object is changed, the equation to determine the angle usually needs to be changed. Giving the user the ability to adjust the coefficients of the equation would be helpful
- Java Server ability to distinguish clients and give only one control at a time: Currently, the server sends and receives data and commands to all connected clients at the same time, not

differentiating. This could cause conflicts if multiple clients try sending commands to the server at the same time. Counter-acting commands could cause problems for the wind tunnel.

- Further mounting of existing hardware and developing enclosure: The hardware, because of development, is only semi-secured to the wind tunnel and the power supply and hardware switches need to be secured and enclosed properly.
- Mounting Webcams: There are four webcams which can provide users with visual feedback in addition to quantitative feedback the client provides. Mounting these cameras in strategic locations and further developing the wind tunnel website would give remote users a more involved experience.

Conclusion

This project provided a means to not only practice hardware software integration, but also to develop the basis of a system of control, both locally and remotely, of the wind tunnel. While the system is by no means at a production readiness, it does provide control and data recording functionality. The primary needs are for calibration and accuracy improvement.

Appendix A: LabView Block Diagrams

Full LabView Block Diagram

This diagram (Figure A-1) represents the full LabView client code. Each specific block is discussed in more detail below. The client begins by connected to the server after verifying that it has a ip address and port to connect to. When the program is stopped it makes sure the blower is turned off by sending a stop signal and then closes the TCP connection.

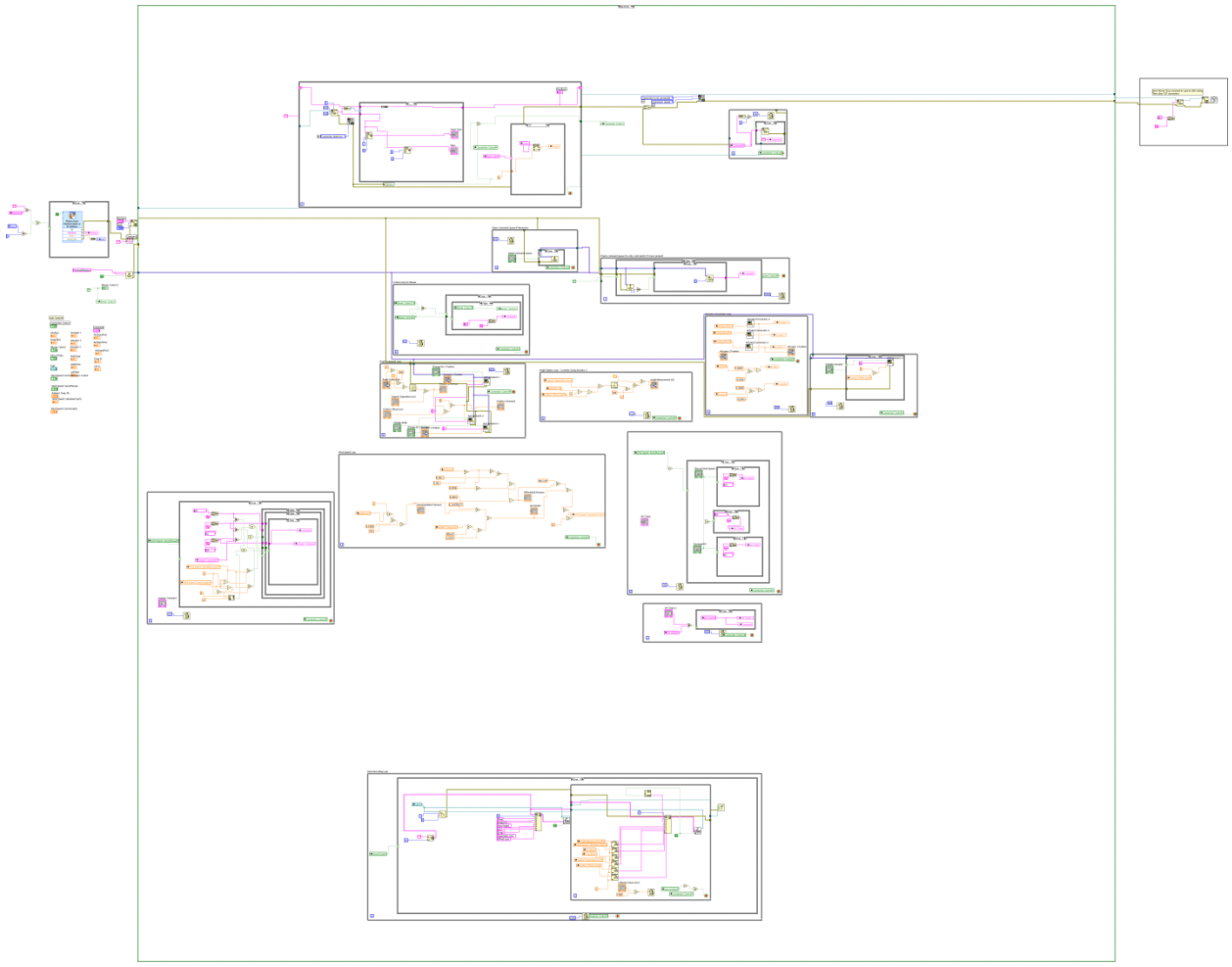


Figure A-1: LabView Client Block Diagram

TCP Receive loop

This loop listens for TCP signals and sorts them into the appropriate field. The client receives a 5 character string relayed by the server from the microcontroller. The first character is the code for the data. The next four characters represent the data itself. The loop splits the string into a one character string and a four character string and then puts the data into a particular field depending on the one character string. Any string combination that is not recognized is output to a separate field the keeps

track of miscellaneous output. This allows users to receive acknowledgement of connection and error messages.

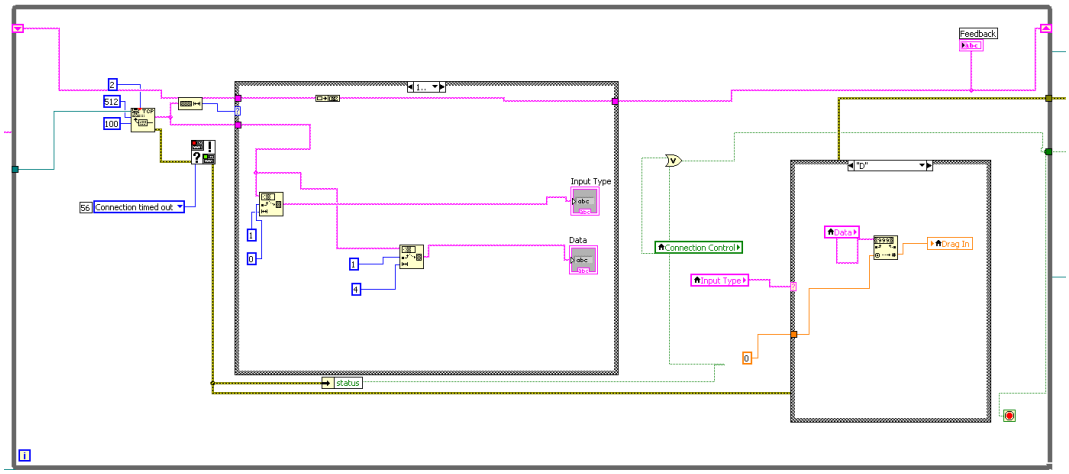


Figure A-2: TCP Reception diagram

String	Signal ID	Data Type
"1XXXX"	1	Actuator 1
"2XXXX"	2	Actuator 2
"3XXXX"	3	Actuator 3
"LXXXX"	L	Lift
"DXXXX"	D	Drag
"SXXXX"	S	Wind Speed Pressure
"PXXXX"	P	Ambient Pressure

Figure A-3: Signal Examples

Command Queue Reset Loop

Clears command queue if it gets too full or the microcontroller queue fills up and it takes too long to respond. Was added for testing and debugging purposes.

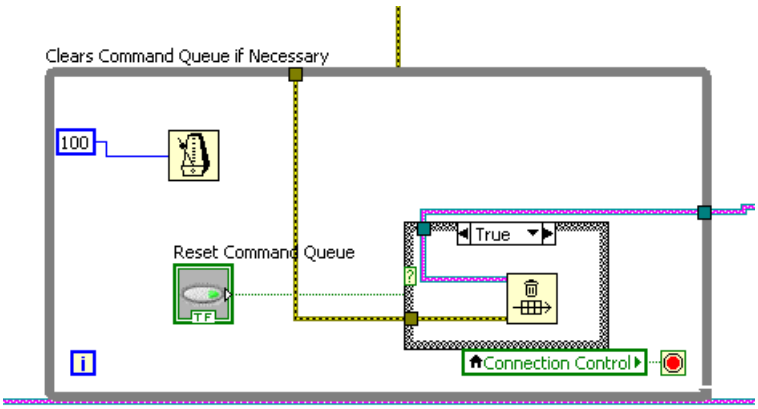


Figure A-4: Command Queue Reset Loop

Command Queue Send Loop

This loop sends commands placed into the queue to the microcontroller. Works in conjunction with the TCP send loop.

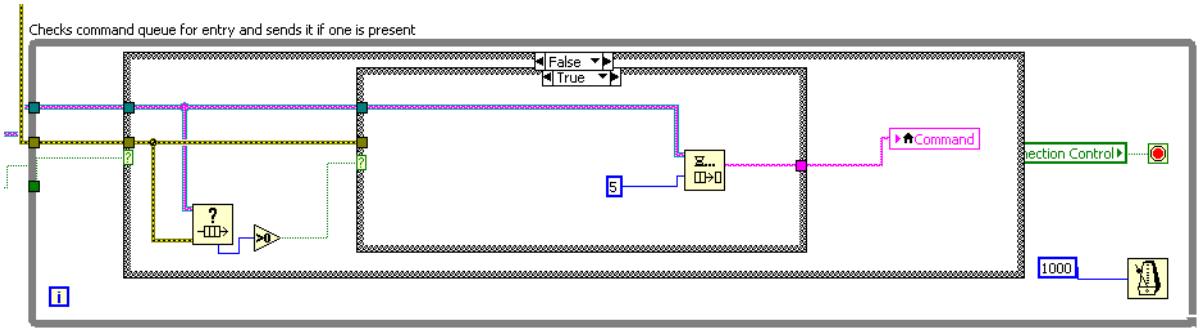


Figure A-5: Command Queue Send Loop

TCP Send Loop

Sends the contents of the command field to the server if the field is not blank. The command field only is full when an item in the command queue is placed in the field. This loop handles all signals going to the microcontroller. To send any command, the string simply needs to be placed in the command field. This gives developers the ability to easily manipulate the ways that input can be accepted and then transferred to this field.

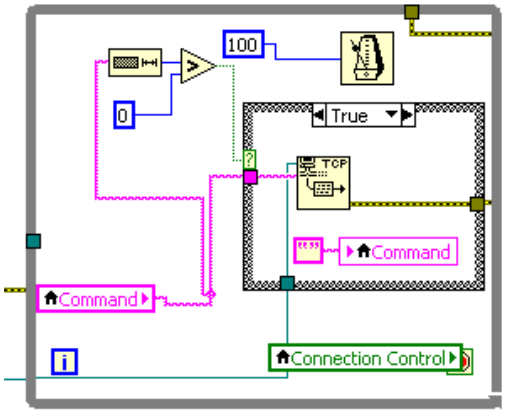


Figure A-6: TCP Send Loop

Blower Control Loop

Enables User to turn the main blower on and off. Keeps track of current blower state. Bypasses actuator commands. Only sends command if it is changing the current blower state.

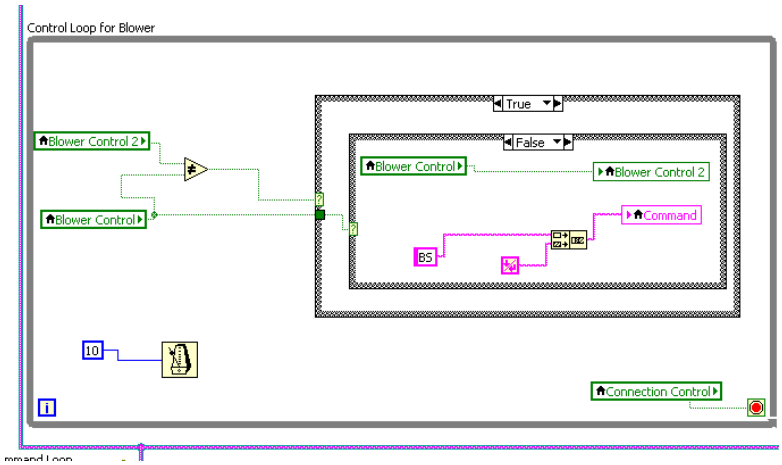


Figure A-7: Blower Control Loop

Angle Command Loop

This loop allows the user to move the actuator based on the given angle. Converts the angle into a position based on given condition data. The position data is converted to a millivolt value and sent to the queue to be sent to the microcontroller.

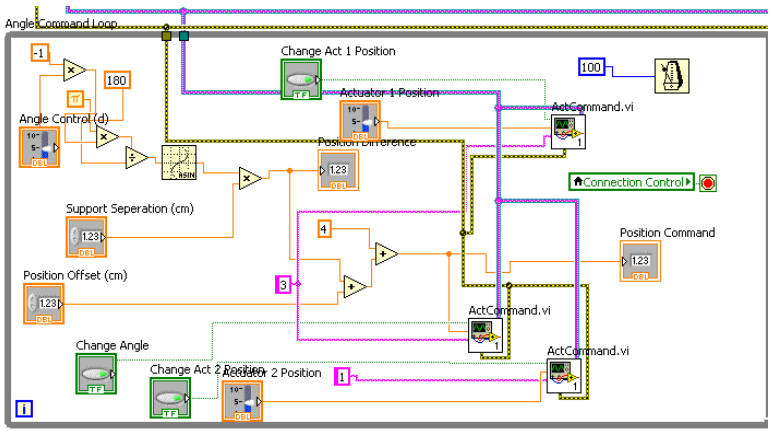


Figure A-8: Angle Command Loop with extra Actuator code

Wind Speed Auto Control Loop

This loop compares the wind speed data to the requested wind speed. Depending upon what the wind speed is and what the desired speed is, the damper is opened or closed. The wind speed is controlled in this fashion or by the user directly in the Manual Damper Control Loop. There is a switch that chooses between these two control loops. They do not run concurrently.

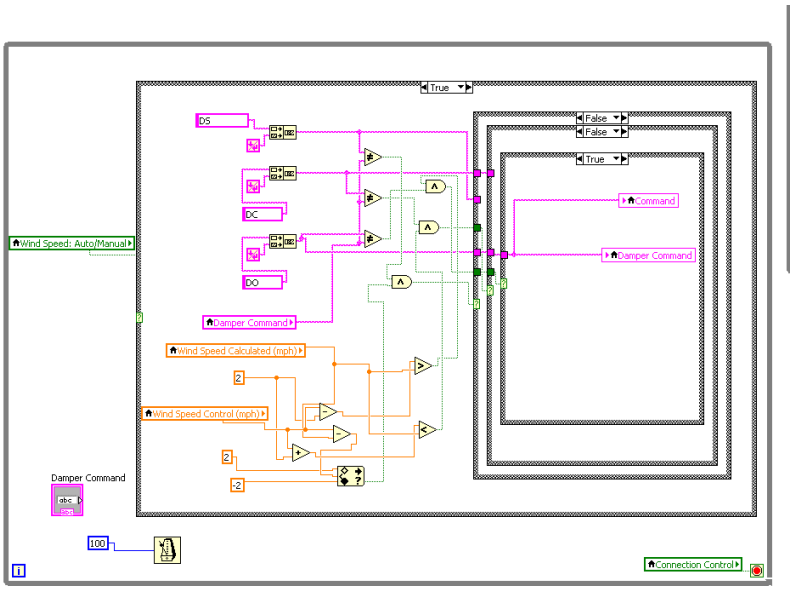


Figure A-9: Wind Speed Auto Control Loop

Wind Speed Calculation Loop

This loop uses the equations in Figure 6 to determine the wind speed based on the differential pressure. The pressure sensor data is sent to the client as a voltage and the wind speed is then computed using the pressure in the tunnel, the ambient pressure, and the temperature as given by the user.

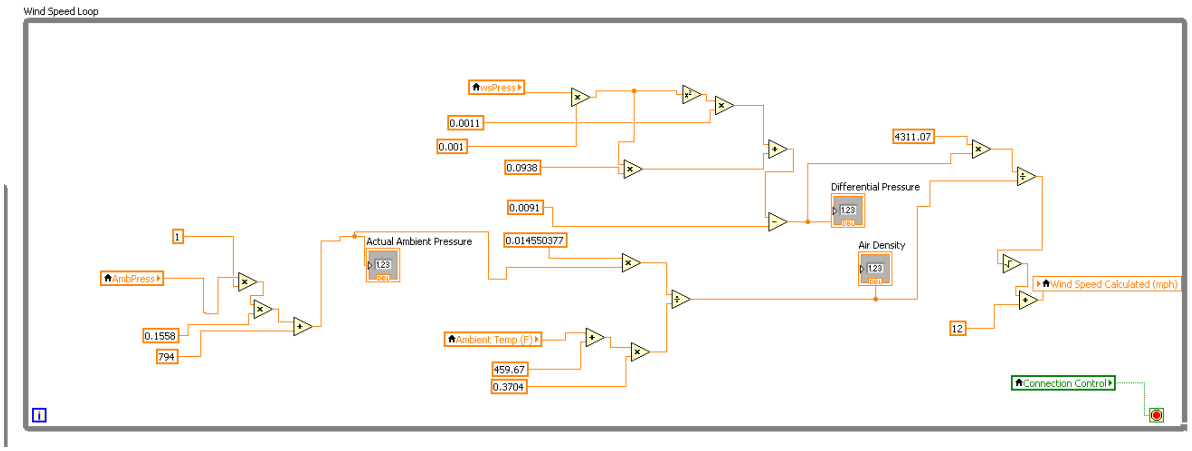


Figure A-10: Wind Speed Calculation Loop

Data Recording Loop

At a time interval chosen by the user, this loop controls the collection and writing of data to a file. The data is exported as simple text tab-delimited. This allows it to be opened in a program like excel with the formatting intact without extensive formatting to make it the normal excel format. Each data set is time stamped to aid in comparing information. The recording can be started and stopped anytime while the client is connected. It can be started at a specific collection interval, stopped, and started at a different interval.

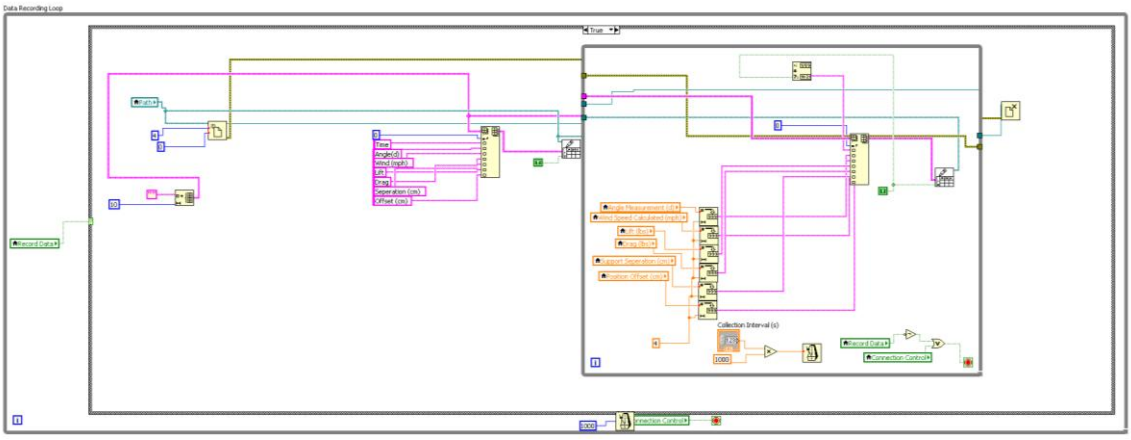


Figure A-11: Data Recording Loop

Angle Display Loop

To improve accuracy and as a check to the integrity of the angle command function, this loop displays the current angle given by the position of the actuator. It calculates the angle by going through the opposite process of the Actuator Control Loop.

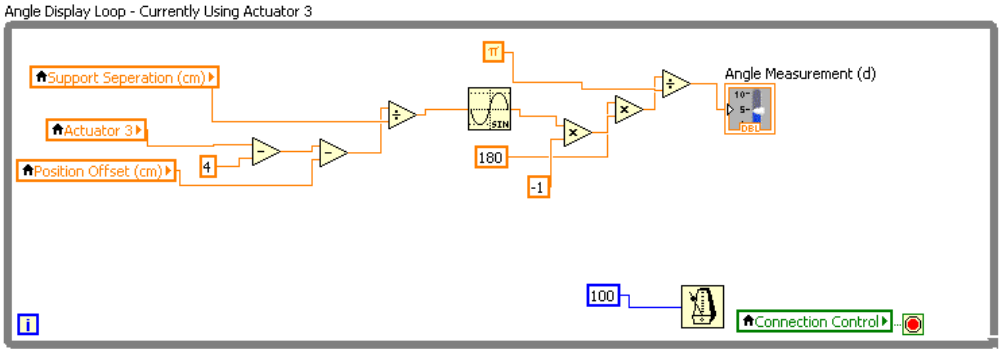


Figure A-12: Angle Display Loop

Actuator Conversion Loop

While title Actuator Conversion Loop, this loop converts the data from the microcontroller into values that are more easily interpreted by a user. This also makes calculations such as the angle calculation easier.

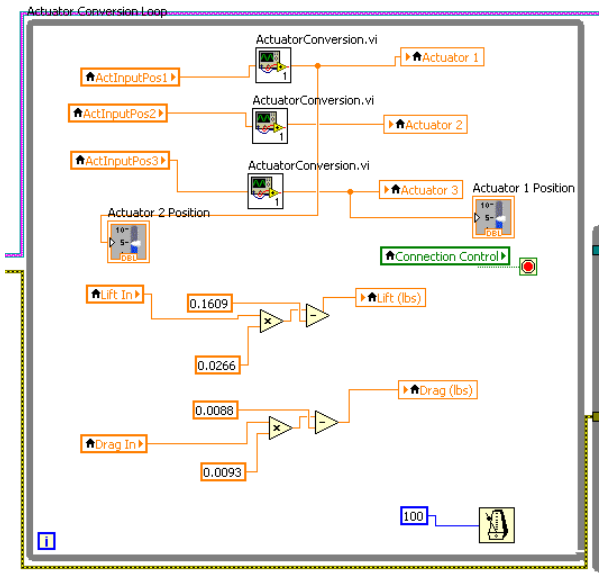


Figure A-13: Actuator Conversion Loop

Actuator Initialization Loop

When starting the client, it automatically sets the actuator to the midpoint of its range of extension. This provides a default starting point. If the user so desires, they can offset this value if the midpoint does not place the target object at a zero angle position. The user can also choose to move the actuator to this position at anytime by hitting the “Initialize Actuator” Button.

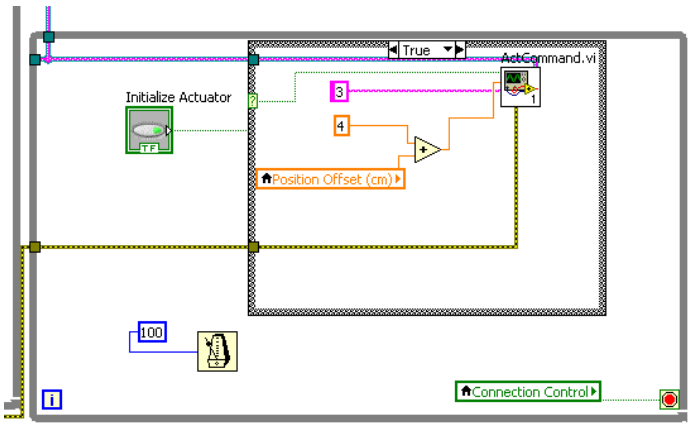


Figure A-14: Actuator Initialization Loop

Wind Speed Manual Control Loop

This loop is the other facet of the wind speed control. It is on when the auto control loop is off. It allows the user to increase or decrease the wind speed by opening or closing the damper. Hitting the increase speed button opens the damper until the button is hit again. The same process is used to close the damper. Switching between auto and manual control automatically causes the damper to be stopped if it is in motion.

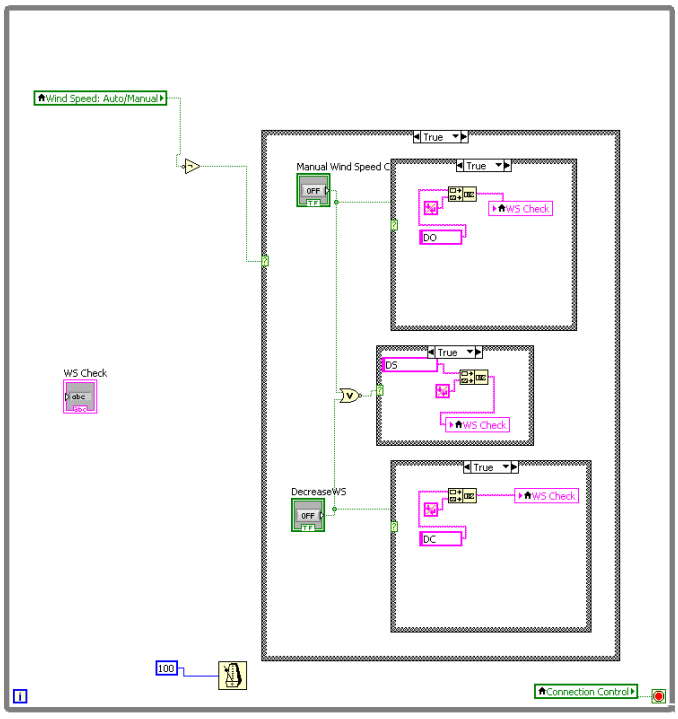


Figure A-15: Wind Speed Manual Control Block

Wind Speed Command Loop

This loop checks to make sure that the wind speed controller loops aren't sending a command repetitively to the microcontroller. This prevents the microcontroller from being overwhelmed by redundant command signals.

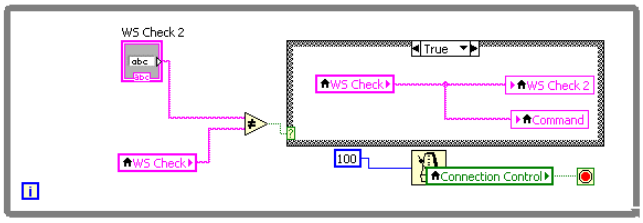
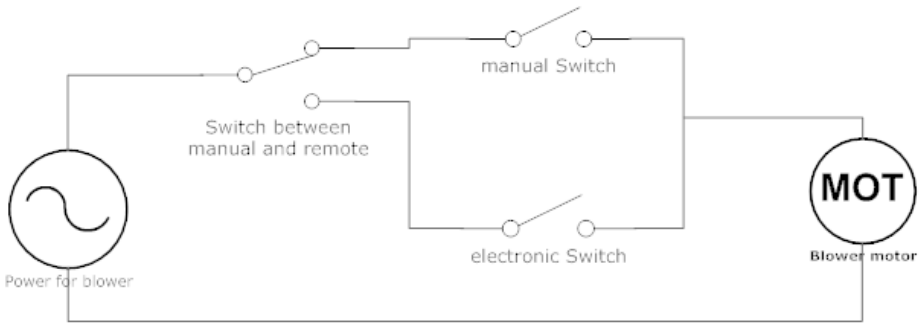


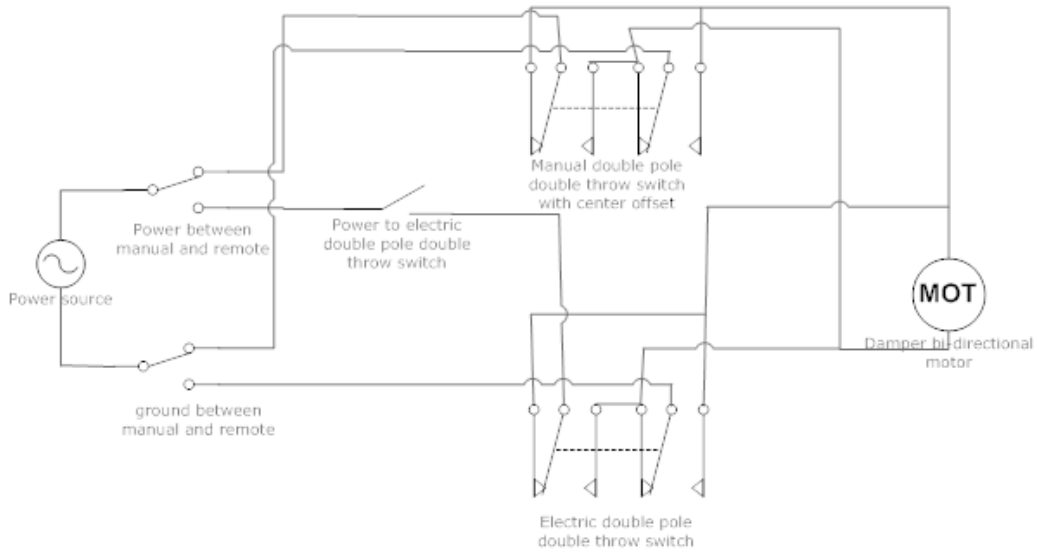
Figure A-16: Wind Speed Command Loop

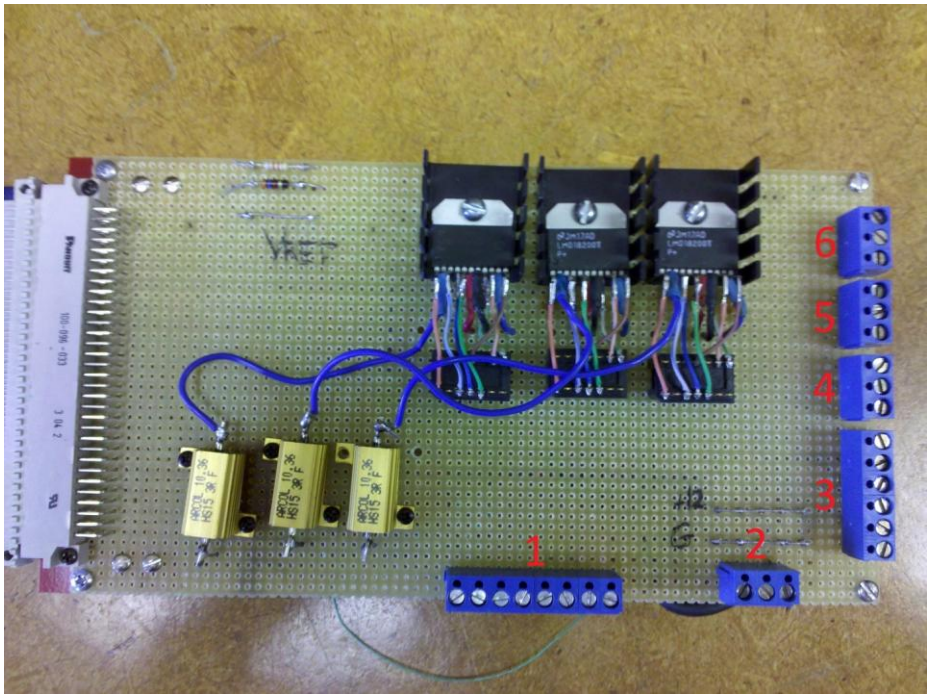
Appendix B: Wiring Diagrams

Blower Motor Wiring Diagram

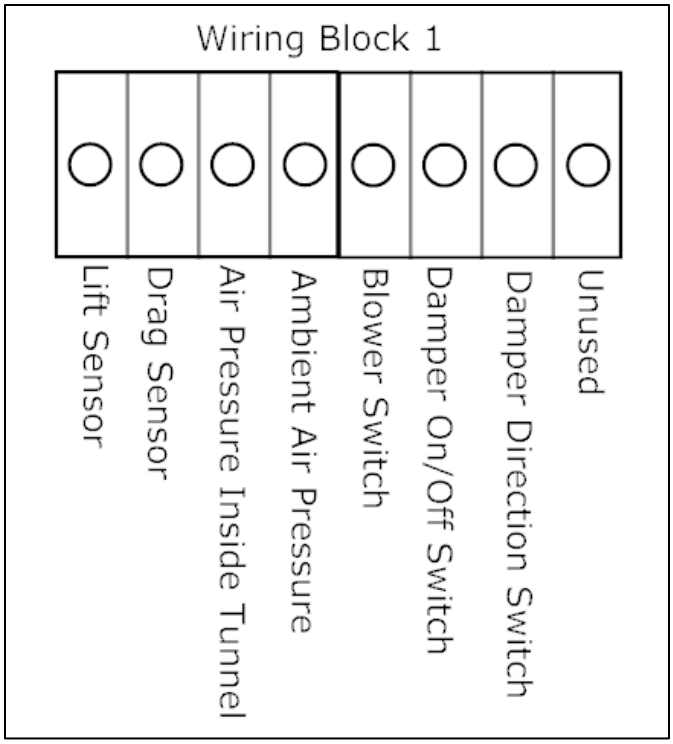


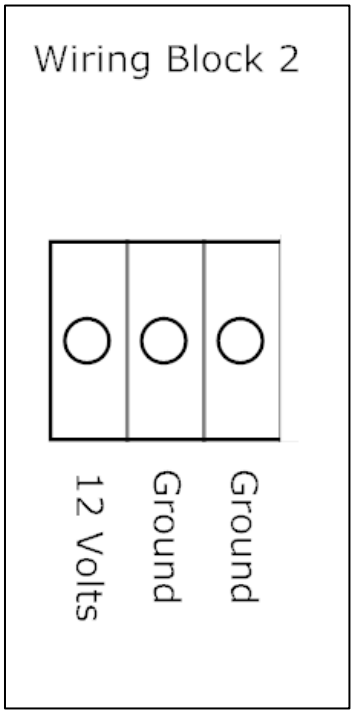
Damper Motor Wiring Diagram

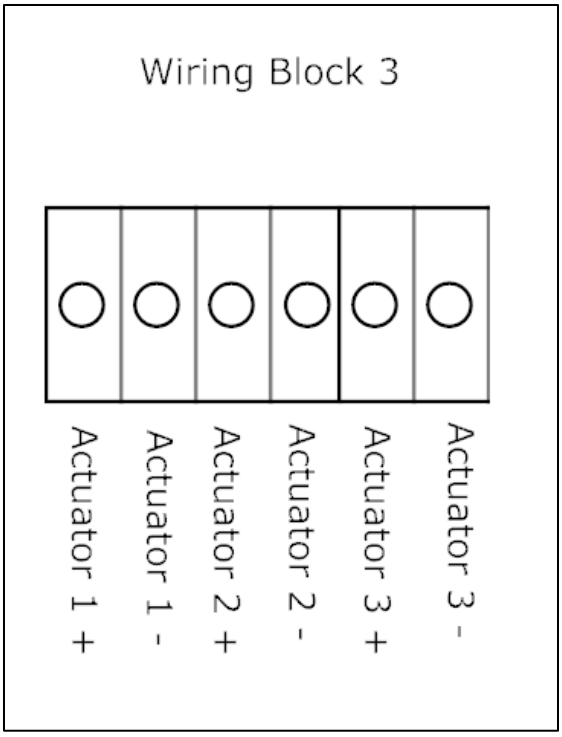


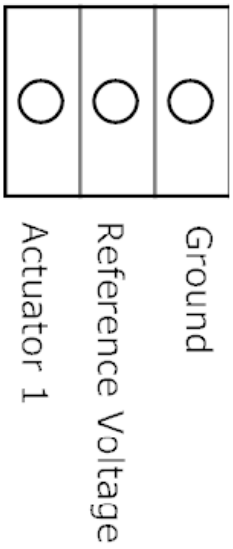


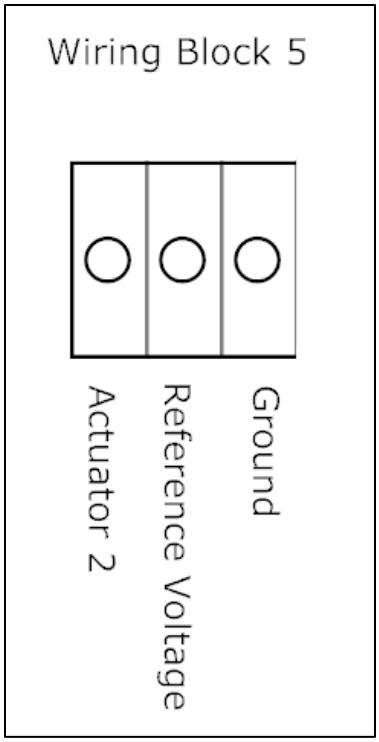
The red numbers indicate which wiring block is being described in the diagrams below.

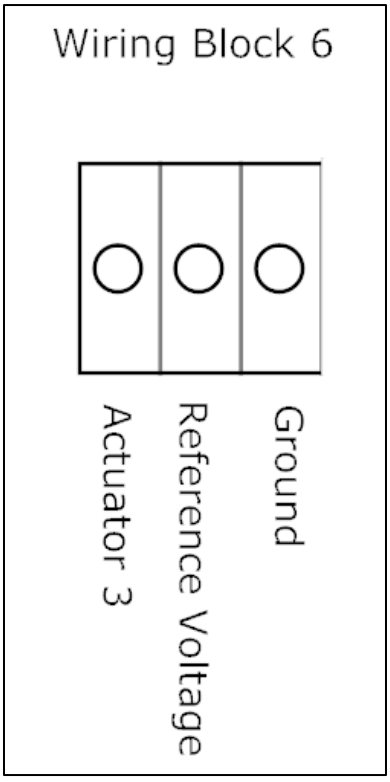

Wiring Block 1
Unused
Damper Direction Switch
Damper On/Off Switch
Blower Switch
Ambient Air Pressure
Air Pressure Inside Tunnel
Drag Sensor
Lift Sensor


Wiring Block 2
Ground
Ground
12 Volts


Wiring Block 3
Actuator 3 -
Actuator 3 +
Actuator 2 -
Actuator 2 +
Actuator 1 -
Actuator 1 +


Wiring Block 4
Ground
Reference Voltage
Actuator 1


Wiring Block 5
Ground
Reference Voltage
Actuator 2


Wiring Block 6
Ground
Reference Voltage
Actuator 3

Appendix C: 8051F120 Firmware Source Code

Adc0int8rr.c

```
#include "adc0int8rr.h"

#include "C8051F120.h"                  // SFR declarations

//-----------------------------------------------------------------------------

// Global VARIABLES

//-----------------------------------------------------------------------------

__xdata long int result[9];          // ADC0 decimated value - visible outside by means

                                  // of defining it as external in the library header

sbit newData;//used by ISR to show new data is ready

unsigned char chConv;//used by ISR to show main loop which was converted

//-----------------------------------------------------------------------------------------

// ADC0_Timer3_Init

//-----------------------------------------------------------------------------------------

//

// Configure Timer3 to auto-reload and generate an interrupt at interval

// specified by <counts> using SYSCLK/12 as its time base.

//

// Configure ADC0 to use Timer3 overflows as conversion source, to

// generate an interrupt on conversion complete, and to use left-justified

// output mode.  Enables ADC end of conversion interrupt. Leaves interrupts disabled.

//

void ADC0_Timer3_Init (unsigned long sysclock)

{

    char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page

    SFRPAGE = TMR3_PAGE;                // set SFR page

    TMR3CN  = 0x00;                     // Stop Timer3; Clear TF3;

  TMR3CF  = 0x00;                     // use SYSCLK/12 as timebase

   // RCAP3   = -counts;                  // Init reload values - Modified by Nick DeTrempe

    RCAP3 = -0x27E0;
        //manually determined for 600 Hz interrupt (100Hz per channel)

    TMR3    = RCAP3;                    // set to reload immediately

    EIE2   &= ~0x01;                    // DISABLE Timer3 interrupts
```

```
    TMR3CN |= 0x04;                     // start Timer3


    SFRPAGE = ADC0_PAGE;

    // ADC0 Control

    ADC0CN = 0x05;                      // ADC0 disabled; normal tracking mode; data is left-
justified

                                        // 0x04 ADC0 conversions are initiated on overflow of
Timer3

                                        // ----00-- ADC0 conversions started manually by setting
AD0BUSY=1

                                        // ----01-- ADC0 conversions are initiated on overflow of
Timer3

                                        // ----11-- ADC0 conversions are initiated on overflow of
Timer2

                                        // 0x01 ADC0 data is left-justified

                                        // 0x40 ADC0 in low power tracking mode - adds 1.5us wait
before conversion starts

                                        // 0x40 is not needed in this case because we will change
the multiplexer source

                                        //      when a conversion is done and let the conversion
begin at the next Timer3

                                        //      rolls over - that should meet this wait time
requirement

                                        //      as long as the sampling frequency is way below
100KHz


    // Conversion speed and input gain

    ADC0CF = (sysclock / 2500000) << 3; // ddddd--- ADC0 conversion clock at 2.5 MHz

    ADC0CF |= 0x00;                     // -----ddd ADC) internal gain (PGA): 0-1, 1-2, 2-4, 3-8,
4-16, 6-0.5


    // Voltage Source and mode

    REF0CN = 0x07;                      // enable: 0x01 on-chip VREF, 0x02 VREF output buffer for
ADC and DAC, and 0x04 temp sensor

    //AMX0CF = 0x00;                      // 0x00 Select 8 independent inputs, 0x0F select 4
differential pairs

    AMX0CF = 0x01;
       //6 independent pairs, channels 6 and 7 are one differential pair.

    AMX0SL = 0x01;                      // Select Channel 0 as ADC mux output,

                                        // When AMX0CF = 0x00 then AMX0SL = channel number, 8 for
TEMP sensor
```

```
    // EIE2 |= 0x01;                    // enable Timer3 interrupts - not doing that this time,
instead/in addition to we must:

    EIE2 |= 0x02;                       // enable ADC interrupts when the conversion is complete

    AD0EN = 1;                          // enable ADC

    // Remember to enable global interrupts when ready


    SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page

}




//-----------------------------------------------------------------------------

// Interrupt Service Routines

//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------

// ADC0_ISR

//-----------------------------------------------------------------------------

//

// ADC0 end-of-conversion ISR

// We post the result in the global variable <result> and round robin switch to the next channel.

//

void ADC0_ISR (void) __interrupt 15

{

    int ch;

    SFRPAGE = ADC0_PAGE;

    ch = AMX0SL;                        // retrieve which channel has just been measured

    result[ch] = ADC0;                  // read ADC value from the previously selected channel

    chConv = ch;

    ch++; if (ch>7) ch=0;

    AMX0SL = ch;                        // pick the next channel to read - round robin 0, 1, ...
8, 0, ...

    newData = 1;

    AD0INT = 0;                         // clear ADC conversion complete indicator - must be done
manualy

}
```

```
//-----------------------------------------------------------------------------
// setGain
//-----------------------------------------------------------------------------
//
// Change gain for all channels of ADC0
//
void setGain(unsigned char gain)
{
/*
    -----000 => *1
    -----001 => *2
    -----010 => *4
    -----011 => *8
    -----10x => *16
    -----11x => /2
*/
    char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page
    unsigned char saveEA = EA;          // save interrupt enabled status

    unsigned char encoded=0;
    while(gain>1)
    {
        encoded++;
        gain=gain>>1;                   // gain/=2;
    }

    EA = 0;                             // disable interrupts
    ADC0CF = (ADC0CF & 0xF8) | ( encoded & 0x07); // set the gain
    EA = saveEA;                        // re-enable interrupts if were enabled
    SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}
```

```
//-----------------------------------------------------------------------------
// getRecentResult
//-----------------------------------------------------------------------------
//
// This function returns the most recent conversion results
// Doing this in main function instead would save a few bytes of code
// However, putting it in a separate function results in code that is easier to follow
// and thus better both for firmware life cycle and educational purposes
//
unsigned int getRecentResult(unsigned char channel)
{
    char saveEA = EA;                   // save interrupt enabled status
    unsigned int returnvalue;
    EA = 0;                             // disable interrupts
    returnvalue = result[channel];
    EA = saveEA;                        // re-enable interrupts if were enabled
    return(returnvalue);
}
```

am_com.c

```
#include "am_com.h"
#include "C8051F120.h"              // Device-specific SFR Definitions
#include <stdio.h>                  // Add support for printf, putchar, getchar, etc.

#ifdef SDCC
char * gets_safe(char *s, short n);
#endif

//-----------------------------------------------------------------------------
// UART_Init
//-----------------------------------------------------------------------------
//
// Configure the UART1 using Timer1, for <baudrate> and 8-N-1.
//
```

```
void UART_Init (unsigned long sysclk, unsigned long baudrate)
{
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page
    const unsigned long int sysclkoverbaud = sysclk/baudrate;

    SFRPAGE = UART_PAGE;
    SCON1   = 0x10;                 // SCON1: mode 0, 8-bit UART, enable RX

    SFRPAGE = TIMER01_PAGE;
    TMOD   &= ~0xF0;
    TMOD   |=  0x20;                // TMOD: timer 1, mode 2, 8-bit reload


    // Set Timer 1 timebase.
    // Note: Since Timer 0 is used by the TCP/IP Library and forces the
    // shared T0/T1 prescaler to sysclk/48, Timer 1 may only be clocked
    // from sysclk or sysclk/48

    // If reload value is less than 8-bits, select sysclk
    // as Timer 1 baud rate generator
    if (sysclkoverbaud>>9 < 1)
    {
        TH1 = -(sysclkoverbaud>>1);
        CKCON |= 0x10;              // T1M = 1; SCA1:0 = xx

        // Otherwise, select sysclk/48 prescaler.
    }
    else
    {
        // Adjust for truncation in special case
        // Note: Additional cases may be required if the system clock is changed.
        if ((baudrate == 115200) && (sysclk == 98000000))
```

```
            TH1 = -((sysclkoverbaud/2/48)+1);
        else
            TH1 = -(sysclkoverbaud/2/48);

        CKCON &= ~0x13;             // Clear all T1 related bits
        CKCON |=  0x02;             // T1M = 0; SCA1:0 = 10
    }

    TL1 = TH1;                      // initialize Timer1
    TR1 = 1;                        // start Timer1

    SFRPAGE = UART_PAGE;
    TI1 = 1;                        // Indicate TX1 ready

    SFRPAGE = SFRPAGE_SAVE;         // Restore SFR page
}



void UART_puts(const char* buffer)
{
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR Page
    SFRPAGE = UART_PAGE;
    while (*buffer)
    {
        putchar(*buffer++);
    }
    SFRPAGE = SFRPAGE_SAVE;         // Restore SFR page
}

void UART_gets(char* buffer, int len)
{
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR Page
```

```
    SFRPAGE = UART_PAGE;
#ifdef SDCC
    gets_safe(buffer,len);          // !! SDCC originally has no max buffer parameter !!
#else
    gets(buffer,len);
#endif
    SFRPAGE = SFRPAGE_SAVE;         // Restore SFR page
}

void UART_Quit (void)
{
    char SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page

    // Disable Timer1
    SFRPAGE = TIMER01_PAGE;
    TR1 = 0;                         // Stop Timer1
    TMOD = 0x00;                     // Restore the TMOD register to its reset value
    CKCON = 0x00;                    // Restore the CKCON register to its reset value

    // Disable UART1
    SFRPAGE = UART_PAGE;
    SCON1 = 0x00;                    // Disable UART1

    SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
}

#ifdef SDCC
char * gets_safe(char *s, short n) {
    char c;
    unsigned short count=0;
    n--;
```

```
    while (1) {
        c=getchar();
        switch(c) {
        case '\b': // backspace
            if (count) {
             //   putchar('\b');
             //   putchar(' ');
             //   putchar('\b');
                s--;
                count--;
            }
            break;
        case '\n':
        case '\r': // CR or LF
          //  putchar('\r');
           // putchar('\n');
             *s=0;
             return s;
        default:
            if (count<n) {
                *s++=c;
                count++;
              //  putchar(c);
            } else {
              //  putchar('\a');
            }
            break;
        }
    }
}

char getchar ()  {
```

```
  char c;
  while (!RI1);
  c = SBUF1;
  RI1 = 0;
  return (c);
}

#define XON  0x11
#define XOFF 0x13

/*
 * putchar (full version):  expands '\n' into CR LF and handles
 *                          XON/XOFF (Ctrl+S/Ctrl+Q) protocol
 */
void putchar (char c)  {
  if (c == '\n')  {
    if (RI1)  {
      if (SBUF1 == XOFF)  {
        do  {
          RI1 = 0;
          while (!RI1);
        }
        while (SBUF1 != XON);
        RI1 = 0;
      }
    }
    while (!TI1);
    TI1 = 0;
    SBUF1 = 0x0d;                    // output CR
  }
  if (RI1)  {
    if (SBUF1 == XOFF)  {
```

```
      do  {
        RI1 = 0;
        while (!RI1);
      }
      while (SBUF1 != XON);
      RI1 = 0;
    }
  }
  while (!TI1);
  TI1 = 0;
  SBUF1 = c;
}

#endif
```

am_com.h

```
#ifndef _AM_COM
#define _AM_COM
void UART_Init (unsigned long sysclk, unsigned long baudrate);
void UART_puts (const char* buffer);
void UART_gets (char* buffer, int len);
void UART_Quit (void);
#endif
```

am_init.c

```
#include "am_init.h"

#include "C8051F120.h"                          // Device-specific SFR Definitions

//-----------------------------------------------------------------------------
// Initialization Routines
//-----------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the SPI, Interrupts, Crossbar and GPIO ports
//
void PORT_Init(void)
{
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page

    SFRPAGE = CONFIG_PAGE;          // set SFR page
    P0MDOUT |= 0x01;                // Set TX1 pin to push-pull
    P1MDOUT |= 0x40;                // Set P1.6(TB_LED) to push-pull
//  P2MDOUT |= 0x00;
//  P3MDOUT |= 0x00;

    // all pins used by the external memory interface are in push-pull mode
    // including /WR (P4.7) and /RD (P4.6) but the reset (P4.5) is open-drain
    // P4MDOUT =  0xC0;
    // P4 = 0xC0;                   // /WR, /RD, are high, RESET is low

    // You may want to set P4.3 (AB4_LED1) and P4.4 (AB4_LED2) to push-pull
    // You may want to set P4.1 (AB4_SW1)  and P4.2 (AM4_SW2)  to open-drain
    P4MDOUT =  0xD8;
    // You may want to prevent permament on on AB4 switches by not pulling them down
    P4 = 0xC6;                      // /WR, /RD, SW1, SW2 are high, RESET is low,

    P5MDOUT =  0xFF;                // P5, P6 contain the address lines
    P6MDOUT =  0xFF;                // P5, P6 contain the address lines
    P7MDOUT =  0xFF;                // P7 contains the data lines
    P5 = 0xFF;                      // P5, P6 contain the address lines
```

```
    P6 = 0xFF;                      // P5, P6 contain the address lines
    P7 = 0xFF;                      // P7 contains the data lines

    TCON &= ~0x01;                  // Make /INT0 level triggered

    XBR0 = 0x80;                    // Enable CP0, Close PCA0 I/O, Close UART0
    XBR1 = 0x04;                    // Enable INT0 input pin, this puts /INT0 on P0.3.
    XBR2 = 0x44;                    // Enable crossbar and weak pull-up, Enable UART1

    SFRPAGE = SFRPAGE_SAVE;         // Restore SFR page
}

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock.
//
void SYSCLK_Init(void)
{
    int i;                          // software timer

    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page

    SFRPAGE = CONFIG_PAGE;           // set SFR page

    OSCICN = 0x83;                  // set internal oscillator to run
                                    // at its maximum frequency

    CLKSEL = 0x00;                  // Select the internal osc. as
                                    // the SYSCLK source
    //Turn on the PLL and increase the system clock by a factor of M/N
```

```
    PLL0CN  = 0x00;                 // Set internal osc. as PLL source
    SFRPAGE = LEGACY_PAGE;
    FLSCL   = 0x30;                 // Set FLASH read time for 100 MHz clk
    SFRPAGE = CONFIG_PAGE;

    PLL0CN |= 0x01;                 // Enable Power to PLL

    PLL0DIV = 0x01;                 // Set Pre-divide value to N (N = 1)
    PLL0MUL = 0x03;                 // Multiply SYSCLK by M (M=3)
    PLL0FLT = 0x01;                 // Set the PLL filter register for
                                    // a reference clock from 12.2 - 19.5 MHz
                                    // and an output clock from 65 - 100 MHz
    for (i=0; i < 256; i++) ;       // Wait at least 5us
    PLL0CN  |= 0x02;                // Enable the PLL
    while(!(PLL0CN & 0x10));        // Wait until PLL frequency is locked
    CLKSEL  = 0x02;                 // Select PLL as SYSCLK source

    SFRPAGE = SFRPAGE_SAVE;         // Restore SFR page
}
```

am_init.h

```
#ifndef _AM_INIT
#define _AM_INIT
#define INTCLK       24500000L      // Internal oscillator frequency in Hz
#define SYSCLK       73500000L      // System clock in Hz
// Initialization Routines
void PORT_Init(void);
void SYSCLK_Init(void);
#endif
```

am_wait.c

```
#include "am_wait.h"
```

```
#include "C8051F120.h"                          // Device-specific SFR Definitions

//-----------------------------------------------------------------------------
// Global Constants
//-----------------------------------------------------------------------------
#define SYSCLK         73500000     // System Clock Frequency in Hz

//-----------------------------------------------------------------------------
// wait_ms
//-----------------------------------------------------------------------------
//
// This routine inserts a delay of <ms> milliseconds.
//
void wait_ms(int ms)
{
    char SFRPAGE_SAVE = SFRPAGE;    // Save Current SFR page

    SFRPAGE = TMR2_PAGE;

    TMR2CN = 0x00;                  // Stop Timer2; Clear TF2;
    TMR2CF = 0x00;                  // use SYSCLK/12 as timebase

    RCAP2 = -(SYSCLK/1000/12);      // Timer 2 overflows at 1 kHz
    TMR2 = RCAP2;

    ET2 = 0;                        // Disable Timer 2 interrupts

    TR2 = 1;                        // Start Timer 2

    while(ms){
        TF2 = 0;
        while(!TF2);                // wait until timer overflows
```

```
        ms--;                       // decrement ms
    }

    TR2 = 0;                        // Stop Timer 2

    SFRPAGE = SFRPAGE_SAVE;         // Restore SFRPAGE
}
```

am_wait.h

```
#ifndef _AM_WAIT
#define _AM_WAIT
void wait_ms(int ms);
#endif
```

c8051F120.h

```
/*-----------------------------------------------------------------------
   Register Declarations for the Cygnal/SiLabs C8051F12x-F13x Processor Range

   Copyright (C) 2003 - Maarten Brock, sourceforge.brock@dse.nl

   This library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   This library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with this library; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
```

```
//-----------------------------------------------------------------------------*/

#ifndef C8051F120_H
#define C8051F120_H


/*  BYTE Registers  */

/*  All Pages */
__sfr __at (0x80) P0             ;  /* PORT 0                                */
__sfr __at (0x81) SP             ;  /* STACK POINTER                         */
__sfr __at (0x82) DPL            ;  /* DATA POINTER - LOW BYTE               */
__sfr __at (0x83) DPH            ;  /* DATA POINTER - HIGH BYTE              */
__sfr __at (0x84) SFRPAGE        ;  /* SFR PAGE SELECT                       */
__sfr __at (0x85) SFRNEXT        ;  /* SFR STACK NEXT PAGE                   */
__sfr __at (0x86) SFRLAST        ;  /* SFR STACK LAST PAGE                   */
__sfr __at (0x87) PCON           ;  /* POWER CONTROL                         */
__sfr __at (0x90) P1             ;  /* PORT 1                                */
__sfr __at (0xA0) P2             ;  /* PORT 2                                */
__sfr __at (0xA8) IE             ;  /* INTERRUPT ENABLE                      */
__sfr __at (0xB0) P3             ;  /* PORT 3                                */
__sfr __at (0xB1) PSBANK         ;  /* FLASH BANK SELECT                     */
__sfr __at (0xB8) IP             ;  /* INTERRUPT PRIORITY                    */
__sfr __at (0xD0) PSW            ;  /* PROGRAM STATUS WORD                   */
__sfr __at (0xE0) ACC            ;  /* ACCUMULATOR                           */
__sfr __at (0xE6) EIE1           ;  /* EXTERNAL INTERRUPT ENABLE 1           */
__sfr __at (0xE7) EIE2           ;  /* EXTERNAL INTERRUPT ENABLE 2           */
__sfr __at (0xF0) B              ;  /* B REGISTER                            */
__sfr __at (0xF6) EIP1           ;  /* EXTERNAL INTERRUPT PRIORITY REGISTER 1*/
__sfr __at (0xF7) EIP2           ;  /* EXTERNAL INTERRUPT PRIORITY REGISTER 2*/
__sfr __at (0xFF) WDTCN          ;  /* WATCHDOG TIMER CONTROL                */
```

```
/*  Page 0x00 */

__sfr __at (0x88) TCON            ;  /* TIMER CONTROL                          */

__sfr __at (0x89) TMOD            ;  /* TIMER MODE                             */

__sfr __at (0x8A) TL0             ;  /* TIMER 0 - LOW BYTE                     */

__sfr __at (0x8B) TL1             ;  /* TIMER 1 - LOW BYTE                     */

__sfr __at (0x8C) TH0             ;  /* TIMER 0 - HIGH BYTE                    */

__sfr __at (0x8D) TH1             ;  /* TIMER 1 - HIGH BYTE                    */

__sfr __at (0x8E) CKCON           ;  /* TIMER 0/1 CLOCK CONTROL                */

__sfr __at (0x8F) PSCTL           ;  /* FLASH WRITE/ERASE CONTROL              */

__sfr __at (0x91) SSTA0           ;  /* UART 0 STATUS                          */

__sfr __at (0x98) SCON0           ;  /* UART 0 CONTROL                         */

__sfr __at (0x98) SCON            ;  /* UART 0 CONTROL                         */

__sfr __at (0x99) SBUF0           ;  /* UART 0 BUFFER                          */

__sfr __at (0x99) SBUF            ;  /* UART 0 BUFFER                          */

__sfr __at (0x9A) SPI0CFG         ;  /* SPI 0 CONFIGURATION                    */

__sfr __at (0x9B) SPI0DAT         ;  /* SPI 0 DATA                             */

__sfr __at (0x9D) SPI0CKR         ;  /* SPI 0 CLOCK RATE CONTROL               */

__sfr __at (0xA1) EMI0TC          ;  /* EMIF TIMING CONTROL                    */

__sfr __at (0xA2) EMI0CN          ;  /* EMIF CONTROL                           */

__sfr __at (0xA2) _XPAGE          ;  /* XDATA/PDATA PAGE                       */

__sfr __at (0xA3) EMI0CF          ;  /* EMIF CONFIGURATION                     */

__sfr __at (0xA9) SADDR0          ;  /* UART 0 SLAVE ADDRESS                   */

__sfr __at (0xB7) FLSCL           ;  /* FLASH SCALE                            */

__sfr __at (0xB9) SADEN0          ;  /* UART 0 SLAVE ADDRESS MASK              */

__sfr __at (0xBA) AMX0CF          ;  /* ADC 0 MUX CONFIGURATION                */

__sfr __at (0xBB) AMX0SL          ;  /* ADC 0 MUX CHANNEL SELECTION            */

__sfr __at (0xBC) ADC0CF          ;  /* ADC 0 CONFIGURATION                    */

__sfr __at (0xBE) ADC0L           ;  /* ADC 0 DATA - LOW BYTE                  */

__sfr __at (0xBF) ADC0H           ;  /* ADC 0 DATA - HIGH BYTE                 */

__sfr __at (0xC0) SMB0CN          ;  /* SMBUS 0 CONTROL                        */

__sfr __at (0xC1) SMB0STA         ;  /* SMBUS 0 STATUS                         */

__sfr __at (0xC2) SMB0DAT         ;  /* SMBUS 0 DATA                           */
```

```
__sfr __at (0xC3) SMB0ADR         ;  /* SMBUS 0 SLAVE ADDRESS                        */
__sfr __at (0xC4) ADC0GTL         ;  /* ADC 0 GREATER-THAN REGISTER - LOW BYTE       */
__sfr __at (0xC5) ADC0GTH         ;  /* ADC 0 GREATER-THAN REGISTER - HIGH BYTE      */
__sfr __at (0xC6) ADC0LTL         ;  /* ADC 0 LESS-THAN REGISTER - LOW BYTE          */
__sfr __at (0xC7) ADC0LTH         ;  /* ADC 0 LESS-THAN REGISTER - HIGH BYTE         */
__sfr __at (0xC8) TMR2CN          ;  /* TIMER 2 CONTROL                              */
__sfr __at (0xC9) TMR2CF          ;  /* TIMER 2 CONFIGURATION                        */
__sfr __at (0xCA) RCAP2L          ;  /* TIMER 2 CAPTURE REGISTER - LOW BYTE          */
__sfr __at (0xCB) RCAP2H          ;  /* TIMER 2 CAPTURE REGISTER - HIGH BYTE         */
__sfr __at (0xCC) TMR2L           ;  /* TIMER 2 - LOW BYTE                           */
__sfr __at (0xCC) TL2             ;  /* TIMER 2 - LOW BYTE                           */
__sfr __at (0xCD) TMR2H           ;  /* TIMER 2 - HIGH BYTE                          */
__sfr __at (0xCD) TH2             ;  /* TIMER 2 - HIGH BYTE                          */
__sfr __at (0xCF) SMB0CR          ;  /* SMBUS 0 CLOCK RATE                           */
__sfr __at (0xD1) REF0CN          ;  /* VOLTAGE REFERENCE 0 CONTROL                  */
__sfr __at (0xD2) DAC0L           ;  /* DAC 0 REGISTER - LOW BYTE                    */
__sfr __at (0xD3) DAC0H           ;  /* DAC 0 REGISTER - HIGH BYTE                   */
__sfr __at (0xD4) DAC0CN          ;  /* DAC 0 CONTROL                                */
__sfr __at (0xD8) PCA0CN          ;  /* PCA 0 COUNTER CONTROL                        */
__sfr __at (0xD9) PCA0MD          ;  /* PCA 0 COUNTER MODE                           */
__sfr __at (0xDA) PCA0CPM0        ;  /* PCA 0 MODULE 0 CONTROL                       */
__sfr __at (0xDB) PCA0CPM1        ;  /* PCA 0 MODULE 1 CONTROL                       */
__sfr __at (0xDC) PCA0CPM2        ;  /* PCA 0 MODULE 2 CONTROL                       */
__sfr __at (0xDD) PCA0CPM3        ;  /* PCA 0 MODULE 3 CONTROL                       */
__sfr __at (0xDE) PCA0CPM4        ;  /* PCA 0 MODULE 4 CONTROL                       */
__sfr __at (0xDF) PCA0CPM5        ;  /* PCA 0 MODULE 5 CONTROL                       */
__sfr __at (0xE1) PCA0CPL5        ;  /* PCA 0 MODULE 5 CAPTURE/COMPARE - LOW BYTE    */
__sfr __at (0xE2) PCA0CPH5        ;  /* PCA 0 MODULE 5 CAPTURE/COMPARE - HIGH BYTE   */
__sfr __at (0xE8) ADC0CN          ;  /* ADC 0 CONTROL                                */
__sfr __at (0xE9) PCA0CPL2        ;  /* PCA 0 MODULE 2 CAPTURE/COMPARE - LOW BYTE    */
__sfr __at (0xEA) PCA0CPH2        ;  /* PCA 0 MODULE 2 CAPTURE/COMPARE - HIGH BYTE   */
__sfr __at (0xEB) PCA0CPL3        ;  /* PCA 0 MODULE 3 CAPTURE/COMPARE - LOW BYTE    */
```

```
__sfr __at (0xEC) PCA0CPH3        ;  /* PCA 0 MODULE 3 CAPTURE/COMPARE - HIGH BYTE    */
__sfr __at (0xED) PCA0CPL4        ;  /* PCA 0 MODULE 4 CAPTURE/COMPARE - LOW BYTE     */
__sfr __at (0xEE) PCA0CPH4        ;  /* PCA 0 MODULE 4 CAPTURE/COMPARE - HIGH BYTE    */
__sfr __at (0xEF) RSTSRC          ;  /* RESET SOURCE                                  */
__sfr __at (0xF8) SPI0CN          ;  /* SPI 0 CONTROL                                 */
__sfr __at (0xF9) PCA0L           ;  /* PCA 0 TIMER - LOW BYTE                        */
__sfr __at (0xFA) PCA0H           ;  /* PCA 0 TIMER - HIGH BYTE                       */
__sfr __at (0xFB) PCA0CPL0        ;  /* PCA 0 MODULE 0 CAPTURE/COMPARE - LOW BYTE     */
__sfr __at (0xFC) PCA0CPH0        ;  /* PCA 0 MODULE 0 CAPTURE/COMPARE - HIGH BYTE    */
__sfr __at (0xFD) PCA0CPL1        ;  /* PCA 0 MODULE 1 CAPTURE/COMPARE - LOW BYTE     */
__sfr __at (0xFE) PCA0CPH1        ;  /* PCA 0 MODULE 1 CAPTURE/COMPARE - HIGH BYTE    */

/*  Page 0x01 */
__sfr __at (0x88) CPT0CN          ;  /* COMPARATOR 0 CONTROL                          */
__sfr __at (0x89) CPT0MD          ;  /* COMPARATOR 0 CONFIGURATION                    */
__sfr __at (0x98) SCON1           ;  /* UART 1 CONTROL                                */
__sfr __at (0x99) SBUF1           ;  /* UART 1 BUFFER                                 */
__sfr __at (0xC8) TMR3CN          ;  /* TIMER 3 CONTROL                               */
__sfr __at (0xC9) TMR3CF          ;  /* TIMER 3 CONFIGURATION                         */
__sfr __at (0xCA) RCAP3L          ;  /* TIMER 3 CAPTURE REGISTER - LOW BYTE           */
__sfr __at (0xCB) RCAP3H          ;  /* TIMER 3 CAPTURE REGISTER - HIGH BYTE          */
__sfr __at (0xCC) TMR3L           ;  /* TIMER 3 - LOW BYTE                            */
__sfr __at (0xCD) TMR3H           ;  /* TIMER 3 - HIGH BYTE                           */
__sfr __at (0xD2) DAC1L           ;  /* DAC 1 REGISTER - LOW BYTE                     */
__sfr __at (0xD3) DAC1H           ;  /* DAC 1 REGISTER - HIGH BYTE                    */
__sfr __at (0xD4) DAC1CN          ;  /* DAC 1 CONTROL                                 */

/*  Page 0x02 */
__sfr __at (0x88) CPT1CN          ;  /* COMPARATOR 1 CONTROL                          */
__sfr __at (0x89) CPT1MD          ;  /* COMPARATOR 1 CONFIGURATION                    */
__sfr __at (0xBA) AMX2CF          ;  /* ADC 2 MUX CONFIGURATION                       */
__sfr __at (0xBB) AMX2SL          ;  /* ADC 2 MUX CHANNEL SELECTION                   */
```

```
__sfr __at (0xBC) ADC2CF          ;  /* ADC 2 CONFIGURATION                        */

__sfr __at (0xBE) ADC2            ;  /* ADC 2 DATA                                 */

__sfr __at (0xC4) ADC2GT          ;  /* ADC 2 GREATER-THAN REGISTER                */

__sfr __at (0xC6) ADC2LT          ;  /* ADC 2 LESS-THAN REGISTER                   */

__sfr __at (0xC8) TMR4CN          ;  /* TIMER 4 CONTROL                            */

__sfr __at (0xC9) TMR4CF          ;  /* TIMER 4 CONFIGURATION                      */

__sfr __at (0xCA) RCAP4L          ;  /* TIMER 4 CAPTURE REGISTER - LOW BYTE        */

__sfr __at (0xCB) RCAP4H          ;  /* TIMER 4 CAPTURE REGISTER - HIGH BYTE       */

__sfr __at (0xCC) TMR4L           ;  /* TIMER 4 - LOW BYTE                         */

__sfr __at (0xCD) TMR4H           ;  /* TIMER 4 - HIGH BYTE                        */

__sfr __at (0xE8) ADC2CN          ;  /* ADC 2 CONTROL                              */


/*  Page 0x03 */

__sfr __at (0x91) MAC0BL          ;  /* MAC0 B Register Low Byte                   */

__sfr __at (0x92) MAC0BH          ;  /* MAC0 B Register High Byte                  */

__sfr __at (0x93) MAC0ACC0        ;  /* MAC0 Accumulator Byte 0 (LSB)              */

__sfr __at (0x94) MAC0ACC1        ;  /* MAC0 Accumulator Byte 1                    */

__sfr __at (0x95) MAC0ACC2        ;  /* MAC0 Accumulator Byte 2                    */

__sfr __at (0x96) MAC0ACC3        ;  /* MAC0 Accumulator Byte 3 (MSB)              */

__sfr __at (0x97) MAC0OVR         ;  /* MAC0 Accumulator Overflow                  */

__sfr __at (0xC0) MAC0STA         ;  /* MAC0 Status Register                       */

__sfr __at (0xC1) MAC0AL          ;  /* MAC0 A Register Low Byte                   */

__sfr __at (0xC2) MAC0AH          ;  /* MAC0 A Register High Byte                  */

__sfr __at (0xC3) MAC0CF          ;  /* MAC0 Configuration                         */

__sfr __at (0xCE) MAC0RNDL        ;  /* MAC0 Rounding Register Low Byte            */

__sfr __at (0xCF) MAC0RNDH        ;  /* MAC0 Rounding Register High Byte           */


/*  Page 0x0F */

__sfr __at (0x88) FLSTAT          ;  /* FLASH STATUS                               */

__sfr __at (0x89) PLL0CN          ;  /* PLL 0 CONTROL                              */

__sfr __at (0x8A) OSCICN          ;  /* INTERNAL OSCILLATOR CONTROL                */

__sfr __at (0x8B) OSCICL          ;  /* INTERNAL OSCILLATOR CALIBRATION            */
```

```
__sfr __at (0x8C) OSCXCN          ;  /* EXTERNAL OSCILLATOR CONTROL                */
__sfr __at (0x8D) PLL0DIV         ;  /* PLL 0 DIVIDER                              */
__sfr __at (0x8E) PLL0MUL         ;  /* PLL 0 MULTIPLIER                           */
__sfr __at (0x8F) PLL0FLT         ;  /* PLL 0 FILTER                               */
__sfr __at (0x96) SFRPGCN         ;  /* SFR PAGE CONTROL                           */
__sfr __at (0x97) CLKSEL          ;  /* SYSTEM CLOCK SELECT                        */
__sfr __at (0x9A) CCH0MA          ;  /* CACHE MISS ACCUMULATOR                     */
__sfr __at (0x9C) P4MDOUT         ;  /* PORT 4 OUTPUT MODE                         */
__sfr __at (0x9D) P5MDOUT         ;  /* PORT 5 OUTPUT MODE                         */
__sfr __at (0x9E) P6MDOUT         ;  /* PORT 6 OUTPUT MODE                         */
__sfr __at (0x9F) P7MDOUT         ;  /* PORT 7 OUTPUT MODE                         */
__sfr __at (0xA1) CCH0CN          ;  /* CACHE CONTROL                              */
__sfr __at (0xA2) CCH0TN          ;  /* CACHE TUNING REGISTER                      */
__sfr __at (0xA3) CCH0LC          ;  /* CACHE LOCK                                 */
__sfr __at (0xA4) P0MDOUT         ;  /* PORT 0 OUTPUT MODE                         */
__sfr __at (0xA5) P1MDOUT         ;  /* PORT 1 OUTPUT MODE                         */
__sfr __at (0xA6) P2MDOUT         ;  /* PORT 2 OUTPUT MODE CONFIGURATION           */
__sfr __at (0xA7) P3MDOUT         ;  /* PORT 3 OUTPUT MODE CONFIGURATION           */
__sfr __at (0xAD) P1MDIN          ;  /* PORT 1 INPUT MODE                          */
__sfr __at (0xB7) FLACL           ;  /* FLASH ACCESS LIMIT                         */
__sfr __at (0xC8) P4              ;  /* PORT 4                                     */
__sfr __at (0xD8) P5              ;  /* PORT 5                                     */
__sfr __at (0xE1) XBR0            ;  /* CROSSBAR CONFIGURATION REGISTER 0          */
__sfr __at (0xE2) XBR1            ;  /* CROSSBAR CONFIGURATION REGISTER 1          */
__sfr __at (0xE3) XBR2            ;  /* CROSSBAR CONFIGURATION REGISTER 2          */
__sfr __at (0xE8) P6              ;  /* PORT 6                                     */
__sfr __at (0xF8) P7              ;  /* PORT 7                                     */


/*  WORD/DWORD Registers  */


/*  Page 0x00 */
```

```
__sfr16 __at (0x8C8A) TMR0         ;  /* TIMER 0 COUNTER                                */
__sfr16 __at (0x8D8B) TMR1         ;  /* TIMER 1 COUNTER                                */
__sfr16 __at (0xCDCC) TMR2         ;  /* TIMER 2 COUNTER                                */
__sfr16 __at (0xCBCA) RCAP2        ;  /* TIMER 2 CAPTURE REGISTER WORD                  */
__sfr16 __at (0xBFBE) ADC0         ;  /* ADC 0 DATA WORD                                */
__sfr16 __at (0xC5C4) ADC0GT       ;  /* ADC 0 GREATER-THAN REGISTER WORD               */
__sfr16 __at (0xC7C6) ADC0LT       ;  /* ADC 0 LESS-THAN REGISTER WORD                  */
__sfr16 __at (0xD3D2) DAC0         ;  /* DAC 0 REGISTER WORD                            */
__sfr16 __at (0xFAF9) PCA0         ;  /* PCA 0 TIMER COUNTER                            */
__sfr16 __at (0xFCFB) PCA0CP0      ;  /* PCA 0 MODULE 0 CAPTURE/COMPARE WORD            */
__sfr16 __at (0xFEFD) PCA0CP1      ;  /* PCA 0 MODULE 1 CAPTURE/COMPARE WORD            */
__sfr16 __at (0xEAE9) PCA0CP2      ;  /* PCA 0 MODULE 2 CAPTURE/COMPARE WORD            */
__sfr16 __at (0xECEB) PCA0CP3      ;  /* PCA 0 MODULE 3 CAPTURE/COMPARE WORD            */
__sfr16 __at (0xEEED) PCA0CP4      ;  /* PCA 0 MODULE 4 CAPTURE/COMPARE WORD            */
__sfr16 __at (0xE2E1) PCA0CP5      ;  /* PCA 0 MODULE 5 CAPTURE/COMPARE WORD            */

/*  Page 0x01 */
__sfr16 __at (0xCDCC) TMR3         ;  /* TIMER 3 COUNTER                                */
__sfr16 __at (0xCBCA) RCAP3        ;  /* TIMER 3 CAPTURE REGISTER WORD                  */
__sfr16 __at (0xD3D2) DAC1         ;  /* DAC 1 REGISTER WORD                            */

/*  Page 0x02 */
__sfr16 __at (0xCDCC) TMR4         ;  /* TIMER 4 COUNTER                                */
__sfr16 __at (0xCBCA) RCAP4        ;  /* TIMER 4 CAPTURE REGISTER WORD                  */

/*  Page 0x03 */
__sfr16 __at (0xC2C1) MAC0A        ;  /* MAC0 A Register                                */
               /* No sfr16 definition for MAC0B because MAC0BL must be written last */
__sfr32 __at (0x96959493) MAC0ACC  ;  /* MAC0 Accumulator                               */
__sfr16 __at (0xCFCE) MAC0RND      ;  /* MAC0 Rounding Register                         */
```

```
/*  BIT Registers  */


/*  P0  0x80 */
__sbit __at (0x80) P0_0           ;
__sbit __at (0x81) P0_1           ;
__sbit __at (0x82) P0_2           ;
__sbit __at (0x83) P0_3           ;
__sbit __at (0x84) P0_4           ;
__sbit __at (0x85) P0_5           ;
__sbit __at (0x86) P0_6           ;
__sbit __at (0x87) P0_7           ;


/*  TCON  0x88 */
__sbit __at (0x88) IT0            ;  /* EXT. INTERRUPT 0 TYPE                    */
__sbit __at (0x89) IE0            ;  /* EXT. INTERRUPT 0 EDGE FLAG               */
__sbit __at (0x8A) IT1            ;  /* EXT. INTERRUPT 1 TYPE                    */
__sbit __at (0x8B) IE1            ;  /* EXT. INTERRUPT 1 EDGE FLAG               */
__sbit __at (0x8C) TR0            ;  /* TIMER 0 ON/OFF CONTROL                   */
__sbit __at (0x8D) TF0            ;  /* TIMER 0 OVERFLOW FLAG                    */
__sbit __at (0x8E) TR1            ;  /* TIMER 1 ON/OFF CONTROL                   */
__sbit __at (0x8F) TF1            ;  /* TIMER 1 OVERFLOW FLAG                    */


/*  CPT0CN  0x88 */
__sbit __at (0x88) CP0HYN0        ;  /* COMPARATOR 0 NEGATIVE HYSTERESIS 0       */
__sbit __at (0x89) CP0HYN1        ;  /* COMPARATOR 0 NEGATIVE HYSTERESIS 1       */
__sbit __at (0x8A) CP0HYP0        ;  /* COMPARATOR 0 POSITIVE HYSTERESIS 0       */
__sbit __at (0x8B) CP0HYP1        ;  /* COMPARATOR 0 POSITIVE HYSTERESIS 1       */
__sbit __at (0x8C) CP0FIF         ;  /* COMPARATOR 0 FALLING EDGE INTERRUPT      */
__sbit __at (0x8D) CP0RIF         ;  /* COMPARATOR 0 RISING EDGE INTERRUPT       */
__sbit __at (0x8E) CP0OUT         ;  /* COMPARATOR 0 OUTPUT                      */
__sbit __at (0x8F) CP0EN          ;  /* COMPARATOR 0 ENABLE                      */
```

```
/*  CPT1CN  0x88 */

__sbit __at (0x88) CP1HYN0        ;  /* COMPARATOR 1 NEGATIVE HYSTERESIS 0          */

__sbit __at (0x89) CP1HYN1        ;  /* COMPARATOR 1 NEGATIVE HYSTERESIS 1          */

__sbit __at (0x8A) CP1HYP0        ;  /* COMPARATOR 1 POSITIVE HYSTERESIS 0          */

__sbit __at (0x8B) CP1HYP1        ;  /* COMPARATOR 1 POSITIVE HYSTERESIS 1          */

__sbit __at (0x8C) CP1FIF         ;  /* COMPARATOR 1 FALLING EDGE INTERRUPT         */

__sbit __at (0x8D) CP1RIF         ;  /* COMPARATOR 1 RISING EDGE INTERRUPT          */

__sbit __at (0x8E) CP1OUT         ;  /* COMPARATOR 1 OUTPUT                         */

__sbit __at (0x8F) CP1EN          ;  /* COMPARATOR 1 ENABLE                         */


/*  FLSTAT  0x88 */

__sbit __at (0x88) FLHBUSY        ;  /* FLASH BUSY                                  */


/*  P1  0x90 */

__sbit __at (0x90) P1_0           ;

__sbit __at (0x91) P1_1           ;

__sbit __at (0x92) P1_2           ;

__sbit __at (0x93) P1_3           ;

__sbit __at (0x94) P1_4           ;

__sbit __at (0x95) P1_5           ;

__sbit __at (0x96) P1_6           ;

__sbit __at (0x97) P1_7           ;


/*  SCON0  0x98 */

__sbit __at (0x98) RI0            ;  /* UART 0 RX INTERRUPT FLAG                    */

__sbit __at (0x98) RI             ;  /* UART 0 RX INTERRUPT FLAG                    */

__sbit __at (0x99) TI0            ;  /* UART 0 TX INTERRUPT FLAG                    */

__sbit __at (0x99) TI             ;  /* UART 0 TX INTERRUPT FLAG                    */

__sbit __at (0x9A) RB80           ;  /* UART 0 RX BIT 8                             */

__sbit __at (0x9B) TB80           ;  /* UART 0 TX BIT 8                             */

__sbit __at (0x9C) REN0           ;  /* UART 0 RX ENABLE                            */

__sbit __at (0x9C) REN            ;  /* UART 0 RX ENABLE                            */
```

```
__sbit __at (0x9D) SM20          ;  /* UART 0 MULTIPROCESSOR EN                    */
__sbit __at (0x9E) SM10          ;  /* UART 0 MODE 1                               */
__sbit __at (0x9F) SM00          ;  /* UART 0 MODE 0                               */

/*  SCON1  0x98 */
__sbit __at (0x98) RI1           ;  /* UART 1 RX INTERRUPT FLAG                    */
__sbit __at (0x99) TI1           ;  /* UART 1 TX INTERRUPT FLAG                    */
__sbit __at (0x9A) RB81          ;  /* UART 1 RX BIT 8                             */
__sbit __at (0x9B) TB81          ;  /* UART 1 TX BIT 8                             */
__sbit __at (0x9C) REN1          ;  /* UART 1 RX ENABLE                            */
__sbit __at (0x9D) MCE1          ;  /* UART 1 MCE                                  */
__sbit __at (0x9F) S1MODE        ;  /* UART 1 MODE                                 */

/*  P2  0xA0 */
__sbit __at (0xA0) P2_0          ;
__sbit __at (0xA1) P2_1          ;
__sbit __at (0xA2) P2_2          ;
__sbit __at (0xA3) P2_3          ;
__sbit __at (0xA4) P2_4          ;
__sbit __at (0xA5) P2_5          ;
__sbit __at (0xA6) P2_6          ;
__sbit __at (0xA7) P2_7          ;

/*  IE  0xA8 */
__sbit __at (0xA8) EX0           ;  /* EXTERNAL INTERRUPT 0 ENABLE                 */
__sbit __at (0xA9) ET0           ;  /* TIMER 0 INTERRUPT ENABLE                    */
__sbit __at (0xAA) EX1           ;  /* EXTERNAL INTERRUPT 1 ENABLE                 */
__sbit __at (0xAB) ET1           ;  /* TIMER 1 INTERRUPT ENABLE                    */
__sbit __at (0xAC) ES0           ;  /* UART0 INTERRUPT ENABLE                      */
__sbit __at (0xAC) ES            ;  /* UART0 INTERRUPT ENABLE                      */
__sbit __at (0xAD) ET2           ;  /* TIMER 2 INTERRUPT ENABLE                    */
__sbit __at (0xAF) EA            ;  /* GLOBAL INTERRUPT ENABLE                     */
```

```
/*  P3  0xB0 */

__sbit __at (0xB0) P3_0           ;

__sbit __at (0xB1) P3_1           ;

__sbit __at (0xB2) P3_2           ;

__sbit __at (0xB3) P3_3           ;

__sbit __at (0xB4) P3_4           ;

__sbit __at (0xB5) P3_5           ;

__sbit __at (0xB6) P3_6           ;

__sbit __at (0xB7) P3_7           ;


/*  IP  0xB8 */

__sbit __at (0xB8) PX0            ;  /* EXTERNAL INTERRUPT 0 PRIORITY               */

__sbit __at (0xB9) PT0            ;  /* TIMER 0 PRIORITY                            */

__sbit __at (0xBA) PX1            ;  /* EXTERNAL INTERRUPT 1 PRIORITY               */

__sbit __at (0xBB) PT1            ;  /* TIMER 1 PRIORITY                            */

__sbit __at (0xBC) PS0            ;  /* SERIAL PORT PRIORITY                        */

__sbit __at (0xBC) PS             ;  /* SERIAL PORT PRIORITY                        */

__sbit __at (0xBD) PT2            ;  /* TIMER 2 PRIORITY                            */


/* SMB0CN 0xC0 */

__sbit __at (0xC0) SMBTOE         ;  /* SMBUS 0 TIMEOUT ENABLE                      */

__sbit __at (0xC1) SMBFTE         ;  /* SMBUS 0 FREE TIMER ENABLE                   */

__sbit __at (0xC2) AA             ;  /* SMBUS 0 ASSERT/ACKNOWLEDGE FLAG             */

__sbit __at (0xC3) SI             ;  /* SMBUS 0 INTERRUPT PENDING FLAG              */

__sbit __at (0xC4) STO            ;  /* SMBUS 0 STOP FLAG                           */

__sbit __at (0xC5) STA            ;  /* SMBUS 0 START FLAG                          */

__sbit __at (0xC6) ENSMB          ;  /* SMBUS 0 ENABLE                              */

__sbit __at (0xC7) BUSY           ;  /* SMBUS 0 BUSY                                */


/* MAC0STA 0xC0 */

__sbit __at (0xC0) MAC0N          ;  /* MAC 0 NEGATIVE FLAG                         */
```

```
__sbit __at (0xC1) MAC0SO          ;  /* MAC 0 SOFT OVERFLOW FLAG                    */
__sbit __at (0xC2) MAC0Z           ;  /* MAC 0 ZERO FLAG                             */
__sbit __at (0xC3) MAC0HO          ;  /* MAC 0 HARD OVERFLOW FLAG                    */

/*  TMR2CN  0xC8 */
__sbit __at (0xC8) CPRL2           ;  /* TIMER 2 CAPTURE SELECT                      */
__sbit __at (0xC9) CT2             ;  /* TIMER 2 COUNTER SELECT                      */
__sbit __at (0xCA) TR2             ;  /* TIMER 2 ON/OFF CONTROL                      */
__sbit __at (0xCB) EXEN2           ;  /* TIMER 2 EXTERNAL ENABLE FLAG                */
__sbit __at (0xCE) EXF2            ;  /* TIMER 2 EXTERNAL FLAG                       */
__sbit __at (0xCF) TF2             ;  /* TIMER 2 OVERFLOW FLAG                       */

/*  TMR3CN  0xC8 */
__sbit __at (0xC8) CPRL3           ;  /* TIMER 3 CAPTURE SELECT                      */
__sbit __at (0xC9) CT3             ;  /* TIMER 3 COUNTER SELECT                      */
__sbit __at (0xCA) TR3             ;  /* TIMER 3 ON/OFF CONTROL                      */
__sbit __at (0xCB) EXEN3           ;  /* TIMER 3 EXTERNAL ENABLE FLAG                */
__sbit __at (0xCE) EXF3            ;  /* TIMER 3 EXTERNAL FLAG                       */
__sbit __at (0xCF) TF3             ;  /* TIMER 3 OVERFLOW FLAG                       */

/*  TMR4CN  0xC8 */
__sbit __at (0xC8) CPRL4           ;  /* TIMER 4 CAPTURE SELECT                      */
__sbit __at (0xC9) CT4             ;  /* TIMER 4 COUNTER SELECT                      */
__sbit __at (0xCA) TR4             ;  /* TIMER 4 ON/OFF CONTROL                      */
__sbit __at (0xCB) EXEN4           ;  /* TIMER 4 EXTERNAL ENABLE FLAG                */
__sbit __at (0xCE) EXF4            ;  /* TIMER 4 EXTERNAL FLAG                       */
__sbit __at (0xCF) TF4             ;  /* TIMER 4 OVERFLOW FLAG                       */

/*  P4  0xC8 */
__sbit __at (0xC8) P4_0            ;
__sbit __at (0xC9) P4_1            ;
__sbit __at (0xCA) P4_2            ;
```

```
__sbit __at (0xCB) P4_3            ;
__sbit __at (0xCC) P4_4            ;
__sbit __at (0xCD) P4_5            ;
__sbit __at (0xCE) P4_6            ;
__sbit __at (0xCF) P4_7            ;

/*  PSW  0xD0 */
__sbit __at (0xD0) P               ;  /* ACCUMULATOR PARITY FLAG                    */
__sbit __at (0xD1) F1              ;  /* USER FLAG 1                                */
__sbit __at (0xD2) OV              ;  /* OVERFLOW FLAG                              */
__sbit __at (0xD3) RS0             ;  /* REGISTER BANK SELECT 0                     */
__sbit __at (0xD4) RS1             ;  /* REGISTER BANK SELECT 1                     */
__sbit __at (0xD5) F0              ;  /* USER FLAG 0                                */
__sbit __at (0xD6) AC              ;  /* AUXILIARY CARRY FLAG                       */
__sbit __at (0xD7) CY              ;  /* CARRY FLAG                                 */

/* PCA0CN D8H */
__sbit __at (0xD8) CCF0            ;  /* PCA 0 MODULE 0 INTERRUPT FLAG              */
__sbit __at (0xD9) CCF1            ;  /* PCA 0 MODULE 1 INTERRUPT FLAG              */
__sbit __at (0xDA) CCF2            ;  /* PCA 0 MODULE 2 INTERRUPT FLAG              */
__sbit __at (0xDB) CCF3            ;  /* PCA 0 MODULE 3 INTERRUPT FLAG              */
__sbit __at (0xDC) CCF4            ;  /* PCA 0 MODULE 4 INTERRUPT FLAG              */
__sbit __at (0xDD) CCF5            ;  /* PCA 0 MODULE 5 INTERRUPT FLAG              */
__sbit __at (0xDE) CR              ;  /* PCA 0 COUNTER RUN CONTROL BIT              */
__sbit __at (0xDF) CF              ;  /* PCA 0 COUNTER OVERFLOW FLAG                */

/*  P5  0xD8 */
__sbit __at (0xD8) P5_0            ;
__sbit __at (0xD9) P5_1            ;
__sbit __at (0xDA) P5_2            ;
__sbit __at (0xDB) P5_3            ;
__sbit __at (0xDC) P5_4            ;
```

```
__sbit __at (0xDD) P5_5           ;
__sbit __at (0xDE) P5_6           ;
__sbit __at (0xDF) P5_7           ;

/* ADC0CN E8H */
__sbit __at (0xE8) AD0LJST        ;  /* ADC 0 RIGHT JUSTIFY DATA BIT                */
__sbit __at (0xE9) AD0WINT        ;  /* ADC 0 WINDOW INTERRUPT FLAG                 */
__sbit __at (0xEA) AD0CM0         ;  /* ADC 0 CONVERT START MODE BIT 0              */
__sbit __at (0xEB) AD0CM1         ;  /* ADC 0 CONVERT START MODE BIT 1              */
__sbit __at (0xEC) AD0BUSY        ;  /* ADC 0 BUSY FLAG                             */
__sbit __at (0xED) AD0INT         ;  /* ADC 0 EOC INTERRUPT FLAG                    */
__sbit __at (0xEE) AD0TM          ;  /* ADC 0 TRACK MODE                            */
__sbit __at (0xEF) AD0EN          ;  /* ADC 0 ENABLE                                */

/* ADC2CN E8H */
__sbit __at (0xE8) AD2WINT        ;  /* ADC 2 WINDOW INTERRUPT FLAG                 */
__sbit __at (0xE9) AD2CM0         ;  /* ADC 2 CONVERT START MODE BIT 0              */
__sbit __at (0xEA) AD2CM1         ;  /* ADC 2 CONVERT START MODE BIT 1              */
__sbit __at (0xEB) AD2CM2         ;  /* ADC 2 CONVERT START MODE BIT 2              */
__sbit __at (0xEC) AD2BUSY        ;  /* ADC 2 BUSY FLAG                             */
__sbit __at (0xED) AD2INT         ;  /* ADC 2 EOC INTERRUPT FLAG                    */
__sbit __at (0xEE) AD2TM          ;  /* ADC 2 TRACK MODE                            */
__sbit __at (0xEF) AD2EN          ;  /* ADC 2 ENABLE                                */

/*  P6  0xE8 */
__sbit __at (0xE8) P6_0           ;
__sbit __at (0xE9) P6_1           ;
__sbit __at (0xEA) P6_2           ;
__sbit __at (0xEB) P6_3           ;
__sbit __at (0xEC) P6_4           ;
__sbit __at (0xED) P6_5           ;
__sbit __at (0xEE) P6_6           ;
```

```
__sbit __at (0xEF) P6_7            ;


/* SPI0CN F8H */

__sbit __at (0xF8) SPIEN           ;  /* SPI 0 SPI ENABLE                    */

__sbit __at (0xF9) TXBMT           ;  /* SPI 0 TX BUFFER EMPTY FLAG          */

__sbit __at (0xFA) NSSMD0          ;  /* SPI 0 SLAVE SELECT MODE 0           */

__sbit __at (0xFB) NSSMD1          ;  /* SPI 0 SLAVE SELECT MODE 1           */

__sbit __at (0xFC) RXOVRN          ;  /* SPI 0 RX OVERRUN FLAG               */

__sbit __at (0xFD) MODF            ;  /* SPI 0 MODE FAULT FLAG               */

__sbit __at (0xFE) WCOL            ;  /* SPI 0 WRITE COLLISION FLAG          */

__sbit __at (0xFF) SPIF            ;  /* SPI 0 INTERRUPT FLAG                */


/*  P7  0xF8 */

__sbit __at (0xF8) P7_0            ;

__sbit __at (0xF9) P7_1            ;

__sbit __at (0xFA) P7_2            ;

__sbit __at (0xFB) P7_3            ;

__sbit __at (0xFC) P7_4            ;

__sbit __at (0xFD) P7_5            ;

__sbit __at (0xFE) P7_6            ;

__sbit __at (0xFF) P7_7            ;



/* Predefined SFR Bit Masks */


#define PCON_IDLE         0x01    /* PCON                                */

#define PCON_STOP         0x02    /* PCON                                */

#define ECCF              0x01    /* PCA0CPMn                            */

#define PWM               0x02    /* PCA0CPMn                            */

#define TOG               0x04    /* PCA0CPMn                            */

#define MAT               0x08    /* PCA0CPMn                            */

#define CAPN              0x10    /* PCA0CPMn                            */
```

```
#define CAPP             0x20    /* PCA0CPMn                          */

#define ECOM             0x40    /* PCA0CPMn                          */

#define PWM16            0x80    /* PCA0CPMn                          */

#define PINRSF           0x01    /* RSTSRC                            */

#define PORSF            0x02    /* RSTSRC                            */

#define MCDRSF           0x04    /* RSTSRC                            */

#define WDTRSF           0x08    /* RSTSRC                            */

#define SWRSF            0x10    /* RSTSRC                            */

#define C0RSEF           0x20    /* RSTSRC                            */

#define CNVRSEF          0x40    /* RSTSRC                            */



/* SFR PAGE DEFINITIONS */


#define CONFIG_PAGE      0x0F     /* SYSTEM AND PORT CONFIGURATION PAGE */

#define LEGACY_PAGE      0x00     /* LEGACY SFR PAGE                   */

#define TIMER01_PAGE     0x00     /* TIMER 0 AND TIMER 1               */

#define CPT0_PAGE        0x01     /* COMPARATOR 0                      */

#define CPT1_PAGE        0x02     /* COMPARATOR 1                      */

#define UART0_PAGE       0x00     /* UART 0                            */

#define UART_PAGE       0x01     /* UART 1                            */

#define SPI0_PAGE        0x00     /* SPI 0                             */

#define EMI0_PAGE        0x00     /* EXTERNAL MEMORY INTERFACE         */

#define ADC0_PAGE        0x00     /* ADC 0                             */

#define ADC2_PAGE        0x02     /* ADC 2                             */

#define SMB0_PAGE        0x00     /* SMBUS 0                           */

#define TMR2_PAGE        0x00     /* TIMER 2                           */

#define TMR3_PAGE        0x01     /* TIMER 3                           */

#define TMR4_PAGE        0x02     /* TIMER 4                           */

#define DAC0_PAGE        0x00     /* DAC 0                             */

#define DAC1_PAGE        0x01     /* DAC 1                             */

#define PCA0_PAGE        0x00     /* PCA 0                             */
```

```
#define PLL0_PAGE         0x0F     /* PLL 0                            */
#define MAC0_PAGE         0x03     /* MULTIPLY / ACCUMULATE 0          */

#endif
```

c8051F120_io.h

```
/*-----------------------------------------------------------------------
;
;
;
;
; FILE NAME: C8051F120.H
; TARGET MCUs: C8051F120, F121, F122, F123, F124, F125, F126, F127
; DESCRIPTION: Switch and LED definitions
;
; REVISION 1.2
;
;-----------------------------------------------------------------------*/
#ifndef C8051F120_IO_H
#define C8051F120_IO_H

__sbit __at (0x96) LED;             // LED='1' means ON
                                    // To be used requires setting it to push--pull: P1MDOUT |=
0x40

__sbit __at (0xB7) SW2;             // SW2='0' means switch pressed
                                    // Requires open-drain: i.e. not setting that bit to 1 in
P3MDOUT



// Note: AB4 switches are accessible on CONFIG_PAGE SFR Page only
// Pins 1 through 4 of port P4 must be configured
// Leave alone Pins 0, and 5-7 of P4. Pin 0 can be reused in the future but
```

```
// Pins 5-7 of P4 are used by Ethernet controller and must not be changed
//
//  SFRPAGE = CONFIG_PAGE;      // set SFR page to page 0x0F from which switch P4 and AB4 LEDs
and Switches are controlled
//  P4 = P4 | 0x06;             // Make sure that bits P4.1 and P4.2 are set high to prevent
permament switch ON
//  P4MDOUT = P4MDOUT & 0xF9;   // Make sure that P4.1 and P4.2 are Open Collector for reading
switches
//  P4MDOUT = P4MDOUT | 0x18;   // Make sure that P4.3 and P4.4 are Push/Pull for driving LEDs
//  Set SFR Page again before using AB4_* if it is changed elesewhere in the program



__sbit __at (0xC9) AB4_SW1;         // SW1 on AB4 Board
__sbit __at (0xCA) AB4_SW2;         // SW2 on AB4 Board
                                    // Require not setting these bits to 1 in P4MDOUT
__sbit __at (0xCB) AB4_LED1;        // Green AB4 Board LED
__sbit __at (0xCC) AB4_LED2;        // Red AB4 Board LED
                                    // To be used require: P4MDOUT |= 0x18


#endif
```

main.c

```
// Program Framework Copyright (C) 2005 Silicon Laboratories, Inc.
// Program Additions Copyright (C) 2010 Aleksander Malinowski
// Program Modified 2011 by Nick DeTrempe
//
// Date: November 29, 2011
// Target: C8051F12x
// Version: 1.0
//
// Description:
// This program polls a flag set by the A/D interrupt routine whenever a
// conversion is complete.  A global variable is referenced to determine
// which of the 6 channels has new data.  When new data is received, it
```

```
// will be averaged with the 15 most recent samples so provide a filtered
// value.  This value will be sent out through the serial port. The receive
// bit for the UART is also being polled to determine if a character has been
// received.  When this occurs, a funciton will loop to continue receiving all
// characters until an end of line character is received.  Afterwards,
// the string is added to the command queue.
//
//    It runs on a C8051F120 board.
//
//
#include "C8051F120.h"                  // Device-specific SFR Definitions
#include "C8051F120_io.h"               // SFR declarations
#include "pwmint.h"
#include "adc0int8rr.h"
#include "am_init.h"
#include "am_com.h"
#include "am_wait.h"//remove from final program
#include <stdio.h>

//#define CTRL_SAMPLE_RATE     6L     // A/D Sample frequency in Hz
#define PWM_SAMPLE_RATE     10000L     // PWM interrupt rate
#define PWM_FRQ_RATE         1000L     // PWM frequency in Hz

__sbit __at (0x91) DIR1            ;//sets direction for actuator 1
__sbit __at (0x92) DIR2            ;//sets direction for actuator 2
__sbit __at (0x93) DIR3            ;//sets direction for actuator 3
__sbit __at (0xB3) windOn                                   ;//sets the wind tunnel on or off
__sbit __at (0xB4) dampOn                                   ;//sets the damper on or off
__sbit __at (0xB5) dampDir                            ;//sets the direction of the damper

#define BAUDRATE     57600L              // Baud rate of UART in bps
#define TEMP_CHANNEL  (8)               // which AD channel measures temperature
```

```
#define NUM_AVG 15
       //number of channels averaged

__xdata char buffer[100];//used to input data

__xdata char command[6][10];//command queue

__xdata long int ch0[NUM_AVG], ch1[NUM_AVG], ch2[NUM_AVG], ch3[NUM_AVG];//used to store sampled
data for average

__xdata long int ch4[NUM_AVG], ch5[NUM_AVG], ch6[NUM_AVG], ch7[NUM_AVG];

__xdata char ct0, ct1, ct2, ct3, ct4, ct5, ct6, ct7 = 0;//used to keep track of the current
sample in running average




char start;//start of the command queue

char stop;//stop of the command queue

char queueFull;

sbit state;

unsigned int tempSum;


void check_UART();

void moveAct(char act, unsigned int newPos);


void main(void)

{

       char i;

       unsigned int cmd;




       state = SW2;

    // Disable watchdog timer

    WDTCN = 0xde;

    WDTCN = 0xad;


    PORT_Init ();

    SYSCLK_Init();
```

```
UART_Quit();
UART_Init(SYSCLK, BAUDRATE);

ADC0_Timer3_Init(SYSCLK);

            Timer4_PWM_Init (SYSCLK, PWM_SAMPLE_RATE);

Timer4_PWM_SetDuty(0, 90);
Timer4_PWM_SetDuty(1, 90);
Timer4_PWM_SetDuty(2, 90);
            Timer4_PWM_SetFrequency(PWM_FRQ_RATE);
EA = 1;                             // Enable global interrupts
            LED = 0;
            start = 0;
            stop = 0;
            queueFull = 0;
            timeOut = 0;//flag to exit an actuator move command
            windOn = 0;
            dampOn = 0;
            dampDir = 0;
            newData = 0;

while (1)
{
                            if(newData){
         int converted = (long)result[chConv] * VREF / ADC0_MAX;
         switch (chConv)
         {
            case 7:  //Actuator 1 sensor
                    sprintf(buffer, "1%4d\n", converted );
                            ch0[ct0] = converted;
                                        ct0 = (ct0 + 1)%NUM_AVG;
```

```
                        converted = 0;
                        for(i = 0;i<NUM_AVG;i++){
                                converted+=ch0[i];
                        }
                        converted/=NUM_AVG;
                        sprintf(buffer, "1%4d\n", converted );
                        UART_puts(buffer);
        break;
case 0:  //Lift sensor
        ch1[ct1] = converted;
                        ct1 = (ct1 + 1)%NUM_AVG;
                        converted = 0;
                        for(i = 0;i<NUM_AVG;i++){
                                converted+=ch1[i];
                        }
                        converted/=NUM_AVG;
                        sprintf(buffer, "L%4d\n", converted );
        UART_puts(buffer);
        break;
case 2:  //Actuator 2 sensor
        ch2[ct2] = converted;
                        ct2 = (ct2 + 1)%NUM_AVG;
                        converted = 0;
                        for(i = 0;i<NUM_AVG;i++){
                                converted+=ch2[i];
                        }
                        converted/=NUM_AVG;
                        sprintf(buffer, "2%4d\n", converted);
        UART_puts(buffer);
        break;
                                case 3:  //Drag sensor
        ch3[ct3] = converted;
```

```
                        ct3 = (ct3 + 1)%NUM_AVG;
                        converted = 0;
                        for(i = 0;i<NUM_AVG;i++){
                                converted+=ch3[i];
                        }
                        converted/=NUM_AVG;
sprintf(buffer, "D%4d\n", converted );
UART_puts(buffer);
        break;
case 4:  //Actuator number 3  position
        ch4[ct4] = converted;
                        ct4 = (ct4 + 1)%NUM_AVG;
                        converted = 0;
                        for(i = 0;i<NUM_AVG;i++){
                                converted+=ch4[i];
                        }
                        converted/=NUM_AVG;
                        sprintf(buffer, "3%4d\n", converted );
        UART_puts(buffer);
        break;
case 5:  //Speed measurement
        ch5[ct5] = converted;
                        ct5 = (ct5 + 1)%NUM_AVG;
                        converted = 0;
                        for(i = 0;i<NUM_AVG;i++){
                                converted+=ch5[i];
                        }
                        converted/=NUM_AVG;
                        sprintf(buffer, "S%4d\n", converted );
        UART_puts(buffer);
        break;
case 6:  //Ambient air pressure
```

```
                        ch6[ct6] = converted;
                                ct6 = (ct6 + 1)%NUM_AVG;
                                converted = 0;
                                for(i = 0;i<NUM_AVG;i++){
                                        converted+=ch6[i];
                                }
                                converted/=NUM_AVG;
                                sprintf(buffer, "P%4d\n", converted );
                        UART_puts(buffer);
                        break;

                }
                newData = 0;
        }

        check_UART();
        if(start != stop){//command in queue
                if(command[0][start] =='L'||command[0][start]=='l'){
                        LED = !LED;
                }

                if(command[0][start] == '1'||command[0][start] ==
'2'||command[0][start] == '3'){//change position
                        cmd = (command[4][start]-48);
                        cmd += (command[3][start]-48)*10;
                        cmd += (command[2][start]-48)*100;
                        cmd += (command[1][start]-48)*1000;
                        if(cmd < 2400){
                                moveAct((command[0][start]-48), cmd);
                        }
                }
                if(command[0][start] == 'B'){//blower command
                        if(command[1][start] == 'O'){
```

```
                        //start the blower
                        windOn = 1;
                    }
                    if(command[1][start] == 'S'){
                        //stop the blower
                        windOn = 0;
                    }
                }
                if(command[0][start] == 'D'){//damper command
                    if(command[1][start] == 'O'){
                        //start opening the damper
                        dampDir = 1;//will be verified later
                        dampOn = 1;
                    }
                    if(command[1][start] == 'C'){
                        //start closing the damper
                        dampDir = 0;//will be verified later
                        dampOn = 1;
                    }
                    if(command[1][start] == 'S'){
                        //stop moving the damper
                        dampOn = 0;
                        dampDir = 0;
                    }
                }//ignore all other commands
                start = (start + 1)%10;
            }
            else{
                //UART_puts("I\n");
            }
    }
}
```

```
/*********************************************************************
this funciton polls RI to see if a character is being received
if a character has not been sent then the function will immediately
return.  Otherwise, it will take the character and store it in the
command array and return.  If the received character constitutes
a complete command then this function will update the command queue
accordingly.  Note that a command ends by either receiving a null
character, an end of line character or maxing out the array (6 chars).
*********************************************************************/

void check_UART(){
        static char i = 0;//used to determine which letter of command has been received
        char SFRPAGE_SAVE = SFRPAGE;
  SFRPAGE = UART_PAGE;
  if(RI == 1){
                                UART_gets(buffer, sizeof(buffer));
                SFRPAGE = SFRPAGE_SAVE;
                if(queueFull==0){//add to queue if not full, ignore otherwise
                        for(i=0;i<6;i++){
                                command[i][stop]=buffer[i];
                                if(buffer[i]=='\n'||buffer[i]=='\r'||buffer[i]==0)break;
                                }
                                if((stop + 1)%10 != start){
                                        stop = (stop + 1)%10;
                                }else{
                                        queueFull = 1;
                                }
                        }
        }else{
        SFRPAGE = SFRPAGE_SAVE;
   }
 }
```

```
//this function will reposition an actuator
//While positioning, it will still poll the serial port
//for characters to be received.  Also, it will poll
//a flag that will be set by timer for if the PWM has
//been generated for more than 90 seconds.  After 90
//seconds of continuous PWM, this funciton will shut off
//the PWM and return.

void moveAct(char act, unsigned int newPos){
        char i;
        unsigned int curPos;
        unsigned int checkPos;//checks for movement every 3 seconds
        unsigned int dataCount=0;//counts number of data samples received
        timeOut = 0;//ensure this is cleared
        switch(act)
        {
                case 1:
                        curPos = (long)result[0] * VREF / ADC0_MAX;
                        checkPos = curPos;
                        if (newPos > curPos){//retract
                                DIR1 = 0;
                                Timer4_PWM_SetOn(0, ON);
                                while (newPos > curPos){//retract
                                        check_UART();//still poll for new commands
           if(newData){
                if(chConv==0){
                        dataCount++;
                        ch0[ct0] = (long)result[chConv] * VREF / ADC0_MAX;;
                                ct0 = (ct0 + 1)%NUM_AVG;
                                curPos = 0;
                                for(i = 0;i<NUM_AVG;i++){
```

```
                                curPos+=ch0[i];
                        }
                        curPos/=NUM_AVG;
                }
        }
        sprintf(buffer, "1%4d\n", curPos);
UART_puts(buffer);
                        if(timeOut)break;//safety precaution
                        if(dataCount == 300){
                                if((curPos-checkPos)<50){
                                        break;
                                }else{
                                        dataCount =0;
                                        checkPos = curPos;
                                }
                        }

                }
                Timer4_PWM_SetOn(0, OFF);
        }else{//extend
                DIR1 = 1;
                Timer4_PWM_SetOn(0, ON);
                while (newPos < curPos){
                        check_UART();
if(newData){
     if(chConv==0){
        dataCount++;
        ch0[ct0] = (long)result[chConv] * VREF / ADC0_MAX;
                        ct0 = (ct0 + 1)%NUM_AVG;
                        curPos = 0;
                        for(i = 0;i<NUM_AVG;i++){
                                curPos+=ch0[i];
```

```
                        }
                        curPos/=NUM_AVG;
                    }
                }
                sprintf(buffer, "1%4d\n", curPos);
UART_puts(buffer);
                        if(timeOut)break;//safety precaution
                        if(dataCount == 300){
                            if((checkPos-curPos)<50){
                                break;
                            }else{
                                dataCount =0;
                                checkPos = curPos;
                            }
                        }
                    }
                    Timer4_PWM_SetOn(0, OFF);
                };
                break;


        case 2:
                curPos = (long)result[2] * VREF / ADC0_MAX;
                checkPos = curPos;
                if (newPos > curPos){//retract
                    DIR2 = 0;
                    Timer4_PWM_SetOn(1, ON);
                    while (newPos > curPos){//retract
                        check_UART();//still poll for new commands
if(newData){
        if(chConv==2){
                dataCount++;
                ch2[ct2] = (long)result[chConv] * VREF / ADC0_MAX;;
```

```
                        ct2 = (ct2 + 1)%NUM_AVG;
                        curPos = 0;
                        for(i = 0;i<NUM_AVG;i++){
                                curPos+=ch2[i];
                        }
                        curPos/=NUM_AVG;
                }
        }
        sprintf(buffer, "2%4d\n", curPos);
UART_puts(buffer);
                        if(timeOut)break;//safety precaution
                        if(dataCount == 300){
                                if((curPos-checkPos)<50){
                                        break;
                                }else{
                                        dataCount =0;
                                        checkPos = curPos;
                                }
                        }
                }
                Timer4_PWM_SetOn(1, OFF);
        }else{//extend
                DIR2 = 1;
                Timer4_PWM_SetOn(1, ON);
                while (newPos < curPos){
                        check_UART();
if(newData){
     if(chConv==2){
        dataCount++;
        ch2[ct2] = (long)result[chConv] * VREF / ADC0_MAX;;
                        ct2 = (ct2 + 1)%NUM_AVG;
                        curPos = 0;
```

```
                        for(i = 0;i<NUM_AVG;i++){
                                curPos+=ch2[i];
                        }
                        curPos/=NUM_AVG;
                }
        }
        sprintf(buffer, "2%4d\n", curPos);
UART_puts(buffer);
                        if(timeOut)break;//safety precaution
                        if(dataCount == 300){
                                if((checkPos-curPos)<50){
                                        break;
                                }else{
                                        dataCount =0;
                                        checkPos = curPos;
                                }
                        }
                }
                Timer4_PWM_SetOn(1, OFF);
        };
        break;


    case 3:
        curPos = (long)result[4] * VREF / ADC0_MAX;
        checkPos = curPos;
        if (newPos > curPos){//retract
                DIR3 = 0;
                Timer4_PWM_SetOn(2, ON);
                while (newPos > curPos){//retract
                        check_UART();//still poll for new commands
if(newData){
    if(chConv==4){
```

```
                dataCount++;
                ch4[ct4] = (long)result[chConv] * VREF / ADC0_MAX;;
                                ct4 = (ct4 + 1)%NUM_AVG;
                                curPos = 0;
                                for(i = 0;i<NUM_AVG;i++){
                                        curPos+=ch4[i];
                                }
                                curPos/=NUM_AVG;
                        }
                }
                sprintf(buffer, "3%4d\n", curPos);
UART_puts(buffer);
                                if(timeOut)break;//safety precaution
                                if(dataCount == 300){
                                        if((curPos-checkPos)<50){
                                                break;
                                        }else{
                                                dataCount =0;
                                                checkPos = curPos;
                                        }
                                }
                        }
                        Timer4_PWM_SetOn(2, OFF);
                }else{//extend
                        DIR3 = 1;
                        Timer4_PWM_SetOn(2, ON);
                        while (newPos < curPos){
                                check_UART();
if(newData){
     if(chConv==4){
                dataCount++;
                ch4[ct4] = (long)result[chConv] * VREF / ADC0_MAX;;
```

```
                                        ct4 = (ct4 + 1)%NUM_AVG;
                                        curPos = 0;
                                        for(i = 0;i<NUM_AVG;i++){
                                                curPos+=ch4[i];
                                        }
                                        curPos/=NUM_AVG;
                                }
                        }
                        sprintf(buffer, "3%4d\n", curPos);
        UART_puts(buffer);
                                        if(timeOut)break;//safety precaution
                                        if(dataCount == 300){
                                                if((checkPos-curPos)<50){
                                                        break;
                                                }else{
                                                        dataCount =0;
                                                        checkPos = curPos;
                                                }
                                        }
                                }
                                Timer4_PWM_SetOn(2, OFF);
                        };
                        break;
                }
        }
```

pwminit.c

```
#include "pwmint.h"

#include "C8051F120.h"                  // SFR declarations
#include "C8051F120_io.h"               // SFR declarations
```

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

// Hardware IO CONSTANTS

//-----------------------------------------------------------------------------

__sbit __at (0xB0) PWMout0;             // output bit 0

__sbit __at (0xB1) PWMout1;             // output bit 1

__sbit __at (0xB2) PWMout2;             // output bit 2


//-----------------------------------------------------------------------------

// Global CONSTANTS

//-----------------------------------------------------------------------------

#define PHASE_PREC 65535                // range of phase accumulator

#define PHASE_HALF 32768                // half of range of phase accumulator


//-----------------------------------------------------------------------------

// Global Variables

//-----------------------------------------------------------------------------


unsigned int sampling    = 50000;       // sampling frequency of output in Hz, defaults to 50 kHz

unsigned int frequency   = 1000;        // frequency of output in Hz, defaults to 1000 Hz

unsigned int phase_add   = 1000L * PHASE_PREC / 50000L; // for 1kHz signal and 50kHz sampling
rate

unsigned long timeCount;                                                    //used as a
safety to ensure that the PWM isn't on too long

PWMstate     output_waveform[6] = {OFF, OFF, OFF, OFF, OFF, OFF};    // channel off/on

unsigned int dutycount[6]       = {PHASE_HALF, PHASE_HALF, PHASE_HALF, PHASE_HALF, PHASE_HALF,
PHASE_HALF};

sbit timeOut;//used to signal an actuator move timeout

// duty cycle in timer ticks, defaults to 50%


//-----------------------------------------------------------------------------

// Timer4_PWM_Init
```

```
//-----------------------------------------------------------------------------
//
// Configure Timer4 to auto-reload and generate an interrupt at interval
// specified by <counts> using SYSCLK/12 as its time base.
//
void Timer4_PWM_Init (unsigned long sysclock, unsigned long rate)
{
    char SFRPAGE_SAVE = SFRPAGE;        // Save Current SFR page
    long counts = sysclock/(12*rate);   // Note that Timer4 is connected to SYSCLK/12

    sampling = rate;
    phase_add  = (frequency * PHASE_PREC) / sampling;

    SFRPAGE = CONFIG_PAGE;              // set SFR page
    P3MDOUT |= 0x3F;                    // Set P3.0 through P3.5 to push-pull

    SFRPAGE = TMR4_PAGE;                // set SFR page
    TMR4CN  = 0x00;                     // Stop Timer4; Clear TF4;
    TMR4CF  = 0x00;                     // use SYSCLK/12 as timebase
//  TMR4CF  = 0x08;                     // use SYSCLK as timebase
    RCAP4   = -counts;                  // set reload value
    TMR4    = RCAP4;                    // set starting value
    EIE2   |= 0x04;                     // enable Timer4 interrupts - bit 00000100 or ET4 = 1;
    TMR4CN |= 0x04;                     // start Timer4

    SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}

//-----------------------------------------------------------------------------
// Parameter Control Functions
//-----------------------------------------------------------------------------
```

```
void Timer4_PWM_SetFrequency(unsigned long newfrequency)
{
    __bit EA_SAVE     = EA;             // Preserve Current Interrupt Status
    EA = 0;                             // disable interrupts
    timeCount=0;
    if (newfrequency<1) newfrequency=1;
    frequency = newfrequency;
    phase_add = (long)frequency * PHASE_PREC / sampling;
    EA = EA_SAVE;                       // restore interrupts
}

void Timer4_PWM_SetOn(unsigned char channel, PWMstate newstate)
{
    __bit EA_SAVE     = EA;            // Preserve Current Interrupt Status
    if(newstate == ON)timeCount = 0;
    EA = 0;                             // disable interrupts
    output_waveform[channel] = newstate;
    EA = EA_SAVE;                       // restore interrupts
}

void Timer4_PWM_SetDuty(unsigned char channel, unsigned char newduty) {
    __bit EA_SAVE     = EA;             // Preserve Current Interrupt Status
    EA = 0;                             // disable interrupts
    dutycount[channel] = (long)PHASE_PREC * newduty / 100;
    EA = EA_SAVE;                       // restore interrupts
}

//-------------------------------------------------------------------------------------
// Interrupt Service Routines
//-------------------------------------------------------------------------------------

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

```
// Timer4_PWM_ISR -- Wave Generator
//-----------------------------------------------------------------------------
//
// This ISR is called on Timer4 overflows.  Timer4 is set to auto-reload mode
// and is used to schedule the DAC output sample rate in this example.
// Note that the value that is written to DAC0 during this ISR call is
// actually transferred to DAC0 at the next Timer4 overflow.
//
void Timer4_PWM_ISR (void) __interrupt 16 __using 3
{

    static unsigned phase_acc = 0;  // holds phase accumulator, Note: will roll over at 65536

    SFRPAGE = TMR4_PAGE;            // set SFR page
    TMR4CN &= ~0x80;                // clear T4 overflow flag

    phase_acc += phase_add;         // increment phase accumulator

    if ( (output_waveform[0] == ON) && (phase_acc < dutycount[0]) ) { PWMout0 = 1; } else {
PWMout0 = 0; }
    if ( (output_waveform[1] == ON) && (phase_acc < dutycount[1]) ) { PWMout1 = 1; } else {
PWMout1 = 0; }
    if ( (output_waveform[2] == ON) && (phase_acc < dutycount[2]) ) { PWMout2 = 1; } else {
PWMout2 = 0; }
    //if ( (output_waveform[3] == ON) && (phase_acc < dutycount[3]) ) { PWMout3 = 1; } else {
PWMout3 = 0; }
   // if ( (output_waveform[4] == ON) && (phase_acc < dutycount[4]) ) { PWMout4 = 1; } else {
PWMout4 = 0; }
    //if ( (output_waveform[5] == ON) && (phase_acc < dutycount[5]) ) { PWMout5 = 1; } else {
PWMout5 = 0; }
          if(timeCount++ == 10000L*90L)timeOut=1;//the pwm has should be done by now
    // This is an interrupt, the original SFR page will be restored upon return from it
}
```

pwminit.h

```
#ifndef  _PWMINT

#define  _PWMINT


typedef enum PWMstate { OFF, ON } PWMstate;


void Timer4 PWM Init         (unsigned long sysclock, unsigned long rate);

void Timer4 PWM SetFrequency(unsigned long newfrequency);


void Timer4 PWM SetOn        (unsigned char channel, PWMstate newstate);          // 0..5, 0 or 1

void Timer4 PWM SetDuty      (unsigned char channel, unsigned char newduty);      // 0..5, 0..100


void Timer4 PWM ISR          (void)    interrupt 16    using 3;


extern sbit timeOut;


#endif
```

Appendix D: Java Server Source Code

COMPort.java

```
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.util.Enumeration;
import javax.comm.*;

public class COMPort {

    public COMPort(String portname) throws Exception {
        init(portname, 2400, 100);
    }

    public COMPort(String portname, int rate) throws Exception {
        init(portname, rate, 100);
    }

    public COMPort(String portname, int rate, int timeout) throws Exception {
        init(portname, rate, timeout);
    }

    public void setHardwareFlowControlMode(boolean enable)  throws Exception {
        if (enable)
            sPort.setFlowControlMode(
                SerialPort.FLOWCONTROL_NONE
            );
        else
            sPort.setFlowControlMode(
                SerialPort.FLOWCONTROL_RTSCTS_IN |
```

```
            SerialPort.FLOWCONTROL_RTSCTS_OUT
        );

}

public InputStream getInputStream() {
    return inStream;
}

public OutputStream getOutputStream() {
    return outStream;
}

public SerialPort getSerialPort() {
    return sPort;
}

public void close() throws IOException {
    sPort.close();
}




private InputStream inStream=null;
private OutputStream outStream=null;
private SerialPort sPort=null;;

private void init(String portname, int rate, int timeout) throws Exception {

    CommPortIdentifier portId=null;

    boolean  located=false;
```

```
        Enumeration portList = CommPortIdentifier.getPortIdentifiers();
        if (!portList.hasMoreElements()) throw new IOException("no com port found");
        while (portList.hasMoreElements()) {
            portId = (CommPortIdentifier) portList.nextElement();
            // System.err.println("CHECKING: "+portId.getName());
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                if (portId.getName().equals(portname)) {
                    located=true; break;
                }
            }
        }
        if (!located) throw new IOException("port "+portname+" not found");
        // System.err.println("FOUND: "+portId.getName());

        sPort = (SerialPort) portId.open("Java Simplified COM Port Access", timeout);
        sPort.setSerialPortParams(rate,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE);
           outStream = sPort.getOutputStream();
        inStream = sPort.getInputStream();
    }
}
```

EmbedImplemenation.java

```
//* A sample implementation of the interface to the embedded system *
import java.io.InputStream;
import java.io.OutputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.*;
import java.io.*;
```

```
import java.io.FilterOutputStream;
import java.io.PrintStream;
import java.lang.Thread;

class EmbedImplementation implements EmbedInterface {
    private final String     comport     = "COM1";   // COM port name
    private final int        comspeed    = 57600;     // COM port speed
    private final int        bufSize     = 256;      // max size of the embedded buffer

    private COMPort          com =   null;
    // private static InputStream   in  =   null;
    private BufferedReader   in  =   null;
    private OutputStream     os =   null;
        //private PrintStream  os = null;

    public void Init() throws Exception {
        // no try-catch blocks here:
        // any exception that occurs here
        // is passed to the main program
        com = new COMPort(comport, comspeed);
        com.setHardwareFlowControlMode(true);
        // in = com.getInputStream();
        in = new BufferedReader(new InputStreamReader(com.getInputStream()));
        os = com.getOutputStream();
               //os = new PrintStream(com.getOutputStream(), true);
    }

    public void Quit() throws Exception {
/*      try {
            in.close();
        } catch (Exception e) {}
        try {
```

```
        out.close();
    } catch (Exception e) {} */
    try {
        com.close();
    } catch (Exception e) {}
    // we can but do not have to pass any exception
}

public void   Start() throws Exception {}
public void   Stop()  throws Exception {}

public synchronized String Send(String command) {
    // make sure that you don't send more than your embedded system buffer !!!
    if (command.length()>bufSize) {
        System.out.println("ERROR: attempt to overrun the embedded buffer");
        return("-ERR data sequence too long");
    }
    try {
                byte temp[];
        // ** PROCESS THE COMMAND HERE ***
        // for the sake of debugging you may want to print something to the console
                System.out.println("Got to EMBED.Send with this command: "+command);
        //os.write(command.getBytes());
                //command = command+"\r\n";
                //os.write(command.getBytes());
                System.out.println("length: " +command.length());
        temp = command.getBytes();
                String eol = "\r\n";
                for(int i = 0;i<command.length();i++){
                System.out.println(temp[i]);
                if(i == 0){
                Thread.sleep(200);}
```

```
                    os.write(temp[i]);
                    os.flush();
                    }
                    os.write(eol.getBytes());
                    os.flush();
                    System.out.println("Sent: ( "+command+ " ) Successfully");

                    //System.out.println(in.readLine());


        } catch (Exception e) {
                 System.out.println("ERROR: SERIAL PORT: "+e);
            return("-ERR");
        }
        //return("+OK sent: "+command);
                return(command);
    }

    public String Recv() {
        try {
            // ** PROCESS THE RECEIVED DATA HERE ***
            return(in.readLine());
        } catch (Exception e) {
                 System.out.println("Error: "+e);
            // for the sake of debugging you may want to print something to the console
            return("-ERR cannot read from the s-link");
        }
    }
}
```

ReferenceTCPIPServer.java

```
//* A Reference TCP/IP Server for Remote Control *

import java.io.PrintWriter;

import java.io.FileWriter;

import java.io.InputStream;

import java.io.OutputStream;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.net.Socket;

import java.net.ServerSocket;

import java.util.Date;

import java.io.*;



// NETWORK SERVER

public class ReferenceTCPIPServer {

    private static final int        maxque      = 3;        // but we allow a few more clients to
try

    private static final int        port        = 8051;     // listen to this port


    private static ServerSocket     SERVER=null;

    private static ReportToClients  FEEDBACK=null;

            static EmbedInterface   EMBED=null;

    private static int              exitlevel=0;


    public static void main(String A[]) {

        try {

            System.out.println("Starting the embeded system ...");

            EMBED=new EmbedImplementation();

            EMBED.Init();

            System.out.println("Starting the feedback thread ...");

            FEEDBACK=new ReportToClients();

            FEEDBACK.start();
```

```
    System.out.println("Starting the network server ...");
    SERVER= new ServerSocket(port, maxque);

    System.out.println("NETWORK XXX SERVER STARTED!");
    Log("Server started");
    // Server main loop
    while (true) {
        Socket client=SERVER.accept();
        // The variable with thread is created
        // and started when a new connection is made
        try {
            new ServiceClient(client);
        } catch (Exception e) {
            System.out.println("ERROR: (Embed/Client) "+e);
        }
    }
} catch (Exception e) {
    System.out.println("ERROR: (Server) "+e);
}
System.out.println("NETWORK XXX SERVER STOPPED!");
Log("Server stopped");
System.out.println("Disconnecting all remote clients ...");
ServiceClient.KillAll();
System.out.println("Shutting down the embeded system ...");
try {
    EMBED.Quit();
} catch (Exception e) {
    System.out.println("SHUTDOWN: (Server) "+e);
}
System.out.println("NETWORK XXX SERVER TERMINATED!");
Log("Server terminated");
System.exit(exitlevel);
```

```
    }

    public static void terminate(int level) {
        System.out.println("Shutting down the server ...");
        System.out.println("Shutting down the feedback thread ...");
        FEEDBACK.stop();
        System.out.println("Shutting down the network service ...");
        try {
            SERVER.close();
        } catch (Exception e) {
            System.out.println("SHUTDOWN: (Server) "+e);
        }
        exitlevel=level;
    }

    public synchronized static void ShowAndLog(String memo) {
        System.out.println(memo);
        Log(memo);
    }

    public synchronized static void Log(String memo) {
        try {
            PrintWriter LOG=new PrintWriter(new FileWriter("xxxLOG.txt", true));
            LOG.println((new Date()).toString()+" : "+memo);
            LOG.flush();
            LOG.close();
        } catch (Exception e) {
            System.out.println("ERROR: (log file) "+e);
        }
    }
}
```

```
// PROCESS A CLIENT CONNECTION
class ServiceClient implements Runnable {
        //Control Flag
        int controlflag = -1;

    // array of connected clients
    private static final int        maxcon = 10;
    private static ServiceClient    CLI[]=new ServiceClient[maxcon];
    static {
        int c; for (c=0; c<ServiceClient.CLI.length; c++) CLI[c]=null;
    }

    // Send message back to all clients
    static void SendAll(String msg) {
        int c;
        for (c=0; c<ServiceClient.CLI.length; c++)
            if (CLI[c]!=null)
                CLI[c].Send(msg);
    }

    // Disconnect all clients
    static void KillAll() {
        int c;
        for (c=0; c<ServiceClient.CLI.length; c++)
            if (CLI[c]!=null) {
                try {
                    (CLI[c].SO).close();
                } catch (Exception e) {}
                CLI[c]=null;
            }
    }
```

```
private int             CO=0;
private Socket          SO=null;
private BufferedReader  SR=null;
private PrintWriter     SW=null;
private String          CLIADDR=""; // client's address

ServiceClient(Socket client) throws Exception {
    SO=client;
    // waking up the embedded system if this is the first connection
    if (getOnlineCount()==0)
        ReferenceTCPIPServer.EMBED.Start();
    // searching for an empty entry
    for (CO=0; CO<ServiceClient.CLI.length; CO++)
        if (CLI[CO]==null)
            break;
    if (CO>=ServiceClient.CLI.length)  CO=-1;
    else                               CLI[CO]=this;
    (new Thread(this, "Serving a Client")).start();
}

// checks how many clients are connected at a time
private int getOnlineCount() {
    int cnt=0;
    for (int i=0; i<ServiceClient.CLI.length; i++)
        if (CLI[i]!=null)
            cnt++;
    return(cnt);
}

// Sends message back to current user
private void Send(String msg) {
```

```
    synchronized (SW) {
        try {
            SW.print(msg+"\r\n");
            SW.flush();
        } catch (Exception e) {
            try {
                SO.close();
            } catch (Exception f) {}
        }
    }
}

public void run() {
    try {
        // storing the client's IP address
        CLIADDR=(SO.getInetAddress()).toString();
        // create a stream for reading via socket connection
        SR=new BufferedReader(new InputStreamReader(SO.getInputStream()));
        // create a stream for writing via socket connection
        SW=new PrintWriter (SO.getOutputStream());

        if (CO>=0) {
            Send("+OK connected");
            ReferenceTCPIPServer.Log("connected  ("+CO+") from "+CLIADDR);
            System.out.println("connected  ("+CO+") from "+CLIADDR);
            while (true) {
                String line=SR.readLine();
                if (line==null) break;
                if (line.equals("exit passcode")) {
                    ReferenceTCPIPServer.terminate(1);
                }
                                System.out.println("Recieved from Client: "+line);
```

```
                                //Send(CLIADDR);
                    // pass the command and transmit back any feedback or confirmation
                    Send( ReferenceTCPIPServer.EMBED.Send(line) );
                                //int temp;
                                //temp = line.length();
                                //for(int i = 0; i < temp;i++){
                                //ReferenceTCPIPServer.EMBED.Send(line.substring(i,i+1));}

    //System.out.println(ReferenceTCPIPServer.EMBED.Send(line));
                }
            } else {
                ReferenceTCPIPServer.Log("rejected   (+) from "+CLIADDR);
                System.out.println("rejected   (+) from "+CLIADDR);
                Send("-ERR server capacity reached");
            }
            SO.close();
        } catch (Exception e) {
            System.out.println("ERROR: (connection) "+e);
            ReferenceTCPIPServer.Log("broken con ("+CO+") from "+CLIADDR+" because of "+e);
        }
        ReferenceTCPIPServer.Log("disconnect ("+CO+") from "+CLIADDR);
        System.out.println("disconnect ("+CO+") from "+CLIADDR);
        if (CO>=0)
            CLI[CO]=null;
        // putting the embedded system to sleep if this was the last connection
        if (getOnlineCount()==0)
            try {
                ReferenceTCPIPServer.EMBED.Stop();
            } catch (Exception e) {
                System.out.println("SLEEP: (Embed) "+e);
            }
}
```

```
}

class ReportToClients implements Runnable {
    private Thread goon =null;

    public void start() {
        goon=new Thread(this);
        goon.start();
    }

    public void stop() {
        Thread temp=goon;
        goon=null;
        // try to merge the threads back if possible within 100ms
        temp.interrupt();
        try {
            temp.join(100);
        } catch (Exception e) {}
    }

    public void run() {
                BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

        while (goon==Thread.currentThread()) {
                //String sendData = "";
                String recData = "";

                //try{
                //recData = input.readLine();
                //}catch(IOException ioe){}
                recData = ReferenceTCPIPServer.EMBED.Recv();
                //System.err.println(recData);
```

```
        ServiceClient.SendAll(recData);



        //ServiceClient.SendAll(ReferenceTCPIPServer.EMBED.Recv());
                //if(tempData.indexOf("")>=-1){
                //System.out.println(tempData.substring(0,1));
                /*if(recData.length() == 5){
                switch(recData.substring(0, 1)){
                case "1": sendData = "A1 ";
                                        sendData = sendData+recData.substring(1,5);

                                        //System.out.println(tempData);
                                        ServiceClient.SendAll(sendData);
                                        break;
                case "2": sendData = "A2 ";
                                        sendData = sendData+recData.substring(1,5);
                                        //System.out.println(tempData);
                                        ServiceClient.SendAll(sendData);
                                        break;
                case "3": sendData = "A3 ";
                                        sendData = sendData+recData.substring(1,5);
                                        //System.out.println(tempData);
                                        ServiceClient.SendAll(sendData);
                                        break;
                case "D":
                case "L":
                case "S":
                default: break;
                }}*/


}
```

```
    }

}


// EMBEDDED SYSTEM INTERFACE

interface EmbedInterface {

    public void   Init()  throws Exception;  // called once when server starts
- initialize the connection

    public void   Quit()  throws Exception;  // called once when server shuts down
- shut down the connection

    public void   Start() throws Exception;  // called each time there were no clients and the
first oen came   - wake up!

    public void   Stop()  throws Exception;  // called each time when the last connected client
leaves          - go to sleep mode!

    public String Send(String command);      // feedback string is sent only to the originating
client          - called as necessary

    public String Recv();                    // feedback is sent to all connected clients
- polled every about 10ms

}
```