24/25

Outdoorsman Weather-mate Pro

By:

Michael Baj & Eric Chambers

Rensselaer Polytechnic Institute
Department
of
Electrical and Computer Systems Engineering

Class: ECSE 4790
Date: November 20, 2000

Table of Contents:

1 Abstract

The Outdoorsman Weather-Mate Pro is a companion to outdoor adventurers providing them with vital weather statistics. A compact, lightweight, and cost effective weather device can be produced with the use of a Motorola 68HC12A4 microcontroller and some electronic sensors. A prototype was constructed with an EVB board and a breadboard to show functionality. The prototype successfully gathered, stored, and processed meteorological data. It can now be integrated into a printed circuit board system for mass production.

2 Introduction

People in the outdoors have a need to know what the weather conditions are and what they will be so that they can take the necessary actions to ensure their safety while enjoying the outdoors. Instruments that exist today are heavy, bulky, and too expensive for the average adventurer. The Outdoorsman Weather-Mate Pro is designed to be a compact, lightweight, and cost effective solution to meet the meteorological needs of everyone from the picnicker to the high adventurer.

Through the use of the Motorola 68HC12A4 EVB, protoboard, electronic sensors, and LCD screens, a working prototype was constructed that can gather the environmental information, process the data and provide the user with the desired information. Electronic sensors are used to gather the environmental information such as: temperature, pressure, humidity, and cardinal orientation. This information is fed into the Motorola 68HC12A4 via analog and digital input lines and then processed, stored and fed back to the user with the use of an LCD.

3 Implementation

3.1 Hardware[1]

3.1.1 Temperature

A LM34 Precision Fahrenheit Temperature Sensor was chosen because of its direct calibration to the Fahrenheit temperature scale, low current draw (less than 90 micro amps), and linear scale factor (10mV / F). +5V was applied and the output was tied to bit 0 of Port AD for use by the microcontroller. No additional signal conditioning was deemed necessary for proper operation.

3.1.2 Pressure

A Motorola MPX2200A 200kPa pressure sensor was chosen. It is temperature compensated, available in an absolute configuration, and the strain gauge style is easily connected to a microcontroller. The sensor requires bipolar power therefore ±12V was used. An amplifier was necessary because the full-scale output is only 40mV. An AMP02 Instrumentation Amplifier was chosen for simple gain setting and bipolar input. The proper gain was chosen by the following equation

$$\frac{200kPa}{40mV} \times \frac{0.2953inHg}{kPa} \times \frac{5V}{37inHg} \approx 200$$

The amplifier has the following gain equation:

$$G = \frac{50k\Omega}{R_G} + 1$$

R_G was set to 250 Ohms to produce a gain of 201. 37 inches Hg was used because the average range for pressure is between 25-35 inches Hg. The output of the amplifier was connected to bit 2 of Port AD for use by the microcontroller.

[1] See Appendix C for Circuit Diagram.

3.1.3 Humidity

A HIH-3605 humidity sensor was chosen for its linear output. The output equation is:

$$V_{OUT} = V_{SUPPLY} \times [0.0062(\%RH) + 0.16]$$

The sensor is capable of operation with 4-9VDC, therefore 5V was used and the output was tied to bit 3 of Port AD for use by the microcontroller.

3.1.4 Compass

A Dinsmore 1490 digital sensor used for the digital compass. Whichever pin is opposite north when the sensor is held upright goes high. Therefore the sensor can be configured for any direction. A mark was placed on the sensor to designate north to provide a reference for the pins. Each pin was connected to one bit of the lower nibble of Port T. No further signal conditioning was needed because the signal generated is digital.

3.1.5 Keypad

A 4x4 Keypad was interfaced to the microcontroller with the use of Port J. The 4 row lines are connected to the lower nibble of Port J. The 4 column lines are connected to the upper nibble of Port J. They were connected in sequential order.

3.1.6 LCD Display

A Hitachi HD44780 LCD screen was interfaced with the microcontroller with the use of Ports G and H.

3.2 Software[2]

The software for the Outdoorsman Weather-mate Pro uses the functionality of the Motorola 68HC12A4, Introl C code and the real timer interrupt to produce the finished product.

3.2.1 main()

main() begins by fist initializing the hardware and the variables. It is necessary to turn on the timer function and set it for use in general I/O. Port T is set for input to receive data from the digital compass. 0x80 is sent to the control register _H12ADTCTL2 to activate the Analog to Digital converter (ATD). A for-loop was used to provide the necessary 100 μs for the initialization to complete.

The RTI was initialized and set up to trigger every 65.536 ms, providing a 1 second counter with every 15 cycles. The RTI interrupt service routine, RTIInt(), handles all clock functions needed and the setting of necessary timing flags.

The last devices to be initialized are the LCD and the keypad.
All of the elements in the arrays were initialized to 0 for memory integrity.

main() enters a continuous while loop that monitors flags and calls the associated function based on the flag status. If a key on the keypad was pressed, then a flag was set. If this flag is set, then it needs to be reset so that another key can be pressed. Each second information is gathered from the ATD sensors by calling pollATD(0) and the display is refreshed with the current information. This is accomplished by calling buildData(). Upon completion of these two functions, the sec flag is reset.

Every minute, a weather prediction is made based on the currently stored information. Again, we gather information from the ATD. This time however, the highs and lows are stored. Once all of the functions have returned, the min flag is reset.

Each hour the highs and lows are stored for that hour to be used in further analysis. The hr flag is then reset.

---

[2] See Appendix B for Introl C Source Code.

3.2.2 RTIInt()

RTIInt is used to simulate a clock. This interrupt was used solely for keeping time. When RTIInt is called, it first clears its flag. This way it will again be able to be called. A separate counter, RTIcount is used to keep track of how many times RTIInt has been called. When it has been called 15 times, approximately one second, we increment the second counter and set the seconds flag. Once the second counter is greater than or equal to 60, we increment the minute counter, set the second counter back to 0, and set the min flag. When the minute counter is greater than or equal to 60, the hour counter is incremented as well as another counter that keeps track of the total number of hours elapsed. The minute counter is set back to 0 and the hr flag is set to true. If and when the hour counter is greater than or equal to 24 hours, an hour overflow flag is set and this signifies that a day has passed. This flag is used in the slope function to determine how many data points to take the slope. The hour counter is also set back to 0. After all of the cases are checked, then the RTIcount variable is again set back to 0 and the process happens all over again.

3.2.3 slope()

Slope is a simple function that calculates the slope for the change in pressure. If the device has not been running for more than 24 hours, it will calculate the slope for only those hours passed, otherwise, it will calculate the slope over a 24 hour period. The calculation for the slope is your typical slope calculation. This gives us an accurate prediction of the pressure change.

3.2.4 prediction()

Prediction is used to predict the weather patterns. This is accomplished by checking the values of the sensors and settings the prediction flags based on those values.

pSlope varies from 0-5.
0: uninitialized
1: A rapid decrease in pressure (decreased by 0.5"/hr)
2: Steady decrease in pressure (decreased by 0.1"/hr)
3: A constant pressure (within +/- 0.1"/hr)
4: Steady increase in pressure (increased by 0.1"/hr)
5: Rapid increase in pressure (increased by 0.5"/hr)

hFlag varies from 0-2.
0: uninitialized
1: Sticky (humidity > 80%)
2: Dry (humidity < 80%)

tFlag varies from 0-2.
0: uninitialized
1: Very Hot! (temperature > 90)
2: Warm (temperature > 32 < 90)
3: Cold (temperature < 32)

3.2.5 Temperature(int time)

Temperature takes the value from _H12ADR0H and stores this information into tempData[0], the current temperature, after some calculations have been performed on the number. In order to get the temperature in degrees Fahrenheit from the sensor with a decimal shift of 1 to the right, we need to perform the following calculation.

$$tempData[0] = \frac{((\_H12ADR0H \times 100) + (\_H12ADR0H\%51))}{51}$$

Every minute, the temperature function performs a check to determine what the highs and lows are with the last 24 hours. If the current temperature is lower than the stored lowest temperature, then it is stored in tempData[1]. If the current temperature is larger than the stored highest temperature, then it is stored in tempData[2]. On the hour, temperature stores the current, high, and the low temperatures. The variable 'time' which is passed in determines which operation is performed and when.

3.2.6 Humidity(int time)

The register _H12ADR3H holds the digital representation of the voltage received by the humidity sensor. In order to increase accuracy and still use integers, the following manipulation to humidity output equation (see 3.1.3) is made.

$$RH = \frac{(V_{OUT} \times 10) \times 323 - 2581}{100} \qquad V_{SUPPLY} = 5$$

The digital reading stored in _H12ADR3H is converted to a voltage by

$$V_O = \frac{((\_H12ADR3H \times 10) + (\_H12ADR3H\%51))}{51}$$

The voltage is then converted to the temperature and stored in humiData[0] for later use.

$$humiData[0] = \frac{V_O \times 323 - 2581}{100}$$

Every minute, the humidity function performs a check to determine what the highs and lows within the last 24 hours. If the current humidity is lower than the stored lowest humidity, then it is stored in humiData[1]. If the current humidity is larger than the stored highest humidity, then it is stored in humiData[2]. On the hour, humidity stores the current, high, and the low humidity. The variable 'time' which is passed in determines which operation is performed and when.

3.2.7 Pressure(int time)

Pressure takes the value from _H12ADR2H and stores this information into presData[0], the current pressure, after some calculations have been performed on the number. In order to get the pressure in inches from the sensor, we need to perform the following calculation.

$$presData[0] = \frac{((\_H12ADR2H \times 6.2) + (\_H12ADR2H\%51))}{51}$$

Every minute, the pressure function performs a check to determine what the highs and lows within the last 24 hours. If the current pressure is lower than the stored lowest pressure, then it is stored in presData[1]. If the current pressure is larger than the stored highest pressure, then it is stored in presData[2]. On the hour, pressure stores the current, high, and the low pressure. The variable 'time' which is passed in determines which operation is performed and when.

3.2.8 Compass()

Compass determines what direction the Dinsmore 1490 digital compass is pointing in. This is accomplished by checking the low byte of Port T to see which bit is set as each bit corresponds to a direction.

Bit	Direction
0	North
1	South
2	East
3	West

If the compass is pointing between two adjacent directions, then both bits are set high and the corresponding output would be given (i.e. NE). Should the bits equal something other than what has been defined, then the compass reading is set to ' ' (blank).

3.2.9 pollATD(int time)

When pollATD is called, the Motorola 68HC12A4 polls the CCF1 bit and then averages the 4 conversion results stored in ADR0-ADR3. This causes the ATD to do 1 conversions on Channels 0-3 and then stop. The conversions are stored in result registers ADR0-ADR3. After this is completed, it calls each of the sensor functions, passing the variable time to each of them, and then returns.

3.2.10 buildData()

This function creates the strings that will be displayed to the LCD. The fist thing that is done is to create a unique value that is a combination of all the prediction flags. Multiplying each flag by a different factor of ten and then adding them together obtain this. This new variable, stats, gives us every combination of weather patterns that can be observed by the Outdoorsman Weather-mate Pro.

buildData() is simply a number of switch statements. A specific switch statement is chosen based on what key has been pressed on the keypad. That value is then stored in memory and is not changed until the keypad is pressed again.
The options and displays that are available to the user are:

Keypad option 1:
```
TMP  HMD PRS DIR
69 F 29% 31" NW
```

Keypad option 2:
```
Constant  Avg P
Warm  Dry
```

Keypad option 3:
```
I see skies of blue
and clouds of white
```

Keypad option 4:
```
Temperature
H:71 F   L:67 F
```

Keypad option 5:
```
Pressure
H:31"    L:28"
```

Keypad option 6:
```
Humidity
H:32%        L:25%
```

Keypad option #:

This key is used for debugging purposed only. It will change the current prediction by changing the flags which will set the 'stats' variable. When pressed, it will prompt you with a number of options each individually used to change the prediction state.

```
Enter Value
pSlope (1-5):

Enter Value
pFlag (1-3):

Enter Value
tFlag (1-3):

Enter Value
hFlag (1-2):
```

If a value is input other than the suggested value, then the default message will be displayed for options 2 and 3.
For instance, if the values 5, 4, 1, 2 were entered, the display would then show:

Keypad option 2:

```
Rapid  ERR
Warm  Sticky
```

Keypad option 3:

```
Predictions
not yet set
```

Keypad option A:

This key is used for debugging purposed only. It will store "dummy" data in the Weather-mate Pro in order to display fixed weather states. This option changes the stored highs and lows and the resulting prediction is "Expect Snow". Consequently, the message will be changed in a minute when the prediction function is called again.

Keypad option B:

This key is used for debugging purposed only. It will store "dummy" data in the Weather-mate Pro in order to display fixed weather states. This option changes the stored highs and lows and the resulting prediction is "Clear skies are here again". Consequently, the message will be changed in a minute when the prediction function is called again.

Keypad option D:

Because we change the states of the variables in order to create test scenarios, we need a means of returning the values back to their previous state. Pressing 'D' on the keypad will do just this.

3.2.11 wr_scrn(char *buffer, char flag)

wr_scrn, or write screen, takes a character pointer and displays this message to the LCD. The LCD is split up to a top and bottom. Besides the character pointer being passed into the function, another value is passed in as well. This determines to what half of the LCD, the information is going to be displayed to. A '1' will display the characters to the top row of the LCD and by passing a '0' to the function; it will display the characters to the bottom of the LCD.

3.2.12 wr_char(char data)

wr_char, or write character, is used to display a single character to the LCD. It first clears the LCD screen and then displays the character. This was not used in our final code for the project, but remained for potential future expansion.

4 Budget

	Proposed	Actual
Motorola 68HC12A4	0	0
Sensors		
LM74 Temperature probe	4.00	1.83
Dinsmore 1490 electronic compass	50.00	14.00
HIH-3605 humidity sensor	40.00	33.02
Motorola MPX2200A 200kPa pressure sensor	15.00	16.00
AMP02 Instrumentation Amplifier	TBD	10.73
Connector Sockets	0	3.79
Miscellaneous	5.00	51.87
Total	$114.00	$131.24

5 Results

The final product worked correctly. The temperature, humidity, pressure, and digital compass were successfully integrated with the Motorola 68HC12A4 microcontroller. Code was written and loaded into the microcontroller, with the use of the Introl C compiler, to gather and store the sensor data, make predictions based on the sensor data, and provide the user with the information via the LCD screen.

The project was constructed on an EVB board and a large breadboard. This proved functionality of the design. By purchasing the sensors and microcontroller on a large scale and generating a printed circuit board (PCB) layout for the project, the design could be mass-produced at a small per unit cost. The PCB would allow for a compact and lightweight package, thus fulfilling the original design goal.

TESTING FOR ACCURACY? (CALIBRATION)

HOW DID FINAL COMPARE TO PROPOSED?

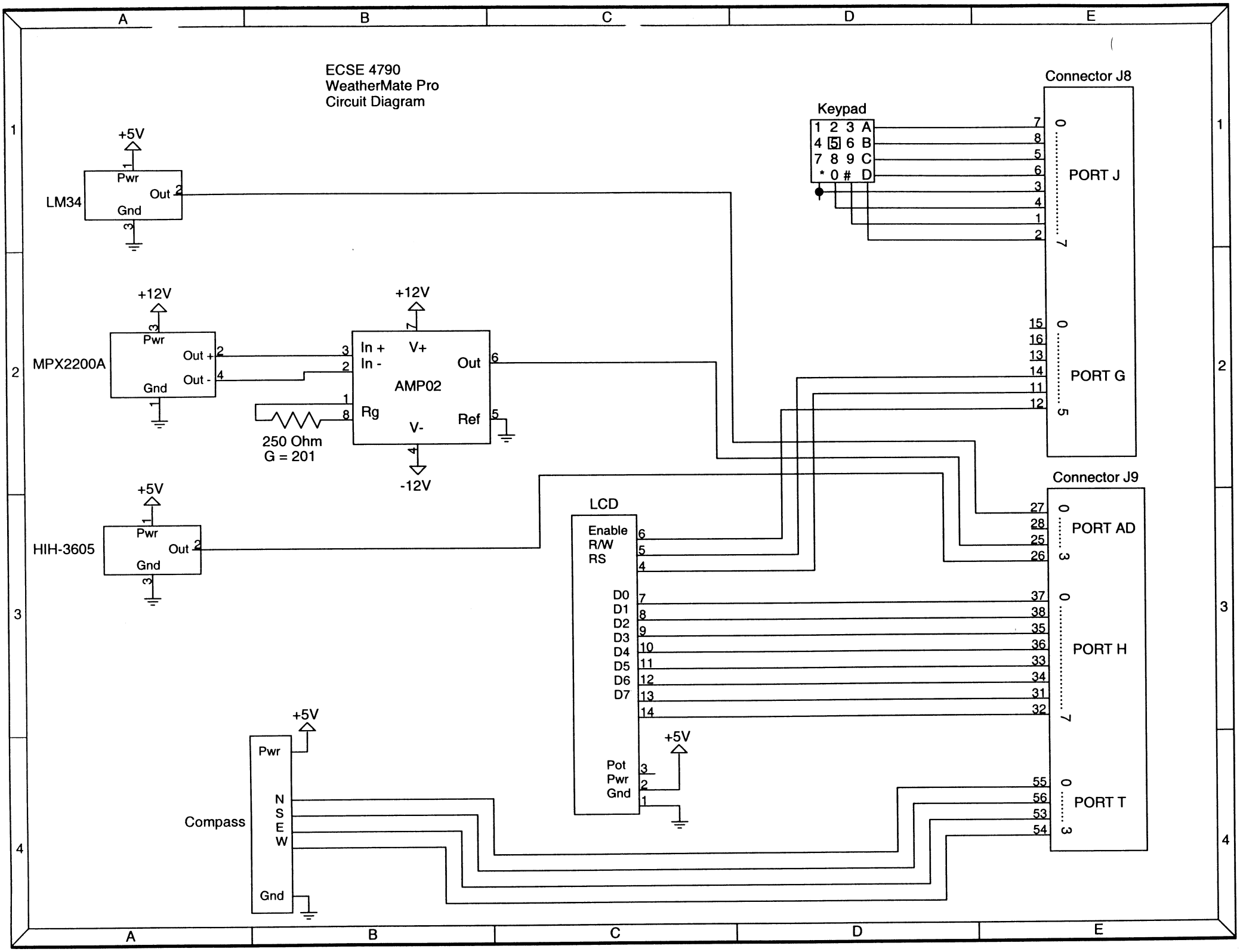
6 References

1. ECSE-4790 Microprocessor Systems Lab 7: Introduction to the 68HC21 – The Magic 8 Ball, Pg 7
2. Rosenberg, Lee, "Motorola 68HC12 User's Manual", Electrical and Computer Systems Engineering, Rensselaer Polytechnic Institute, Rev. 1.1, August 2000

7 Bibliography

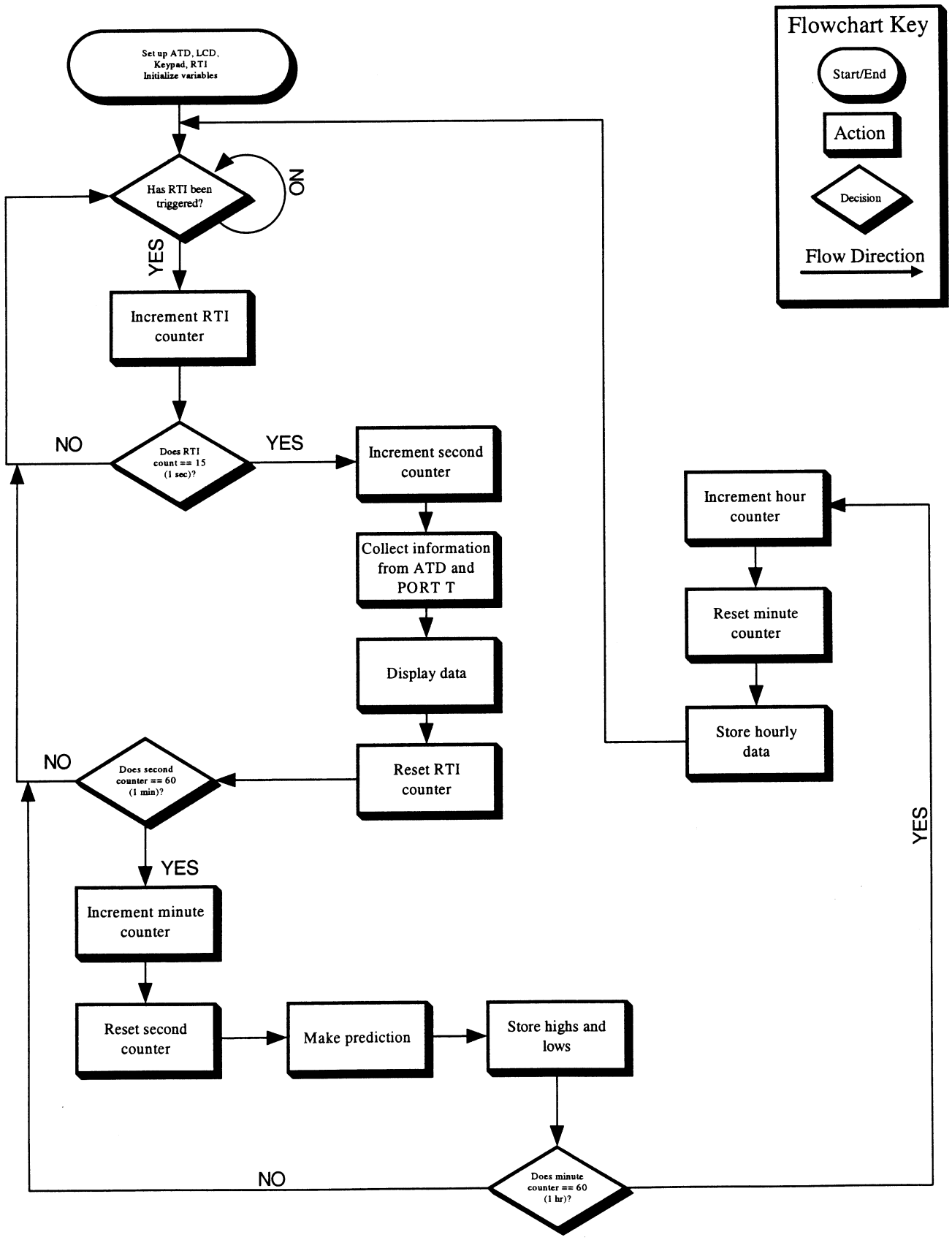
1. Cady, Fredrick M, & Sibigtroth, James M., "Software and Hardware Engineering; Motorola MC68HC12", Oxford University Press Inc., 2000

Appendix A: Schematic

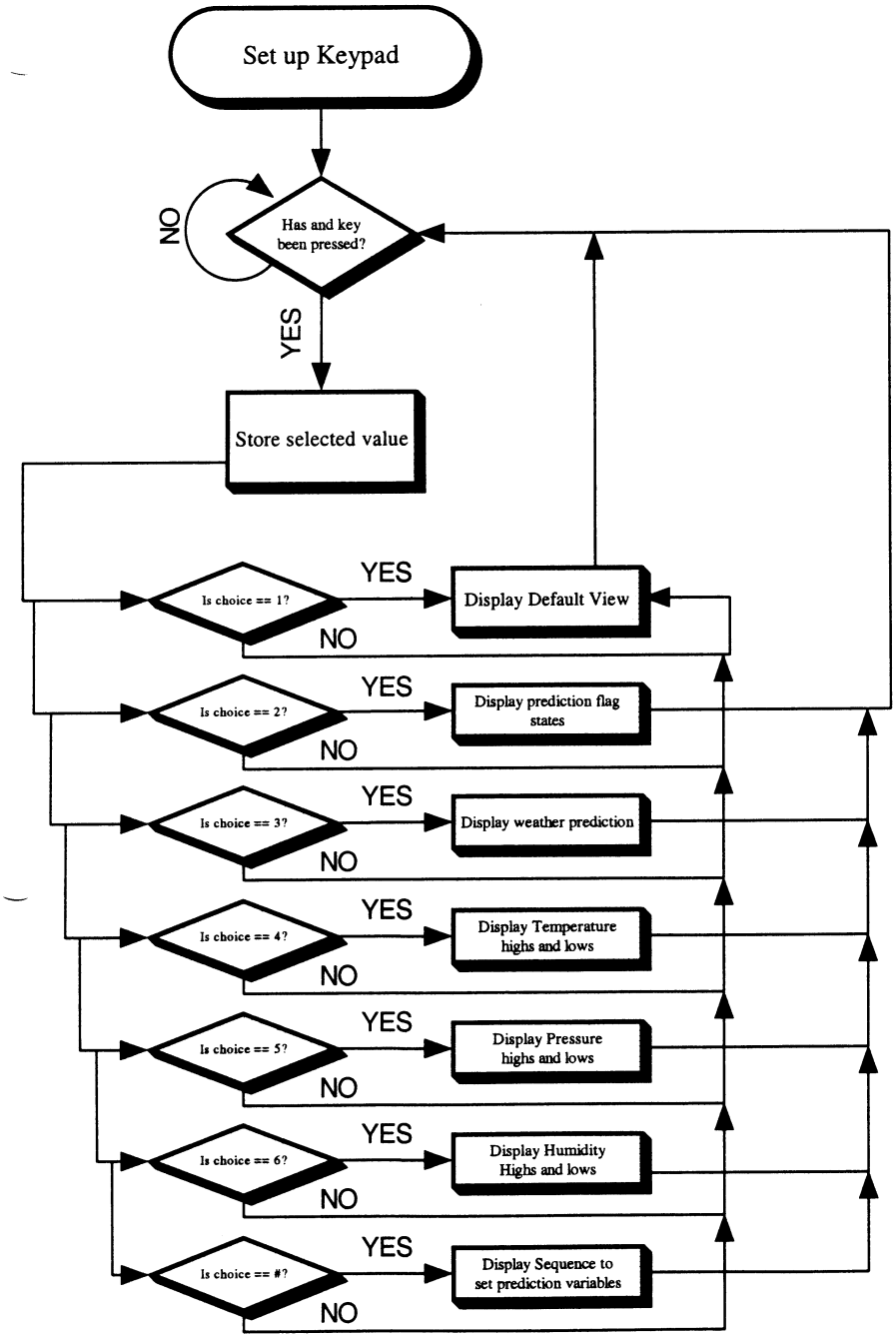

ECSE 4790
WeatherMate Pro
Circuit Diagram
LM34
+5V
Pwr
Out
Gnd
MPX2200A
+12V
Pwr
Out +
Out -
Gnd
AMP02
In +
In -
V+
Out
Rg
Ref
V-
250 Ohm
G = 201
-12V
HIH-3605
+5V
Pwr
Out
Gnd
Compass
+5V
Pwr
N
S
E
W
Gnd
LCD
Enable
R/W
RS
D0
D1
D2
D3
D4
D5
D6
D7
Pot
Pwr
Gnd
+5V
Keypad
1 2 3 A
4 5 6 B
7 8 9 C
* 0 # D
Connector J8
PORT J
PORT G
Connector J9
PORT AD
PORT H
PORT T

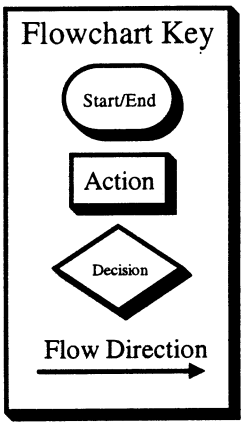
Appendix B: Flow Charts

Outdoorsman Weathermate Pro

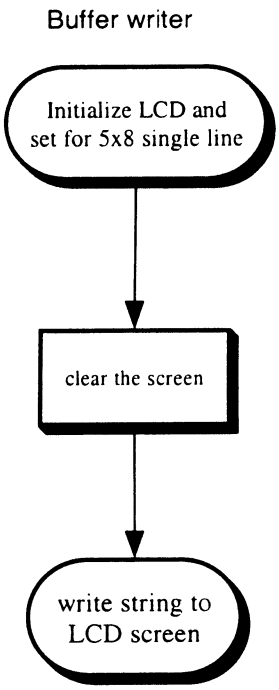


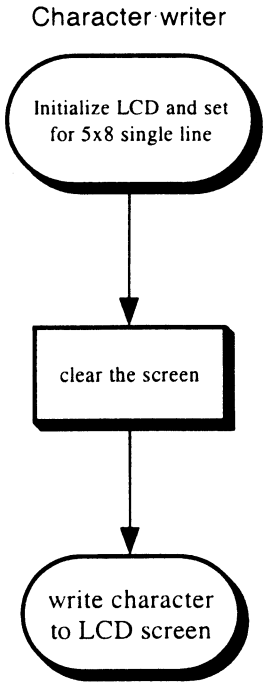
Key Pad

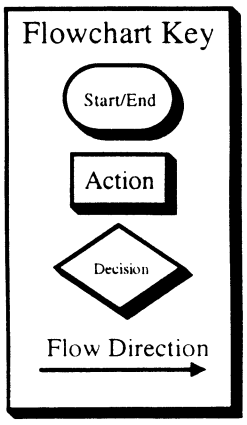

Set up Keypad
Has and key been pressed?
NO
YES
Store selected value
Is choice == 1?
YES
NO
Display Default View
Is choice == 2?
YES
NO
Display prediction flag states
Is choice == 3?
YES
NO
Display weather prediction
Is choice == 4?
YES
NO
Display Temperature highs and lows
Is choice == 5?
YES
NO
Display Pressure highs and lows
Is choice == 6?
YES
NO
Display Humidity Highs and lows
Is choice == #?
YES
NO
Display Sequence to set prediction variables


Flowchart Key
Start/End
Action
Decision
Flow Direction

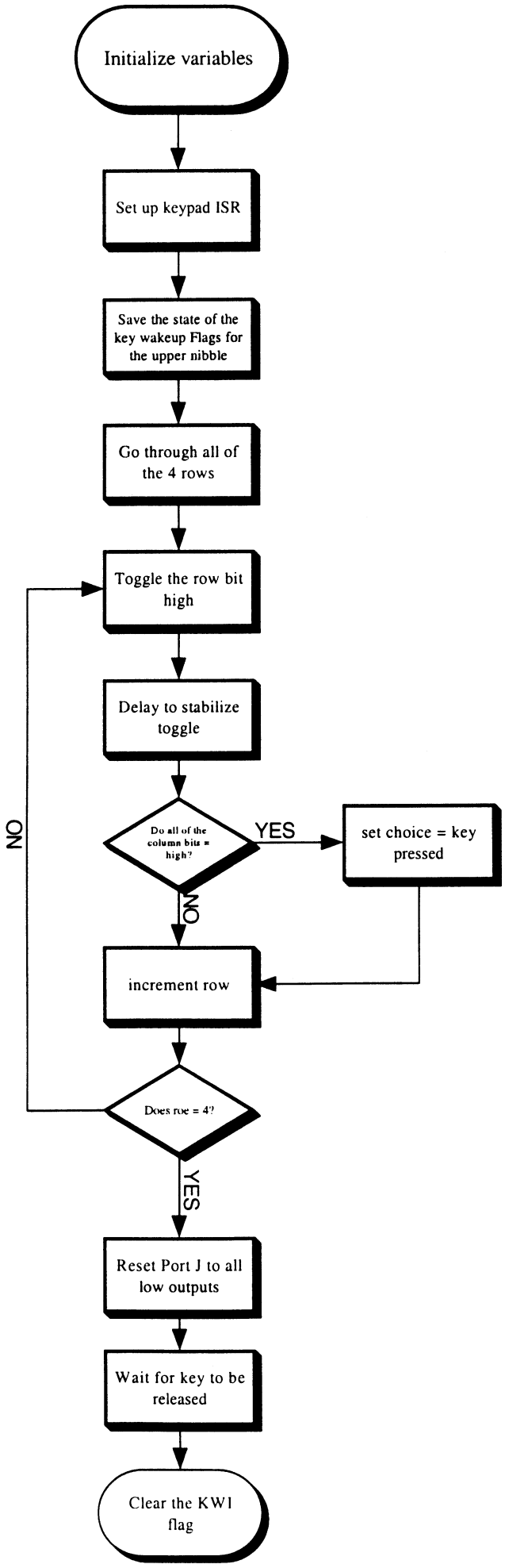
LCD Display functions

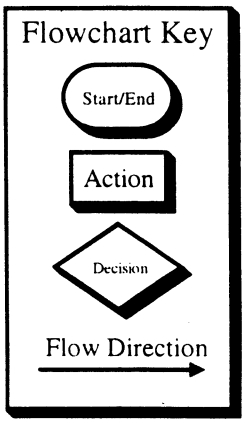






"Keypad.h"


Initialize variables
Set up keypad ISR
Save the state of the key wakeup Flags for the upper nibble
Go through all of the 4 rows
Toggle the row bit high
Delay to stabilize toggle
Do all of the column bits = high?
YES
set choice = key pressed
NO
increment row
Does row = 4?
NO
YES
Reset Port J to all low outputs
Wait for key to be released
Clear the KWI flag


Flowchart Key
Start/End
Action
Decision
Flow Direction

Appendix C: Source Code

```
////////////////////////////////////////////////////////////////////////////
//      ECSE 4790
//      Final Project:  Outdoorsman Weathermate Pro
//      Group Members:          Eric Chambers, Michael Baj
//      Date:                   10/23/00
//      File:                   Weathermate.c
//      Description:
//              The Outdoorsman Weathermate Pro brings in environmental data on the
//      ATD PORT.
//      PAD0:  Temperature
//      PAD2:  Humidity
//      PAD3:  Pressure
//      PORTT:  Cardinal Directions
// This data is stored.  Every minute, the highs and lows are stored and
//      again every hour.  Prediction information is calculated on the stored
//      data.
//
//      Arguments:
//              n/a
//
//      Output:
//              All things are sent to LCD;  A few display options are
//      available to the user by pressing a key on the keypad
//      1:      Default View  This provides current weather information to the user
//      2:      Display states of prediction flags
//      3:      Weather prediction
//      4:      Temperature highs and lows
//      5:  Pressure higs and lows
//      6:      Humidity highs and lows
//      #:      Manually set the prediction flags for testing purposes
//      Any other key not defined will return the user back to the default
//      view, which is the same if you press key 1.
//
//      Notes:
//              Weather predicition information is gathered every minute.
//      Therefore the user must wait said time before being able to read
// any of the prediction information.
//              Should the user enter invalid values for the falgs, ERR will
//      be displayed on the screen for option 2 and "Predictions not yet
// set" for option 3.
//              For the fist minute, the lows will be a very large value.  This
//      is such that the code will be able to evaluate a low.
//
//      This code was written and tested using Introl C 4.0
////////////////////////////////////////////////////////////////////////////

// includes
#include <hc812a4.h>
#include <introl.h>
#include <dbug12.h>
#include <keypad.h>
#include <Lcd12.h>
#include <string.h>

//Global Variables
#define highLow 75              // size of data arrays

int RTIcount    = 0;    // Number of times RTI was called
int numHours    = 0;    // Total number of hours passed
int tFlag       = 0;    // Current Temperature State
int hFlag       = 0;    // Current Humidity State
int pFlag       = 0;    // Current Pressure State
int pSlope      = 0;    // Current Change in Pressure
int sec         = 0;    // Second passed flag
int min         = 0;    // Minute passed flag
int hr          = 0;    // hour passed flag
int secCounter  = 0;    // Number of seconds passed
int minCounter  = 0;    // Number of minuted passed
int hrCounter   = 0;    // Number of hours passed
int hourOverflow = 0;   // Flag which denotes hrCounter > 24
char direction1;        // N & S cardinal Directions
char direction2;        // E & W cardinal Directions
int tempData[highLow]; // 0 + 3n = current
int presData[highLow]; // 1 + 3n = low
int humiData[highLow]; // 2 + 3n = high
int presPred[24][2];    // stored pressure data per hour
char predLCD[16] = "            ";// Used to display info to LCD
char upprLCD[16] = "TMP  HMD PRS DIR";      // Used to display info to LCD
char dataLCD[16] = "            ";// Used to display info to LCD

// interrupt service routine declarations
__mod2__ void RTIInt();

// Function Prototypes
int slope();
void prediction();
void Temperature(int time);
void Humidity(int time);
void Pressure(int time);
void Compass();
void pollATD(int time);
void buildData();
void wr_scrn(char *buffer, char flag);      // Writes ASCII characters to LCD screen
void wr_char(char data);                    // Writes one char to the LCD screen


void __main()
{
        int i;

        _H12TIOS=0x00;                      // Turns of timer function and sets for general I/O
        _H12DDRTT=0x00;             // Set up port T Digital I/O for input

        // Initialize the ADC for use
        _H12ADTCTL2=0x80;   // power on the ADC
        for(i=0; i<100; i++);       // need to wait at least 100 usec
        _H12ADTCTL3=0x00;   // don't stop at breakpoints
        _H12ADTCTL4=0x43;   // set ATD clock speed to 125 KHz

        DB12->SetUserVector(RTI, RTIInt); // Set up interrupt vectors
        _H12RTICTL=0x87;            //  Set up RTI to trigger every 65.536 ms
```

```
	OpenXLCD(0x3F);		// Initialize LCD and set for 5x8 double line
	KeypadInit();		// Initialize and Activate the Keypad
	// "choice" is set by the keypad depression

	// Initialize Highs and lows
	for(i=0; i<highLow+1; i++)
	{
		tempData[i] = 0;
		presData[i] = 0;
		humiData[i] = 0;
	}

	humiData[1] = 99;		// Set to max value so we can calculate lows
	presData[1] = 99;		// Set to max value so we can calculate lows
	tempData[1] = 99;		// Set to max value so we can calculate lows

	// Initialize Prediction Data
	for(i=0; i<24; i++)
	{
		presPred[i][0] = 0;
		presPred[i][1] = 0;
	}

	while(1)
	{
		if(keypressFlag);		// Wait for key press
		keypressFlag = 0;		// Reset the Flag
		if(sec)
		{
			pollATD(0);
			buildData();
			sec = 0;		// Reset flag
		}
		if(min)
		{
			// Update the Highs and Lows
			prediction();
			pollATD(1);
			min = 0;		// Reset flag
		}
		if(hr)
		{
			pollATD(2);
			hr = 0;			// Reset flag
		}
	}
}


// Store current temperature
void Temperature(int time)
{
	tempData[0] = ((_H12ADR0H * 100) + (_H12ADR0H % 51))/51;  // temp in degrees F
	if(time == 1)				// Every Minute
	{
		if(tempData[0] < tempData[1]) // Checks for low
```

```
		tempData[1] = tempData[0];

		if(tempData[0] > tempData[2]) // Checks for high
		tempData[2] = tempData[0];
	}

	else if(time == 2)		// Every Hour
	{
		tempData[hrCounter+3] = tempData[0]; // Stores current
		tempData[hrCounter+4] = tempData[1]; // Stores low
		tempData[hrCounter+5] = tempData[2]; // Stores high
	}
}

// Store current humity
void Humidity(int time)
{
	int Vo;
	Vo = ((_H12ADR3H * 10) + (_H12ADR3H % 51))/51; // Voltage * 10
	humiData[0] = (Vo * 323 - 2581)/100;	// relative Humidity in %

	if(time == 1)		// Every Minute
	{
		if(humiData[0] < humiData[1]) // Checks for low
		humiData[1] = humiData[0];

		if(humiData[0] > humiData[2]) // Checks for high
		humiData[2] = humiData[0];
	}
	else if(time == 2)		// Every Hour
	{
		humiData[hrCounter+3] = humiData[0]; // Stores current
		humiData[hrCounter+4] = humiData[1]; // Stores low
		humiData[hrCounter+5] = humiData[2]; // Stores high
	}
}

// Store current pressure
void Pressure(int time)
{
	presData[0] = ((_H12ADR2H * 6.2) + (_H12ADR2H % 51))/51; // Pressure in Inches
	if(time == 1)		// Every Minute
	{
		if(presData[0] < presData[1])  // Checks for low
		presData[1] = presData[0];

		if(presData[0] > presData[2])  // Checks for high
		presData[2] = presData[0];
	}
	else if(time == 2)		// Every Hour
	{
		presData[hrCounter+3] = presData[0]; // Stores current
		presData[hrCounter+4] = presData[1]; // Stores low
		presData[hrCounter+5] = presData[2]; // Stores high
	}
	presPred[hrCounter][0] = presData[0];	// Current Pressure
```

```
	presPred[hrCounter][1] = numHours;		// Hour Index
}

// Predict weather pattern
void prediction()
{
	int pS;

	pS = slope();

	if(humiData[0] > 80)
		hFlag = 2;	// Sticky
	else
		hFlag = 1;	// Dry

	if(tempData[0] > 90)
		tFlag = 3;	// Very Hot!!!
	else if(tempData[0] > 32)
		tFlag = 2;	// Warm
	else
		tFlag = 1;	// Cold

	if(presData[0] > 31)
		pFlag = 3;	// High Pressure
	else if(presData[0] > 30)
		pFlag = 1;	// Average Pressure
	else
		pFlag = 2;	// Low Pressure

	if(pS > 5)		// rapidly increasing (increased by 0.5"/hr)
		pSlope = 5;
	else if(pS > 1)  // steady increase (increased by 0.1"/hr)
		pSlope = 4;
	else if(pS > -1) // constant (within ±0.1"/hr)
		pSlope = 3;
	else if(pS > -5) // steady decrease (decreased by 0.1"/hr)
		pSlope = 2;
	else			// rapid decrease (decreased by 0.5"/hr)
		pSlope = 1;
}

// Store current direction
void Compass()
{
	if(!(_H12PORTT & 0x01))
		direction1 = 'N';
	else if(!(_H12PORTT & 0x02))
		direction1 = 'S';
	else
		direction1 = ' ';

	if(!(_H12PORTT & 0x04))
		direction2 = 'E';
	else if(!(_H12PORTT & 0x08))
		direction2 = 'W';
	else
		direction2 = ' ';
}

// Writes one string to the screen
void wr_scrn(char *buffer, char flag)
{
	// when flag is set to 0, will write to bottom row, else writes to top row
	if(flag)
		WriteCmdXLCD(0x01);  // clear LCD screen
	else
		WriteCmdXLCD(0xC0); // Moves Cursor to second line ie. sets the DDRAM to
0x40
	WriteBuffer(buffer);	// write data string to LCD screen
}

// Writes a single character to the screen
void wr_char(char data)
{
	WriteCmdXLCD(0x01);		// clear LCD screen
	WriteDataXLCD(data);  // write character to LCD screen
}

// Gets information from the A2D converters
void pollATD(int time)
{
	// This module polls the CCF1 bit and then averages the 4 conversion
	// results stored in ADR0-ADR3.

	// This causes the ATD to do 1 conversions on Channels 0-3
	// and then stop.  The conversions are stored in result registers
	// ADR0-ADR3.

	_H12ADTCTL5 = 0x10;		// set ATD to single scan of channels 0-3
	while(!(_H12ADTSTAT & 0x01));
	Temperature(time);
	Humidity(time);
	Pressure(time);
	Compass();
}

void buildData()
{
	int i, stats;

	stats = (pSlope*1000) + (pFlag*100) + (tFlag*10) + hFlag;
	DB12->strcpy(predLCD, "");
	DB12->strcpy(dataLCD, "");

	switch (choice)
	{
	case '2':				// Display States of Flags
		if(pSlope == 5)		// Rapid Increase in Pressure
			strcat(predLCD, "Rapid Up  ");
		else if(pSlope == 4)	// Steady Increase in Pressure
			strcat(predLCD, "Steady Up ");
```

```
        else if(pSlope == 3)        // No change in Pressure
                strcat(predLCD, "Constant  ");
        else if(pSlope == 2)        // Steady Decrease in Pressure
                strcat(predLCD, "Steady Dn ");
        else if(pSlope == 1)        // Rapid Decrease in Pressure
                strcat(predLCD, "Rapid Dn  ");
        else                                        // Pressure Flag has not been set
                strcat(predLCD, "ERR  ");


        if(pFlag == 3)                      // High Pressure
                strcat(predLCD, "High P");
        else if(pFlag == 2)                 // Low Pressure
                strcat(predLCD, "Low P");
        else if(pFlag == 1)                 // Average Pressure
                strcat(predLCD, "Avg P");
        else                                        // Pressure Flag has not been set
                strcat(predLCD, "ERR");


        if(tFlag == 3)                      // Hot
                strcat(dataLCD, "Hot  ");
        else if(tFlag == 2)                 // Warm
                strcat(dataLCD, "Warm  ");
        else if(tFlag == 1)                 // Cold
                strcat(dataLCD, "Cold  ");
        else                                        // Temperature Flag has not been set
                strcat(dataLCD, "ERR  ");


        if(hFlag == 2)                      // Sticky
                strcat(dataLCD, "Sticky");
        else if(hFlag == 1)                 // Dry
                strcat(dataLCD, "Dry");
        else                                        // Humidity Flag has not been set
                strcat(dataLCD, "ERR");

        wr_scrn(&predLCD, 1);  // Writes to the top row of the LCD
        wr_scrn(&dataLCD, 0);  // Writes to the bottom row of the LCD
        break;

case '3':                                           // Display Weather Predictions
        if( (stats == 5132) || (stats == 5131) || (stats == 5122) || (stats == 5121) ||
                (stats == 4132) || (stats == 4131) || (stats == 4122) || (stats == 4121) ||
                (stats == 3221) )
                strcat(predLCD, "Chance of Rain");

        else if( (stats == 3211) )
                strcat(predLCD, "Chance of Snow");

        else if( (stats == 3312) || (stats == 3311) || (stats == 3112) || (stats == 3111) )
                strcat(predLCD, "Clear and Cold");

        else if( (stats == 5331) || (stats == 4331) ){
                strcat(predLCD, "Clear skies are");
                strcat(dataLCD, "here again");
        }

        else if( (stats == 4221) ){
                strcat(predLCD, "Clearing, Fair");
                strcat(dataLCD, "for several days");
        }

        else if( (stats == 5322) || (stats == 4322) )
                strcat(predLCD, "Cloudy and Fair");

        else if( (stats == 1212) ){
                strcat(predLCD, "Continued");
                strcat(dataLCD, "Blizzard");
        }

        else if( (stats == 3321) || (stats == 3121) )
                strcat(predLCD, "Continued Fair");

        else if( (stats == 2222) )
                strcat(predLCD, "Continued Rain");

        else if( (stats == 2211) || (stats == 1211) )
                strcat(predLCD, "Continued Snow");

        else if( (stats == 2232) || (stats == 2231) || (stats == 2221) || (stats == 1232) ||
                (stats == 1231) || (stats == 1221) )
                strcat(predLCD, "Continued Storm");

        else if( (stats == 2132) || (stats == 2131) || (stats == 2121) || (stats == 1132) ||
                (stats == 1131) || (stats == 1121) || (stats == 2212) || (stats == 5232) ||
                (stats == 5222) || (stats == 4232) || (stats == 4222) || (stats == 2332) ||
                (stats == 2322) || (stats == 1332) || (stats == 1322) || (stats == 2331) ||
                (stats == 1331) || (stats == 1321) || (stats == 5231) || (stats == 5221) ||
                (stats == 4231) )
                strcat(predLCD, "Expect Rain");

        else if( (stats == 5212) || (stats == 4212) || (stats == 2312) || (stats == 2311) ||
                (stats == 2112) || (stats == 1312) || (stats == 1311) || (stats == 1112) ||
                (stats == 5211) || (stats == 4211) )
                strcat(predLCD, "Expect Snow");

        else if( (stats == 5111) || (stats == 4111) ){
                strcat(predLCD, "Fair,");
                strcat(dataLCD, "Cold Day");
        }

        else if( (stats == 5112) || (stats == 4112) )
                strcat(predLCD, "Fair, Expect Rain");

        else if( (stats == 3231) || (stats == 3222) ){
                strcat(predLCD, "High Chance");
                strcat(dataLCD, "of Rain");
        }

        else if( (stats == 3212) )
                strcat(predLCD, "High Chance of Snow");
```

```
else if( (stats == 5332) || (stats == 4332) || (stats == 3332) || (stats == 3132) )
        strcat(predLCD, "Hot & Humid");

else if( (stats == 3331) || (stats == 3131) )
        strcat(predLCD, "Hot and Fair");

else if( (stats == 5321) || (stats == 4321) ){
        strcat(predLCD, "I see skies of blue");
        strcat(dataLCD, "and clouds of white");
}

else if( (stats == 5312) || (stats == 5311) || (stats == 4312) || (stats == 4311) ){
        strcat(predLCD, "Lovely Winter");
        strcat(dataLCD, "Day");
}

else if( (stats == 2111) || (stats == 1111) )
        strcat(predLCD, "Possibly Snow");

else if( (stats == 1222) ){
        strcat(predLCD, "Rain & high winds");
        strcat(dataLCD, "Expect Storm");
}

else if( (stats == 2321) ){
        strcat(predLCD, "Slow Rise in Temp");
        strcat(dataLCD, "Fair for few days");
}

else if( (stats == 3322) || (stats == 3122) )
        strcat(predLCD, "Warm & Humid");

else if( (stats == 3232) )
        strcat(predLCD, "Warm & Rainy");

else if( (stats == 2122) || (stats == 1122) ){
        strcat(predLCD, "Wind Increasing");
        strcat(dataLCD, "Expect Rain");
}

else{
        strcat(predLCD, "Predictions\0");
        strcat(dataLCD, "not yet set\0");
}

wr_scrn(&predLCD, 1);  // Writes to the top row of the LCD
wr_scrn(&dataLCD, 0);  // Writes to the bottom row of the LCD
break;

case '4':                                   // Display Highs and Lows for
temperature
        strcat(predLCD, "Temperature");
        dataLCD[0]  = 0x48;                                     // H
        dataLCD[1]  = 0x3A;                                     // :
        dataLCD[2]  = tempData[2]/10 + 0x30;    // 10's temperature
        dataLCD[3]  = tempData[2]%10 + 0x30;  // 1's temperature
        dataLCD[4]  = 0xDF;                                     //
degree symbol
        dataLCD[5]  = 0x46;                                     // F
        dataLCD[6]  = 0x20;                                     //
Space
        dataLCD[7]  = 0x20;                                     //
Space
        dataLCD[8]  = 0x20;                                     //
Space
        dataLCD[9]  = 0x4C;                                     // L
        dataLCD[10] = 0x3A;                                     // :
        dataLCD[11] = tempData[1]/10 + 0x30;   // 10's temperature
        dataLCD[12] = tempData[1]%10 + 0x30; // 1's temperature
        dataLCD[13] = 0xDF;                                     //
degree symbol
        dataLCD[14] = 0x46;                                     // F
        dataLCD[15] = 0x20;                                     // space

        wr_scrn(&predLCD, 1);  // Writes to the top row of the LCD
        wr_scrn(&dataLCD, 0);  // Writes to the bottom row of the LCD
        break;

case '5':                                   // Display Highs and Lows for
pressure
        strcat(predLCD, "Pressure");
        dataLCD[0]  = 0x48;                                     // H
        dataLCD[1]  = 0x3A;                                     // :
        dataLCD[2]  = presData[2]/10 + 0x30;    // 10's pressure
        dataLCD[3]  = presData[2]%10 + 0x30;   // 1's pressure
        dataLCD[4]  = 0x22;                                     // "
        dataLCD[5]  = 0x20;                                     //
Space
        dataLCD[6]  = 0x20;                                     //
Space
        dataLCD[7]  = 0x20;                                     //
Space
        dataLCD[8]  = 0x20;                                     //
Space
        dataLCD[9]  = 0x4C;                                     // L
        dataLCD[10] = 0x3A;                                     // :
        dataLCD[11] = presData[1]/10 + 0x30;    // 10's pressure
        dataLCD[12] = presData[1]%10 + 0x30;  // 1's pressure
        dataLCD[13] = 0x22;                                     // "
        dataLCD[14] = 0x20;                                     //
Space
        dataLCD[15] = 0x20;                                     // space

        wr_scrn(&predLCD, 1);  // Writes to the top row of the LCD
        wr_scrn(&dataLCD, 0);  // Writes to the bottom row of the LCD
        break;

case '6':                                   // Display Highs and Lows for
humidity
        strcat(predLCD, "Humidity");
        dataLCD[0]  = 0x48;                                     // H
```

```
dataLCD[1]  = 0x3A;                                    // :
dataLCD[2]  = humiData[2]/100 + 0x30;  // 100's humidity
dataLCD[3]  = humiData[2]/10 + 0x30;   // 10's humidity
dataLCD[4]  = humiData[2]%10 + 0x30;  // 1's humidity
dataLCD[5]  = 0x25;                                    // %
dataLCD[6]  = 0x20;                                    //
Space
dataLCD[7]  = 0x20;                                    //
Space
dataLCD[8]  = 0x20;                                    //
Space
dataLCD[9]  = 0x4C;                                    // L
dataLCD[10] = 0x3A;                                    // :
dataLCD[11] = humiData[1]/100 + 0x30;  // 100's humidity
dataLCD[12] = humiData[1]/10 + 0x30;   // 10's humidity
dataLCD[13] = humiData[1]%10 + 0x30;  // 1's humidity
dataLCD[14] = 0x25;                                    // %
dataLCD[15] = 0x20;                                    // space

wr_scm(&predLCD, 1);  // Writes to the top row of the LCD
wr_scm(&dataLCD, 0);  // Writes to the bottom row of the LCD
break;

case '#':                                   // Testing mode.  Manualy enter
Flags
strcat(predLCD, "Enter Value");
strcat(dataLCD, "pSlope (1-5):");
wr_scm(&predLCD, 1);  // Writes to the top row of the LCD
wr_scm(&dataLCD, 0);  // Writes to the bottom row of the LCD
while(!keypressFlag);    // Wait for key press
keypressFlag = 0;                // Reset the Flag
pSlope = choice - 0x30;

DB12->strcpy(dataLCD, "");
strcat(dataLCD, "pFlag (1-3):");
wr_scm(&dataLCD, 0);  // Writes to the bottom row of the LCD
while(!keypressFlag);    // Wait for key press
keypressFlag = 0;                // Reset the Flag
pFlag = choice - 0x30;

DB12->strcpy(dataLCD, "");
strcat(dataLCD, "tFlag (1-3):");
wr_scm(&dataLCD, 0);  // Writes to the bottom row of the LCD
while(!keypressFlag);    // Wait for key press
keypressFlag = 0;                // Reset the Flag
tFlag = choice - 0x30;

DB12->strcpy(dataLCD, "");
strcat(dataLCD, "hFlag (1-2):");
wr_scm(&dataLCD, 0);  // Writes to the bottom row of the LCD
while(!keypressFlag);    // Wait for key press
keypressFlag = 0;                // Reset the Flag
hFlag = choice - 0x30;

choice = "1";                    // go back to default view
break;
```

```
case 'A':                                   // Test case scenario #1: Expect
Snow
// Initialize Highs, Lows, and current
tempData[0] = 30;
tempData[1] = 24;
tempData[2] = 31;

presData[0] = 31;
presData[1] = 27;
presData[2] = 30;

humiData[0] = 81;
humiData[1] = 35;
humiData[2] = 82;

// Initialize Prediction Data
for(i=0; i<24; i++)
{
        if(i%2)
                presPred[i][0] = 29-i*0.1;
        else
                presPred[i][0] = 29+i*0.05;
        presPred[i][1] = i;
}

hourOverflow = 1;
prediction();
choice = 1;
break;

case 'B':                                   // Test case scenario #2: Good
Weather
// Initialize Highs, Lows, and current
tempData[0] = 92;
tempData[1] = 72;
tempData[2] = 93;

presData[0] = 32;
presData[1] = 30;
presData[2] = 32;

humiData[0] = 54;
humiData[1] = 51;
humiData[2] = 76;

// Initialize Prediction Data
for(i=0; i<24; i++)
{
        if(i%2)
                presPred[i][0] = 27+i*0.15;
        else
                presPred[i][0] = 27-i*0.05;
        presPred[i][1] = i;
}
```

```
                hourOverflow = 1;
                prediction();
                choice = 1;
                break;

        case 'D':                                   // Return to previous values
                hourOverflow = 0;
                break;

        default:
                DB12->strcpy(upprLCD, "TMP  HMD PRS DIR");
                dataLCD[0]  = tempData[0]/10 + 0x30;   // 10's temperature
                dataLCD[1]  = tempData[0]%10 + 0x30;  // 1's temperature
                dataLCD[2]  = 0xDF;                                     //
degree symbol
                dataLCD[3]  = 0x46;                                     // F
                dataLCD[4]  = 0x20;                                     // space
                dataLCD[5]  = humiData[0]/10 + 0x30;   // 10's humidity
                dataLCD[6]  = humiData[0]%10 + 0x30;  // 1's humidity
                dataLCD[7]  = 0x25;                                     // %
                dataLCD[8]  = 0x20;                                     // space
                dataLCD[9]  = presData[0]/10 + 0x30;   // 10's pressure
                dataLCD[10] = presData[0]%10 + 0x30;  // 1's pressure
                dataLCD[11] = 0x22;                                     // "
                dataLCD[12] = 0x20;                                     // space
                dataLCD[13] = direction1;                       // N, S
                dataLCD[14] = direction2;                       // E, W
                dataLCD[15] = 0x20;                                     // space

                wr_scm(&upprLCD, 1);  // Writes to the top row of the LCD
                wr_scm(&dataLCD, 0);  // Writes to the bottom row of the LCD
                break;
        }
}


// RTI Interrupt routine
__mod2__ void RTIInt()
{
        _H12RTIFLG = 0x80;    // Clear flag
        RTIcount++;

        if(RTIcount >= 15)        // Every Second
        {
                secCounter++;
                sec = 1;

                if(secCounter >= 60) // Every Minute
                {
                        minCounter++;
                        secCounter = 0;
                        min = 1;

                        if(minCounter >= 60) // Every Hour
                        {
                                hrCounter++;
                                numHours++;
                                minCounter = 0;
                                hr = 1;

                                if(hrCounter >= 24) // Every Day
                                {
                                        hourOverflow = 1; // At least one day has passed
                                        hrCounter = 0;
                                }
                        }
                }
                RTIcount = 0;
        }
}


// Calculate Slope
int slope()
{
        int i, m, N, sumXY, sumX, sumY, sumX2, numerator, denominator, gain;
        gain  = 10;
        sumXY = 0;
        sumX  = 0;
        sumY  = 0;
        sumX2 = 0;

        if(hourOverflow)
                N = 24;
        else
                N = hrCounter;

        for(i=0; i<N; i++)
        {
                sumXY += presPred[i][0] * presPred[i][1];
                sumX  += presPred[i][0];
                sumY  += presPred[i][1];
                sumX2 += presPred[i][0] * presPred[i][0];
        }

        numerator   = ((N * sumXY) - (sumX * sumY)) * gain;
        denominator = (N * sumX2) - (sumX * sumX);
        m = numerator / denominator;
        return m;
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//      ECSE 4790
//      Lab 7c: Magic 8 Ball - Part 3
//      Group Members:          Eric Chambers, Michael Baj
//      Date: 10/11/00
//      File: keypad.h
//      Description:
//              Header file that holds the function needed for keypad usage.
//              See "Notes" for proper wiring of keypad.  KeypadInit() must be
//              called to properly activate the keypad.  Then the ISR is
//              triggered by a key press on the keypad.  The keyp pressed is
//              stored in "choice" as a global variable for the programmer to
//              use.  The keypressFlag must be reset before another keypress can
//              be detected.
//      Arguments:
//              n/a
//      Output:
//              Global Variable "choice".
//      Notes:
//              The row selection bits should be wired to the lower nibble of
//              Port J.  The column selection bits should be wired to the upper
//              nibble of Port J.
//
//      This code was written and tested using Introl C 4.0
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// includes
#include <hc812a4.h>
#include <dbug12.h>

// Function Prototypes
void KeypadInit();                      // initialize the keypad
__mod2__ void KeyWakeupISR();           // ISR for each keypad depression (KWI)

// Global Variables
int keypressFlag = 0;                   // notification of key press
char choice = 0;                        // the key that was pressed
char ASCIIlookup[0x89];                 // ASCII conversion lookup table

void KeypadInit()
{
        DB12->SetUserVector(PortJKey, KeyWakeupISR);

        _H12DDRJ  = 0x0F;               // Set the row bits (lower nibble) for output
                                        // Set the column bits (upper nibble) as input

        _H12PORTJ = _H12PORTJ & 0xF0;   // Write lows to the row select bits

        _H12KPOLJ = 0x00;               // Set up port J to detect falling edges
        _H12PUPSJ = 0xFF;               // Select Pull-ups for Port J
        _H12PULEJ = 0xF0;               // Enable Pull-ups on upper nibble (columns)
        _H12KWIEJ = 0xF0;               // Enable the upper nibble as KWI triggers
        _H12KWIFJ = 0xFF;               // Clear any KWI flags that may be set

        // ASCII conversion entries for possible keypad values.  The row byte
        // is logic-OR with the column byte to produce the index.
        ASCIIlookup[0x11] = '1';
        ASCIIlookup[0x21] = '2';
        ASCIIlookup[0x41] = '3';
        ASCIIlookup[0x81] = 'A';
        ASCIIlookup[0x12] = '4';
        ASCIIlookup[0x22] = '5';
        ASCIIlookup[0x42] = '6';
        ASCIIlookup[0x82] = 'B';
        ASCIIlookup[0x14] = '7';
        ASCIIlookup[0x24] = '8';
        ASCIIlookup[0x44] = '9';
        ASCIIlookup[0x84] = 'C';
        ASCIIlookup[0x18] = '*';
        ASCIIlookup[0x28] = '0';
        ASCIIlookup[0x48] = '#';
        ASCIIlookup[0x88] = 'D';
}

__mod2__ void KeyWakeupISR()
{
        int i,j;
        char column_loc = 0x00;
        char row_loc = 0x01;
        keypressFlag = 1;                       // toggle flag for key press

        column_loc = _H12KWIFJ & 0xF0;          // Save the state of the Key Wakeup
                                                // Flags to determine which column
                                                // the key press was in

        for(i=0; i<4; i++)
        {
                _H12PORTJ = row_loc & 0x0F;     // Toggle the row bit high
                for(j=0; j<150; j++);           // delay to stabilize toggle

                if((_H12PORTJ & 0xF0) == 0xF0)          // Check the column bits
                {
                        choice = ASCIIlookup[(column_loc | row_loc)];
                }
                row_loc = row_loc << 1;         // increment to the next row
        }
        _H12PORTJ = _H12PORTJ & 0xF0;           // Reset row bits on Port J to low
                                                // outputs
        while(!(column_loc & _H12PORTJ));       // Wait for key to be released
        _H12KWIFJ = 0xFF;                       // Clear the KWI flag
}
```

Appendix D: Prediction Charts

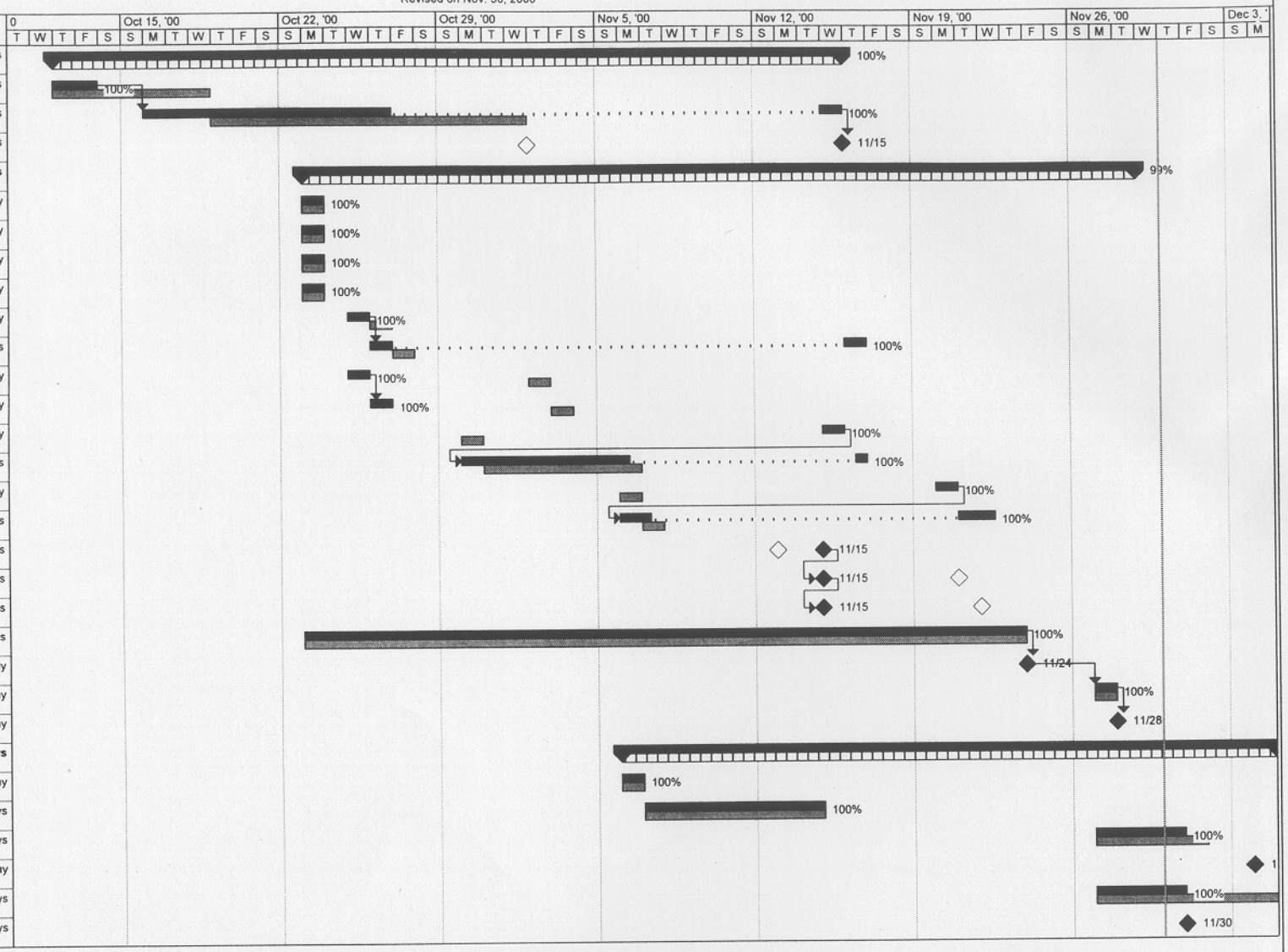
pSlope	pFlag	tFlag	hFlag	top LCD				bottom LCD
5	3	3	2	Rapid ↑	High	Hot	Sticky	Hot & Humid
5	3	3	1	Rapid ↑	High	Hot	Dry	Clear Skies are here again
5	3	2	2	Rapid ↑	High	Warm	Sticky	Cloudy and Fair
5	3	2	1	Rapid ↑	High	Warm	Dry	I see skies of blue and clouds of white
5	3	1	2	Rapid ↑	High	Cold	Sticky	Lovely Winter Day
5	3	1	1	Rapid ↑	High	Cold	Dry	Lovely Winter Day
5	2	3	2	Rapid ↑	Low	Hot	Sticky	Expect Rain within 12 hours
5	2	3	1	Rapid ↑	Low	Hot	Dry	Expext Rain within 24 hours
5	2	2	2	Rapid ↑	Low	Warm	Sticky	Expect Rain within 12 hours
5	2	2	1	Rapid ↑	Low	Warm	Dry	Expext Rain within 24 hours
5	2	1	2	Rapid ↑	Low	Cold	Sticky	Expect Snow
5	2	1	1	Rapid ↑	Low	Cold	Dry	Expext Snow
5	1	3	2	Rapid ↑	Avg	Hot	Sticky	Chance of Rain
5	1	3	1	Rapid ↑	Avg	Hot	Dry	Chance of Rain
5	1	2	2	Rapid ↑	Avg	Warm	Sticky	Chance of Rain
5	1	2	1	Rapid ↑	Avg	Warm	Dry	Chance of Rain
5	1	1	2	Rapid ↑	Avg	Cold	Sticky	Fair, Expect Rain
5	1	1	1	Rapid ↑	Avg	Cold	Dry	Fair, Cold day
4	3	3	2	Steady ↑	High	Hot	Sticky	Hot & Humid
4	3	3	1	Steady ↑	High	Hot	Dry	Clear Skies are here again
4	3	2	2	Steady ↑	High	Warm	Sticky	Cloudy and Fair
4	3	2	1	Steady ↑	High	Warm	Dry	I see skies of blue and clouds of white
4	3	1	2	Steady ↑	High	Cold	Sticky	Lovely Winter Day
4	3	1	1	Steady ↑	High	Cold	Dry	Lovely Winter Day
4	2	3	2	Steady ↑	Low	Hot	Sticky	Expect Rain within 12 hours
4	2	3	1	Steady ↑	Low	Hot	Dry	Expext Rain within 24 hours
4	2	2	2	Steady ↑	Low	Warm	Sticky	Expect Rain within 12 hours
4	2	2	1	Steady ↑	Low	Warm	Dry	Clearing, Fair for several days
4	2	1	2	Steady ↑	Low	Cold	Sticky	Expect Snow
4	2	1	1	Steady ↑	Low	Cold	Dry	Expext Snow
4	1	3	2	Steady ↑	Avg	Hot	Sticky	Chance of Rain
4	1	3	1	Steady ↑	Avg	Hot	Dry	Chance of Rain
4	1	2	2	Steady ↑	Avg	Warm	Sticky	Chance of Rain
4	1	2	1	Steady ↑	Avg	Warm	Dry	Chance of Rain
4	1	1	2	Steady ↑	Avg	Cold	Sticky	Fair, Expect Rain
4	1	1	1	Steady ↑	Avg	Cold	Dry	Fair, Cold day
3	3	3	2	No Ch	High	Hot	Sticky	Hot and Humid
3	3	3	1	No Ch	High	Hot	Dry	Hot and Fair
3	3	2	2	No Ch	High	Warm	Sticky	Warm and Humid
3	3	2	1	No Ch	High	Warm	Dry	Continued Fair
3	3	1	2	No Ch	High	Cold	Sticky	Clear and Cold
3	3	1	1	No Ch	High	Cold	Dry	Clear and Cold
3	2	3	2	No Ch	Low	Hot	Sticky	Warm and Rainy
3	2	3	1	No Ch	Low	Hot	Dry	High Chance of Rain
3	2	2	2	No Ch	Low	Warm	Sticky	High Chance of Rain
3	2	2	1	No Ch	Low	Warm	Dry	Chance of Rain
3	2	1	2	No Ch	Low	Cold	Sticky	High Chance of Snow
3	2	1	1	No Ch	Low	Cold	Dry	Chance of Snow
3	1	3	2	No Ch	Avg	Hot	Sticky	Hot and Humid

3	1	3	1	No Ch	Avg	Hot	Dry	Hot and Fair
3	1	2	2	No Ch	Avg	Warm	Sticky	Warm and Humid
3	1	2	1	No Ch	Avg	Warm	Dry	Continued Fair
3	1	1	2	No Ch	Avg	Cold	Sticky	Clear and Cold
3	1	1	1	No Ch	Avg	Cold	Dry	Clear and Cold
2	3	3	2	Steady ↓	High	Hot	Sticky	Expect Rain within 12 hours
2	3	3	1	Steady ↓	High	Hot	Dry	Expect Rain within 24 hours
2	3	2	2	Steady ↓	High	Warm	Sticky	Expect Rain within 12 hours
2	3	2	1	Steady ↓	High	Warm	Dry	Slow Rise in Temp, Fair for few days
2	3	1	2	Steady ↓	High	Cold	Sticky	Expect Snow
2	3	1	1	Steady ↓	High	Cold	Dry	Expect Snow
2	2	3	2	Steady ↓	Low	Hot	Sticky	Continued Storm
2	2	3	1	Steady ↓	Low	Hot	Dry	Continued Storm
2	2	2	2	Steady ↓	Low	Warm	Sticky	Continued Rain
2	2	2	1	Steady ↓	Low	Warm	Dry	Continued Storm
2	2	1	2	Steady ↓	Low	Cold	Sticky	Expect Rain with few Hours
2	2	1	1	Steady ↓	Low	Cold	Dry	Continued Snow
2	1	3	2	Steady ↓	Avg	Hot	Sticky	Expect Rain
2	1	3	1	Steady ↓	Avg	Hot	Dry	Expect Rain
2	1	2	2	Steady ↓	Avg	Warm	Sticky	Wind Increasing, Expect rain
2	1	2	1	Steady ↓	Avg	Warm	Dry	Expect Rain
2	1	1	2	Steady ↓	Avg	Cold	Sticky	Expect Snow
2	1	1	1	Steady ↓	Avg	Cold	Dry	Possibly Snow
1	3	3	2	Rapid ↓	High	Hot	Sticky	Expect Rain within 12 hours
1	3	3	1	Rapid ↓	High	Hot	Dry	Expect Rain within 24 hours
1	3	2	2	Rapid ↓	High	Warm	Sticky	Expect Rain within 12 hours
1	3	2	1	Rapid ↓	High	Warm	Dry	Expect Rain within 24 hours
1	3	1	2	Rapid ↓	High	Cold	Sticky	Expect Snow
1	3	1	1	Rapid ↓	High	Cold	Dry	Expect Snow
1	2	3	2	Rapid ↓	Low	Hot	Sticky	Continued Storm
1	2	3	1	Rapid ↓	Low	Hot	Dry	Continued Storm
1	2	2	2	Rapid ↓	Low	Warm	Sticky	Rain with High Winds, Expect Storm
1	2	2	1	Rapid ↓	Low	Warm	Dry	Continued Storm
1	2	1	2	Rapid ↓	Low	Cold	Sticky	Continued Blizzard
1	2	1	1	Rapid ↓	Low	Cold	Dry	Continued Snow
1	1	3	2	Rapid ↓	Avg	Hot	Sticky	Expect Rain
1	1	3	1	Rapid ↓	Avg	Hot	Dry	Expect Rain
1	1	2	2	Rapid ↓	Avg	Warm	Sticky	Wind Increasing, Expect rain
1	1	2	1	Rapid ↓	Avg	Warm	Dry	Expect Rain
1	1	1	2	Rapid ↓	Avg	Cold	Sticky	Expect Snow
1	1	1	1	Rapid ↓	Avg	Cold	Dry	Possibly Snow

Appendix E: Gantt Chart

Microprocessor Systems
ECSE 4790
Section A
Revised on Nov. 30, 2000

ID		Task Name	Duration
1	✓	**Material Gathering**	**25 days**
2	✓	Component Price Comparis	2 days
3	✓	Purchase Components	10 days
4	✓	Purchase Complete	0 days
5		**Assembly**	**27 days**
6	✓	Attach LCD	1 day
7	✓	Program LCD Module	1 day
8	✓	Attach Scroll Button	1 day
9	✓	Program Scroll Module	1 day
10	✓	Attach Temp Sensor	1 day
11	✓	Program Temp Module	2 days
12	✓	Attach Humidity Sensor	1 day
13	✓	Program Humidity Module	1 day
14	✓	Attach Compass	1 day
15	✓	Program Compass Module	6 days
16	✓	Attach Pressure Sensor	1 day
17	✓	Program Pressure Module	3 days
18		Build Wind Sensor	0 days
19		Attach Wind Sensor	0 days
20		Program Wind Module	0 days
21	✓	Program Meteorological Pr	24 days
22	✓	Modules Complete	1 day
23	✓	Program Main Routine	1 day
24	✓	Assembly Complete	1 day
25	✓	**Evaluation**	**21 days**
26	✓	Demonstration	1 day
27	✓	Progress Report	6 days
28	✓	Prepare Presentation	4 days
29	✓	Presentation	1 day
30	✓	Prepare Final Report	4 days
31	✓	Final Report	0 days

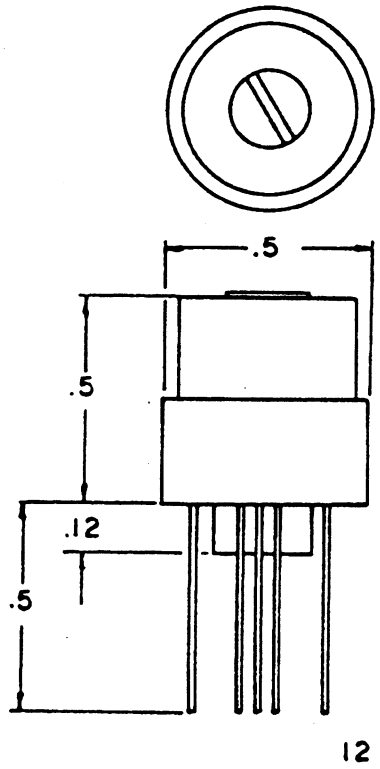


Project: Outdoorsman Weathermate P
Team: Michael Baj & Eric Chambers
Date: Thu 11/30/00

Critical	Baseline	Rolled Up Critical	Rolled Up Baseline
Critical Split	Baseline Split	Rolled Up Critical Split	Rolled Up Baseline Milestone
Critical Progress	Baseline Milestone	Rolled Up Critical Progress	Rolled Up Milestone
Task	Milestone	Rolled Up Task	External Tasks
Split	Summary Progress	Rolled Up Split	Project Summary
Task Progress	Summary	Rolled Up Task Progress	

Appendix F: Data Sheets

No. 1490 DIGITAL SENSOR Updated 06 December 1993



No. 1490 Sensor may be operated from input voltage of 5 to 20 volts DC with 8 to 13 recommended. Input should be both "spike" and polarity protected. Power requirement is approximately 30 mils (0.030 amps).

Output will sink up to 25 mils (0.025 amps) per output channel. The sensor will switch so that no more than two adjacent output channels are "on" at any one time.

Output is **open collector NPN** sinking the output to ground, thus does not add to the input requirement.

No. 1490 Sensor is internally designed to respond to directional change similar to a liquid filled compass. It will return to the indicated direction from a 90^0 displacement in approximately 2.5 to 3.5 seconds with no overswing.

Sensor No. 1490 should be operated in a vertical position. The sensor indicates the horizontal component or compass component of the earth's field. If off vertical, some of the vertical component of the earth's field is introduced which may create some directional error. Generally, tilt up to 12^0 is acceptable with little error.

The sensor is manufactured for pins down operation but may be furnished for pins up operation on request at no extra cost. The sensor operates equally well pins up or down.

No. 1490 sensor weighs approximately 2.25 grams. The dimensions are shown on the drawing upper left. Operating temperature is -20^0 C to $+85^0$ C. The sensor may be stored without damage in wider temperature limits and may be subjected to high flux levels (up to 1000 gauss) without permanent damage.

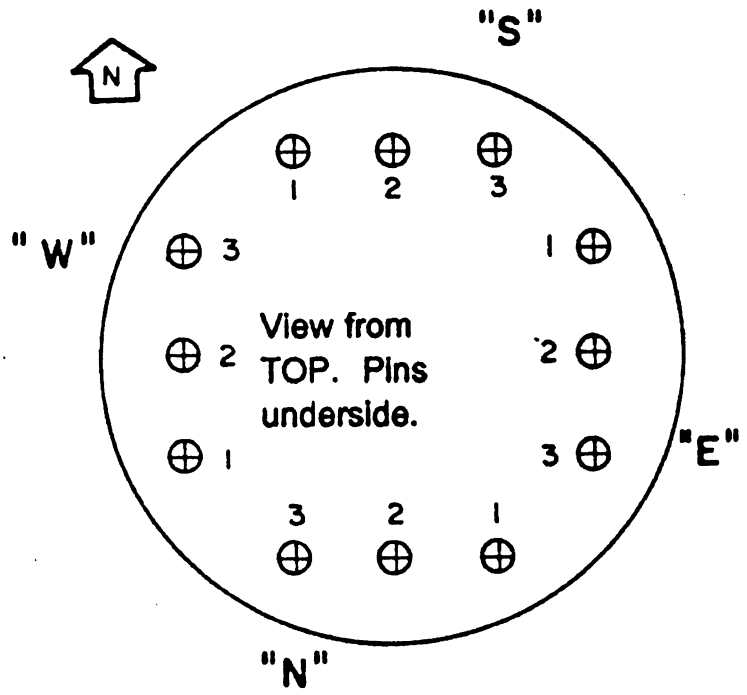
No. 1490 sensor and sensing systems are covered by isued patents and patents pending.

The pins are on 0.050 centers but may be distorted for 0.100 spacing without damage to the sensor or its measurements. The four V_{cc} and four grounds may be common connected.

DINSMORE INSTRUMENT COMPANY
1814 REMELL STREET
FLINT, MI 48503 USA
Tel: 810 744 1330 Fax: 810 744 1790

PIN-OUT FOR DIGITAL COMPASS SENSOR (No. 1490)

output coding
when this is:



1 = V_{cc}

2 = Ground

3 = Signal

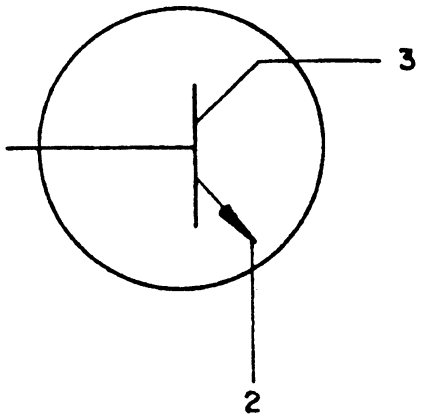
V_{cc} = 5 to 20 Volts;
8 to 13 recommended.

Output will sink
25 mils @ 12 Volts.

Temperature Range:
-20° C to +85° C.

Reverse polarity
will damage Ic

Output is Open Collector



DINSMORE INSTRUMENT COMPANY
1814 REMELL STREET
FLINT, MI 48503 USA

Tel: 810 744 1330
Fax: 810 744 1790

Updated 05 December 1993

Order this document
by MPX2200/D

200 kPa On-Chip Temperature Compensated & Calibrated Pressure Sensors

MPX2200
MPX2201
SERIES
Motorola Preferred Device

0 to 200 kPa (0 to 29 psi)
40 mV FULL SCALE SPAN
(TYPICAL)

The MPX2200 and MPX2201 series device is a silicon piezoresistive pressure sensor providing a highly accurate and linear voltage output — directly proportional to the applied pressure. The sensor is a single monolithic silicon diaphragm with the strain gauge and a thin–film resistor network integrated on–chip. The chip is laser trimmed for precise span and offset calibration and temperature compensation. They are designed for use in applications such as pump/motor controllers, robotics, level indicators, medical diagnostics, pressure switching, barometers, altimeters, etc.

Features

- Temperature Compensated Over 0°C to +85°C
- Patented Silicon Shear Stress Strain Gauge
- ±0.25% Linearity (MPX2200D)
- Easy to Use Chip Carrier Package
- Available in Absolute, Differential and Gauge Configurations

Application Examples

- Pump/Motor Controllers
- Robotics
- Level Indicators
- Medical Diagnostics
- Pressure Switching
- Barometers
- Altimeters

Figure 1 illustrates a block diagram of the internal circuitry on the stand–alone pressure sensor chip.

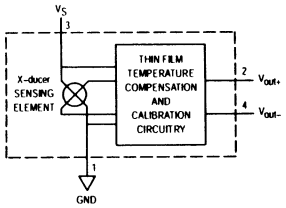
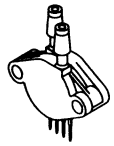


Figure 1. Temperature Compensated Pressure Sensor Schematic

BASIC CHIP
CARRIER ELEMENT
CASE 344–15, STYLE 1



DIFFERENTIAL
PORT OPTION
CASE 344C–01, STYLE 1

NOTE: Pin 1 is the notched pin.

PIN NUMBER			
1	Gnd	3	V_S
2	$+V_{out}$	4	$-V_{out}$

VOLTAGE OUTPUT versus APPLIED DIFFERENTIAL PRESSURE

The differential voltage output of the X–ducer is directly proportional to the differential pressure applied.

The absolute sensor has a built–in reference vacuum. The output voltage will decrease as vacuum, relative to ambient, is drawn on the pressure (P1) side.

The output voltage of the differential or gauge sensor increases with increasing pressure applied to the pressure (P1) side relative to the vacuum (P2) side. Similarly, output voltage increases as increasing vacuum is applied to the vacuum (P2) side relative to the pressure (P1) side.

Preferred devices are Motorola recommended choices for future use and best overall value.

X–ducer is a trademark of Motorola, Inc.

REV 8

MOTOROLA

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Overpressure(8) (P1 > P2)	P_{max}	400	kPa
Burst Pressure(8) (P1 > P2)	P_{burst}	2000	kPa
Storage Temperature	T_{stg}	–40 to +125	°C
Operating Temperature	T_A	–40 to +125	°C

OPERATING CHARACTERISTICS (V_S = 10 Vdc, T_A = 25°C unless otherwise noted, P1 > P2)

Characteristics		Symbol	Min	Typ	Max	Unit
Pressure Range(1)		P_{OP}	0	—	200	kPa
Supply Voltage		V_S	—	10	16	Vdc
Supply Current		I_o	—	6.0	—	mAdc
Full Scale Span(3)		V_{FSS}	38.5	40	41.5	mV
Offset(4)		V_{off}	–1.0	—	1.0	mV
Sensitivity		$\Delta V/\Delta P$	—	0.2	—	mV/kPa
Linearity(5)	MPX2200D Series MPX2200A Series MPX2201GP	—	–0.25 –1.0 –0.5	— — —	0.25 1.0 0.5	$\%V_{FSS}$
Pressure Hysteresis(5) (0 to 200 kPa)		—	—	±0.1	—	$\%V_{FSS}$
Temperature Hysteresis(5) (–40°C to +125°C)		—	—	±0.5	—	$\%V_{FSS}$
Temperature Effect on Full Scale Span(5)		TCV_{FSS}	–1.0	—	1.0	$\%V_{FSS}$
Temperature Effect on Offset(5)		TCV_{off}	–1.0	—	1.0	mV
Input Impedance		Z_{in}	1300	—	2500	Ω
Output Impedance		Z_{out}	1400	—	3000	Ω
Response Time(6) (10% to 90%)		t_R	—	1.0	—	ms
Warm–Up		—	—	20	—	ms
Offset Stability(9)		—	—	±0.5	—	$\%V_{FSS}$

MECHANICAL CHARACTERISTICS

Characteristics	Symbol	Min	Typ	Max	Unit
Weight, (Basic Element Case 344–15)	—	—	2.0	—	Grams
Common Mode Line Pressure(7)	—	—	—	690	kPa

NOTES:

1. 1.0 kPa (kiloPascal) equals 0.145 psi.
2. Device is ratiometric within this specified excitation range. Operating the device above the specified excitation range may induce additional error due to device self–heating.
3. Full Scale Span (V_{FSS}) is defined as the algebraic difference between the output voltage at full rated pressure and the output voltage at the minimum rated pressure.
4. Offset (V_{off}) is defined as the output voltage at the minimum rated pressure.
5. Accuracy (error budget) consists of the following:
   - Linearity: Output deviation from a straight line relationship with pressure, using end point method, over the specified pressure range.
   - Temperature Hysteresis: Output deviation at any temperature within the operating temperature range, after the temperature is cycled to and from the minimum or maximum operating temperature points, with zero differential pressure applied.
   - Pressure Hysteresis: Output deviation at any pressure within the specified range, when this pressure is cycled to and from the minimum or maximum rated pressure, at 25°C.
   - TcSpan: Output deviation at full rated pressure over the temperature range of 0 to 85°C, relative to 25°C.
   - TcOffset: Output deviation with minimum rated pressure applied, over the temperature range of 0 to 85°C, relative to 25°C.
6. Response Time is defined as the time for the incremental change in the output to go from 10% to 90% of its final value when subjected to a specified step change in pressure.
7. Common mode pressures beyond specified may result in leakage at the case–to–lead interface.
8. Exposure beyond these limits may cause permanent damage or degradation to the device.
9. Offset stability is the product's output deviation when subjected to 1000 hours of Pulsed Pressure, Temperature Cycling with Bias Test.

National Semiconductor

December 1994

LM34/LM34A/LM34C/LM34CA/LM34D Precision Fahrenheit Temperature Sensors

General Description

The LM34 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Fahrenheit temperature. The LM34 thus has an advantage over linear temperature sensors calibrated in degrees Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Fahrenheit scaling. The LM34 does not require any external calibration or trimming to provide typical accuracies of ±½°F at room temperature and ±1½°F over a full −50 to +300°F temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM34's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies or with plus and minus supplies. As it draws only 75 μA from its supply, it has very low self-heating, less than 0.2°F in still air. The LM34 is rated to operate over a −50° to +300°F temperature range, while the LM34C is rated for a −40° to +230°F range (0°F with improved accuracy). The LM34 series is available packaged in hermetic TO-46 transistor packages, while the LM34C, LM34CA and LM34D are also available in the plastic TO-92 transistor package. The LM34D is also available in an 8-lead surface mount small outline package. The LM34 is a complement to the LM35 (Centigrade) temperature sensor.

Features

- Calibrated directly in degrees Fahrenheit
- Linear +10.0 mV/°F scale factor
- 1.0°F accuracy guaranteed (at +77°F)
- Rated for full −50° to +300°F range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 5 to 30 volts
- Less than 90 μA current drain
- Low self-heating, 0.18°F in still air
- Nonlinearity only ±0.5°F typical
- Low-impedance output, 0.4Ω for 1 mA load

Connection Diagrams

TO-46
Metal Can Package*

TL/H/6685–1

*Case is connected to negative pin (GND)

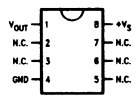
Order Numbers LM34H, LM34AH, LM34CH, LM34CAH or LM34DH
See NS Package Number H03H

TO-92
Plastic Package

TL/H/6685–2

Order Number LM34CZ, LM34CAZ or LM34DZ
See NS Package Number Z03A

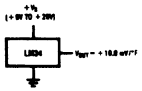
SO-8
Small Outline Molded Package



TL/H/6685–20

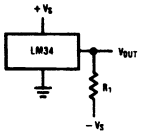
Top View
N.C. = No Connection
Order Number LM34DM
See NS Package Number M08A

Typical Applications



TL/H/6685–3

FIGURE 1. Basic Fahrenheit Temperature Sensor (+5° to +300°F)



CHOOSE $R_1 = (-V_S)/50$ μA
$V_{OUT} = +3.000$ mV AT +300°F
$= +750$ mV AT +75°F
$= -500$ mV AT −50°F

TL/H/6685–4

FIGURE 2. Full-Range Fahrenheit Temperature Sensor

TRI-STATE® is a registered trademark of National Semiconductor Corporation

RRD-B30M75/Printed in U.S.A.

Absolute Maximum Ratings (Note 10)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage	+35V to −0.2V
Output Voltage	+6V to −1.0V
Output Current	10 mA
Storage Temperature,	
TO-46 Package	−76°F to +356°F
TO-92 Package	−76°F to +300°F
SO-8 Package	−65°C to +150°C
ESD Susceptibility (Note 11)	800V
Lead Temp.	
TO-46 Package (Soldering, 10 seconds)	+300°C
TO-92 Package (Soldering, 10 seconds)	+260°C
SO Package (Note 12):	
Vapor Phase (60 seconds)	215°C
Infrared (15 seconds)	220°C

Specified Operating Temp. Range (Note 2)

	T_{MIN} to T_{MAX}
LM34, LM34A	−50°F to +300°F
LM34C, LM34CA	−40°F to +230°F
LM34D	+32°F to +212°F

DC Electrical Characteristics (Note 1, Note 6)

Parameter	Conditions	LM34A Typical	LM34A Tested Limit (Note 4)	LM34A Design Limit (Note 5)	LM34CA Typical	LM34CA Tested Limit (Note 4)	LM34CA Design Limit (Note 5)	Units (Max)
Accuracy (Note 7)	T_A = +77°F	±0.4	±1.0		±0.4	±1.0		°F
	T_A = 0°F	±0.6			±0.6		±2.0	°F
	T_A = T_{MAX}	±0.8	±2.0		±0.8	±2.0		°F
	T_A = T_{MIN}	±0.8	±2.0		±0.8		±3.0	°F
Nonlinearity (Note 8)	$T_{MIN} \le T_A \le T_{MAX}$	**±0.35**		**±0.7**	**±0.30**		**±0.6**	°F
Sensor Gain (Average Slope)	$T_{MIN} \le T_A \le T_{MAX}$	**+10.0**	**+9.9, +10.1**		**+10.0**		**+9.9, +10.1**	mV/°F, min mV/°F, max
Load Regulation (Note 3)	T_A = +77°F	±0.4	±1.0		±0.4	±1.0		mV/mA
	$T_{MIN} \le T_A \le T_{MAX}$ $0 \le I_L \le 1$ mA	**±0.5**		**±3.0**	**±0.5**		**±3.0**	mV/mA
Line Regulation (Note 3)	T_A = +77°F	±0.01	±0.05		±0.01	±0.05		mV/V
	$5V \le V_S \le 30V$	**±0.02**		**±0.1**	**±0.02**		**±0.1**	mV/V
Quiescent Current (Note 9)	V_S = +5V, +77°F	75	90		75	90		μA
	V_S = +5V	**131**		**160**	**116**		**139**	μA
	V_S = +30V, +77°F	76	92		76	92		μA
	V_S = +30V	**132**		**163**	**117**		**142**	μA
Change of Quiescent Current (Note 3)	$4V \le V_S \le 30V$, +77°F	+0.5	2.0		0.5	2.0		μA
	$5V \le V_S \le 30V$	**+1.0**		**3.0**	**1.0**		**3.0**	μA
Temperature Coefficient of Quiescent Current		**+0.30**		**+0.5**	**+0.30**		**+0.5**	μA/°F
Minimum Temperature for Rated Accuracy	In circuit of *Figure 1*, I_L = 0	+3.0		+5.0	+3.0		+5.0	°F
Long-Term Stability	T_J = T_{MAX} for 1000 hours	±0.16			±0.16			°F

Note 1: Unless otherwise noted, these specifications apply: −50°F ≤ T_J ≤ +300°F for the LM34 and LM34A; −40°F ≤ T_J ≤ +230°F for the LM34C and LM34CA; and +32°F ≤ T_J ≤ +212°F for the LM34D. V_S = +5 Vdc and I_{LOAD} = 50 μA in the circuit of *Figure 2*; +6 Vdc for LM34 and LM34A for 230°F ≤ T_J ≤ 300°F. These specifications also apply from +5°F to T_{MAX} in the circuit of *Figure 1*.

Note 2: Thermal resistance of the TO-46 package is 720°F/W junction to ambient and 43°F/W junction to case. Thermal resistance of the TO-92 package is 324°F/W junction to ambient. Thermal resistance of the small outline molded package is 400°F/W junction to ambient. For additional thermal resistance information see table in the Typical Applications section.

Note 3: Regulation is measured at constant junction temperature using pulse testing with a low duty cycle. Changes in output due to heating effects can be computed by multiplying the internal dissipation by the thermal resistance.

Note 4: Tested limits are guaranteed and 100% tested in production.

Note 5: Design limits are guaranteed (but not 100% production tested) over the indicated temperature and supply voltage ranges. These limits are not used to calculate outgoing quality levels.

Note 6: Specification in **BOLDFACE TYPE** apply over the full rated temperature range.

Note 7: Accuracy is defined as the error between the output voltage and 10 mV/°F times the device's case temperature at specified conditions of voltage, current, and temperature (expressed in °F).

Note 8: Nonlinearity is defined as the deviation of the output-voltage-versus-temperature curve from the best-fit straight line over the device's rated temperature range.

Note 9: Quiescent current is defined in the circuit of *Figure 1*.

Note 10: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its rated operating conditions (see Note 1).

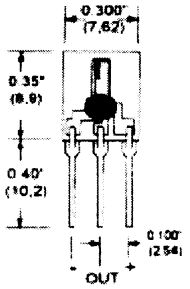
Note 11: Human body model, 100 pF discharged through a 1.5 kΩ resistor.

Note 12: See AN-450 "Surface Mounting Methods and Their Effect on Product Reliability" or the section titled "Surface Mount" found in a current National Semiconductor Linear Data Book for other methods of soldering surface mount devices.

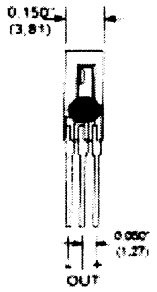
(15546) (ne10701)

Honeywell

MICRO SWITCH
HIH-3605 SERIES HUMIDITY SENSORS



IH-3605-A



IH-3605-B

Features:

- **Linearity: ± 2.0% RH**
- **RH Range: 0–100% RH**
- **Temperature Range: –40°C to +85°C**
- **Accuracy*: ±2.0% RH**
- **Voltage Output: Vout = Vsupply (0.0062 (% RH) + 0.16) @ 25°C**
- **Temperature Compensated: True RH = (% RH)/(1.0546-0.00216 T); T=°C**

Laser-trimmed RHIC chip provides linear voltage output Vs % RH, fast response, and resistance to wetting and chemicals with good stability. Supply voltage: 4–9VDC.

Stock No.	Type	Lead Space	Data Included	% RH Interchange-ability
91F7135	HIH-3605-A	0.100"	No	±5%
91F7136	HIH-3605-A-CP	0.100"	Yes	±5%
91F7137	HIH-3605-B	0.050"	No	±5%
91F7138	HIH-3605-B-CP	0.050"	Yes	±5%

* Accuracy figured when calibrating individual sensor performance using data printout

Applications:

- **Refrigeration.**
- **Drying**
- **Battery-powered systems**

ANALOG DEVICES

High Accuracy 8-Pin Instrumentation Amplifier

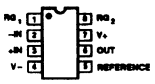
AMP02

FEATURES
- **Low Offset Voltage: 100 µV max**
- **Low Drift: 2 µV/°C max**
- **Wide Gain Range 1 to 10,000**
- **High Common-Mode Rejection: 115 dB min**
- **High Bandwidth (G = 1000): 200 kHz typ**
- **Gain Equation Accuracy: 0.5% max**
- **Single Resistor Gain Set**
- **Input Overvoltage Protection**
- **Low Cost**
- **Available In Die Form**

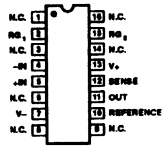
APPLICATIONS
- **Differential Amplifier**
- **Strain Gauge Amplifier**
- **Thermocouple Amplifier**
- **RTD Amplifier**
- **Programmable Gain Instrumentation Amplifier**
- **Medical Instrumentation**
- **Data Acquisition Systems**

PIN CONNECTIONS

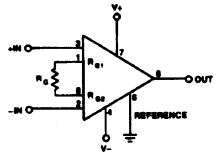
Epoxy Mini-DIP (P Suffix) and Cerdip (Z Suffix)



16-Pin SOL (S Suffix)



NC = NO CONNECT



$$G = \frac{V_{OUT}}{(+IN) - (-IN)} = \left(\frac{50k\Omega}{R_G}\right) + 1$$

FOR SOL CONNECT SENSE TO OUTPUT

Figure 1. Basic Circuit Connections

GENERAL DESCRIPTION

The AMP02 is the first precision instrumentation amplifier available in an 8-pin package. Gain of the AMP02 is set by a single external resistor, and can range from 1 to 10,000. No gain set resistor is required for unity gain. The AMP02 includes an input protection network that allows the inputs to be taken 60 V beyond either supply rail without damaging the device.

Laser trimming reduces the input offset voltage to under 100 µV. Output offset voltage is below 4 mV and gain accuracy is better than 0.5% for gain of 1000. PMI's proprietary thin-film resistor process keeps the gain temperature coefficient under 50 ppm/°C.

Due to the AMP02's design, its bandwidth remains very high over a wide range of gain. Slew rate is over 4 V/µs making the AMP02 ideal for fast data acquisition systems.

A reference pin is provided to allow the output to be referenced to an external dc level. This pin may be used for offset correction or level shifting as required. In the 8-pin package, sense is internally connected to the output.

For an instrumentation amplifier with the highest precision, consult the AMP01 data sheet. For the highest input impedance and speed, consult the AMP05 data sheet.

REV. D

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 617/329-4700 **Fax: 617/326-8703**

AMP02–SPECIFICATIONS

ELECTRICAL CHARACTERISTICS (@ V_S = ±15 V, V_{CM} = 0 V, T_A = +25°C, unless otherwise noted.)

Parameter	Symbol	Conditions	AMP02E Min	AMP02E Typ	AMP02E Max	AMP02F Min	AMP02F Typ	AMP02F Max	Units
OFFSET VOLTAGE									
Input Offset Voltage	V_{IOS}	T_A = +25°C		20	100		40	200	µV
		40°C ≤ T_A ≤ +85°C		50	200		100	350	µV
Input Offset Voltage Drift	TCV_{IOS}	40°C ≤ T_A ≤ +85°C		0.5	2		1	4	µV/°C
Output Offset Voltage	V_{OOS}	T_A = +25°C		1	4		2	8	mV
		-40°C ≤ T_A ≤ +85°C		4	10		9	20	mV
Output Offset Voltage Drift	TCV_{OOS}	-40°C ≤ T_A ≤ +85°C		50	100		100	200	µV/°C
Power Supply Rejection	PSR	V_S = ±4.8 V to ±18 V							
		G = 100, 1000	115	125		110	115		dB
		G = 10	100	110		95	100		dB
		G = 1	80	90		75	80		dB
		V_S = ±4.8 V to ±18 V							
		-40°C ≤ T_A ≤ +85°C							
		G = 1000, 100	110	120		105	110		dB
		G = 10	95	110		90	95		dB
		G = 1	75	90		70	75		dB
INPUT CURRENT									
Input Bias Current	I_B	T_A = +25°C		2	10		4	20	nA
Input Bias Current Drift	TCI_B	40°C ≤ T_A ≤ +85°C		150			250		pA/°C
Input Offset Current	I_{OS}	T_A = +25°C		1.2	5		2	10	nA
Input Offset Current Drift	TCI_{OS}	-40°C ≤ T_A ≤ +85°C		9			15		pA/°C
INPUT									
Input Resistance	R_{IN}	Differential, G ≤ 1000		10			10		GΩ
		Common-Mode, G = 1000		16.5			16.5		GΩ
Input Voltage Range	IVR	T_A = +25°C (Note 1)	±11			±11			V
Common-Mode Rejection	CMR	V_{CM} = ±11 V							
		G = 1000, 100	115	120		110	115		dB
		G = 10	100	115		95	110		dB
		G = 1	80	95		75	90		dB
		V_{CM} = ±11 V							
		-40°C ≤ T_A ≤ +85°C							
		G = 100, 1000	110	120		105	115		dB
		G = 10	95	110		90	105		dB
		G = 1	75	90		70	85		dB
GAIN									
Gain Equation Accuracy	$G = \frac{50\,k\Omega}{R_G} + 1$	G = 1000			0.50			0.70	%
		G = 100			0.30			0.50	%
		G = 10			0.25			0.40	%
		G = 1			0.02			0.05	%
Gain Range	G		1		10k	1		10k	V/V
Nonlinearity		G = 1 to 1000		0.006			0.006		%
Temperature Coefficient	G_{TC}	1 ≤ G ≤ 1000 (Notes 2, 3)		20	50		20	50	ppm/°C
OUTPUT RATING									
Output Voltage Swing	V_{OUT}	T_A = +25°C, R_L = 1 kΩ	±12	±13		±12	±13		V
		R_L = 1 kΩ, -40°C ≤ T_A ≤ +85°C	±11	±12		±11	±12		V
Positive Current Limit		Output-to-Ground Short		22			22		mA
Negative Current Limit		Output-to-Ground Short		32			32		mA
NOISE									
Voltage Density, RTI	e_n	f_O = 1 kHz							
		G = 1000		9			9		nV/√Hz
		G = 100		10			10		nV/√Hz
		G = 10		18			18		nV/√Hz
		G = 1		120			120		nV/√Hz
Noise Current Density, RTI	i_n	f_O = 1 kHz, G = 1000		0.4			0.4		pA/√Hz
Input Noise Voltage	e_n p-p	0.1 Hz to 10 Hz							
		G = 1000		0.4			0.4		µV p-p
		G = 100		0.5			0.5		µV p-p
		G = 10		1.2			1.2		µV p-p
DYNAMIC RESPONSE									
Small Signal Bandwidth (-3 dB)	BW	G = 1		1200			1200		kHz
		G = 10		300			300		kHz
G = 100, 1000				200			200		kHz
Slew Rate	SR	G = 10, R_L = 1 kΩ	4	6		4	6		V/µs
Settling Time	t_S	To 0.01% ±10 V Step							
		G = 1 to 1000		10			10		µs
SENSE INPUT									
Input Resistance	R_{IN}			25			25		kΩ
Voltage Range				±11			±11		V
REFERENCE INPUT									
Input Resistance	R_{IN}			50			50		kΩ
Voltage Range				±11			±11		V
Gain to Output				1			1		V/V

TO: Professor R. P. Kraft
FROM: Eric Chambers and Michael Baj
SUBJECT: Outdoorsman Weather-Mate Pro
DATE: October 7, 2000

The Outdoorsman Weather-Mate Pro is a companion to outdoor adventurers providing them with vital weather statistics. Every outdoorsman will find this to be a valuable tool in avoidance of Mother Nature's wrath.

Problem

People in the outdoors have a need to know what the weather conditions are and what they will be so that they can take the necessary actions to ensure their safety while enjoying the outdoors. Instruments that exist today are heavy, bulky, and too expensive for the average adventurer. Everybody from family picnickers to the highly adventurous will find the Outdoorsman Weather-Mate Pro to be a compact, lightweight, and cost effective solution to their meteorological needs.

Goal

- Construct an electronic device that gathers, stores, and processes meteorological data.

Plan of Action

Through the use of the Motorola 68HC12A4 EVB, Protoboard, electronic sensors, and LCD screens, we will be able to construct a working prototype that can gather the environmental information, process the data and provide the user with the desired information. We can use electronic sensors to gather the environmental information such as: temperature, pressure, humidity, and cardinal orientation. This information can be feed into the Motorola 68HC12A4 via analog and digital input lines and then processed, stored and fed back to the user with the use of an LCD. If the project is deemed successful and cost worthy, the prototype circuitry can be integrated into a printed circuit board system for mass production.

Please refer to the attached Gantt Chart for schedule details.

Parts List:

1. Motorola 68HC12A4 microcontroller with EVB
2. LM74 Temperature probe
3. Magnetic compass with digital output
4. Humidity sensor
5. Pressure sensor
6. Wind speed transducer (to be built)

Budget:

Motorola 68HC12A4		*
Sensors		
LM74 Temperature probe		4.00
Magnetic compass with digital output		50.00
Humidity sensor		40.00
Pressure sensor		15.00
Wind speed transducer		TBD
Other misc. parts		10.00
	Total	$119.00

* The Motorola 68HC12A4 EVB and protoboard will be provided by ECSE Dept.

TBD The wind transducer will be constructed if time allows therefore the cost cannot be determined until the time of construction.

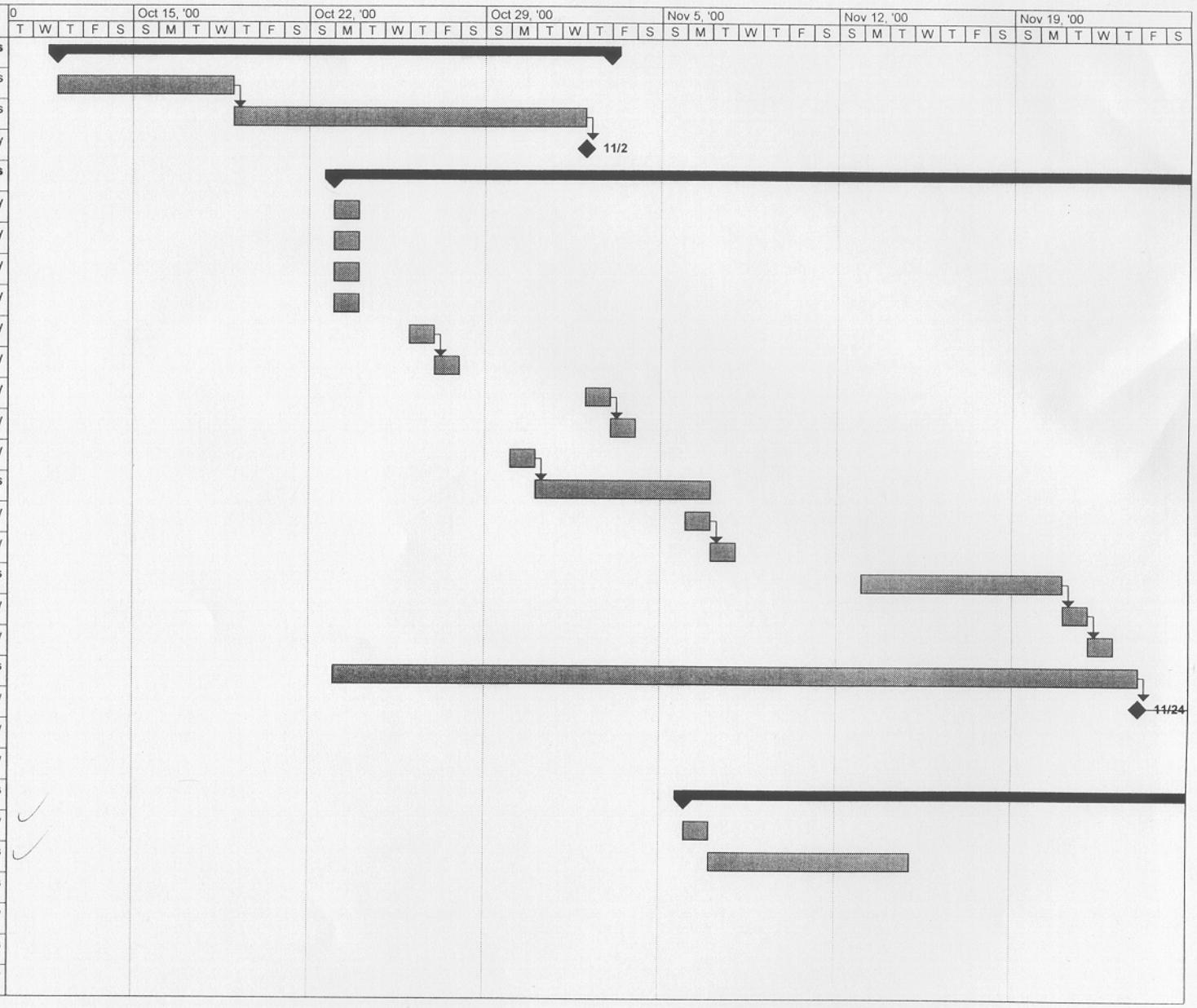
Evaluation

The projects goals will be considered satisfied upon meeting the following criteria:

- At least 3 of the 5 environmental transducers are properly interfaced to the mircocontroller.
- The information is successfully stored in memory
- The information is successfully processed and displayed on the LCD

Microprocessor Systems
ECSE 4790
Section A

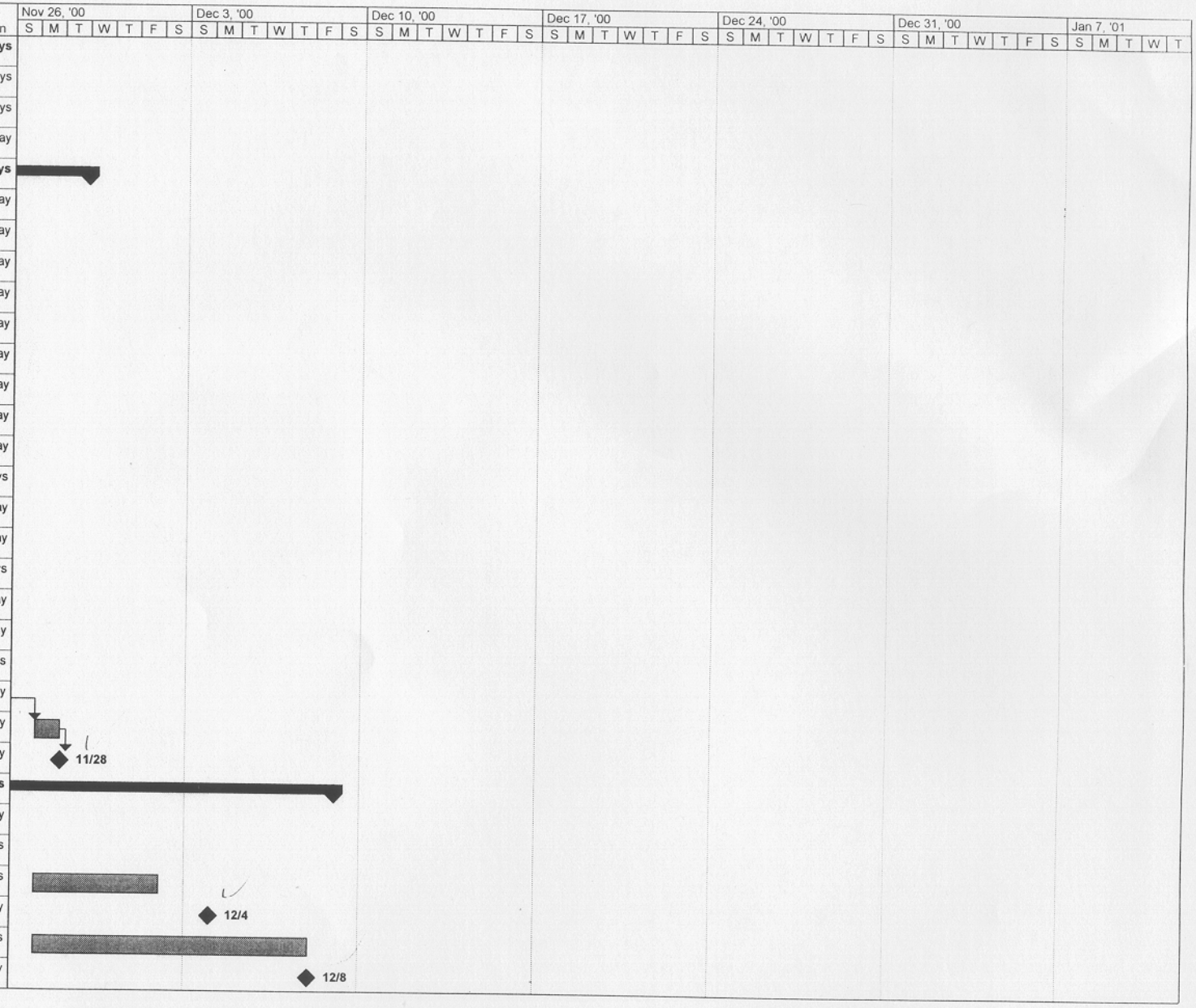
ID	Task Name	Duration
1	**Material Gathering**	**16 days**
2	Component Price Comparison	5 days
3	Purchase Components	10 days
4	Purchase Complete	1 day
5	**Assembly**	**27 days**
6	Attach LCD	1 day
7	Program LCD Module	1 day
8	Attach Scroll Button	1 day
9	Program Scroll Module	1 day
10	Attach Temp Sensor	1 day
11	Program Temp Module	1 day
12	Attach Humidity Sensor	1 day
13	Program Humidity Module	1 day
14	Attach Compass	1 day
15	Program Compass Module	5 days
16	Attach Pressure Sensor	1 day
17	Program Pressure Module	1 day
18	Build Wind Sensor	6 days
19	Attach Wind Sensor	1 day
20	Program Wind Module	1 day
21	Program Meteorological Processing	24 days
22	Modules Complete	1 day
23	Program Main Routine	1 day
24	Assembly Complete	1 day
25	**Evaluation**	**25 days**
26	Demonstration	1 day
27	Progress Report	6 days
28	Prepare Presentation	5 days
29	Presentation	1 day
30	Prepare Final Report	9 days
31	Final Report	1 day



Project: Outdoorsman WeatherMate Pro
Team: Michael Baj & Eric Chambers
Date: Tue 10/10/00

Task
Split
Progress
Milestone
Summary
Rolled Up Task
Rolled Up Split
Rolled Up Milestone
Rolled Up Progress
External Tasks
Project Summary

ID	Task Name	Duration
1	**Material Gathering**	**16 days**
2	Component Price Comparison	5 days
3	Purchase Components	10 days
4	Purchase Complete	1 day
5	**Assembly**	**27 days**
6	Attach LCD	1 day
7	Program LCD Module	1 day
8	Attach Scroll Button	1 day
9	Program Scroll Module	1 day
10	Attach Temp Sensor	1 day
11	Program Temp Module	1 day
12	Attach Humidity Sensor	1 day
13	Program Humidity Module	1 day
14	Attach Compass	1 day
15	Program Compass Module	5 days
16	Attach Pressure Sensor	1 day
17	Program Pressure Module	1 day
18	Build Wind Sensor	6 days
19	Attach Wind Sensor	1 day
20	Program Wind Module	1 day
21	Program Meteorological Processing	24 days
22	Modules Complete	1 day
23	Program Main Routine	1 day
24	Assembly Complete	1 day
25	**Evaluation**	**25 days**
26	Demonstration	1 day
27	Progress Report	6 days
28	Prepare Presentation	5 days
29	Presentation	1 day
30	Prepare Final Report	9 days
31	Final Report	1 day



Project: Outdoorsman WeatherMate Pro
Team: Michael Baj & Eric Chambers
Date: Tue 10/10/00

Task
Split
Progress
Milestone
Summary
Rolled Up Task
Rolled Up Split
Rolled Up Milestone
Rolled Up Progress
External Tasks
Project Summary