A MICRO-COMPUTER CONTROLLED CALORIE MONITOR FOR HUMAN POWERED VEHICLES

GARY G. SIEGMUND

Bachelor of Science in Electrical Engineering

Cleveland State University

June, 1980

Submitted in partial fulfillment of requirements for the degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

At the

CLEVELAND STATE UNIVERSITY

August, 2005

Lee Anne

This thesis is dedicated to my wife. Not only did she participate in several of the tests, her encouragement and pride in my work were treasured gifts from a soulmate.

Acknowledgments

First I would like to sincerely thank Dr. Dan Simon for his inspiration as an exceptional teacher and his support as the thesis advisor. Secondly, I wish to recognize the following faculty: Dr. Pong Chu, Dr. Vijaya Konangi, Dr. William Schultz, Dr. Ana Stankovic, and Dr. Wenbing Zhao. In different ways I admire each of them and I value what I learned from their lectures. Thirdly, I want to thank the members of the thesis committee whose interest and support were greatly appreciated. Finally, I want to thank my fellow students and the faculty and staff of the Electrical and Computer Engineering Department. A friend is a chance encounter nourished with sincerity.

A MICRO-COMPUTER CONTROLLED
CALORIE MONITOR FOR HUMAN POWERED VEHICLES

GARY G. SIEGMUND

ABSTRACT

The amount of calories expended during an exercise interval is a primary indication of training intensity and effectiveness. If the exercise apparatus is a human powered vehicle, the potential exists for effectively measuring several parameters of the training session. Unfortunately, existing monitoring systems are either invasive or inaccurate. This paper describes the design of an energy monitoring system that displays the cumulative calories expended during a riding interval, without requiring attachments to the body of the participant, or modification of the vehicle.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

CONCEPTS OF EXERCISE MEASUREMENT

Initially this opening chapter gives a quick overview of the material covered in the thesis. Secondly, the relationship to previous work is described, with a focus on the qualities that are similar and the aspects that are unique. Thirdly, a foundation is laid to support a bridge between the principles of exercise and parameters that are measurable.

1.1 Overview of Thesis

This thesis describes an energy monitoring system for a Human Powered Vehicle (HPV). Human powered transportation is highly attractive as it is very supportive of the environment, while providing cardiovascular conditioning for the participant. A need that exists is the ability to monitor the extent of cardiovascular conditioning. The contribution of this thesis is a device that measures the intensity and effectiveness of exercise, by determining the amount of calories consumed during an exercise session.

Since direct calorimetry in an open environment is not feasible, an alternative measurement process was required. Chapter 1 looks at muscle function in the human

body. The thesis analyzes the fundamentals of muscle activity and the body's energy systems. The physiology of muscle contraction is briefly covered with an emphasis on the relationship of biological activity to relevant measurable quantities.

Chapter 2 looks at sensors. The measurement parameters the system requires form a definition for the number and type of measurement sensors. One of the particularly challenging areas was the need for an air speed sensor. While using a hot-wire anemometer design was attractive, these devices typically have the disadvantage of relatively high power consumption. A requirement was the design of a hot-wire anemometer that was capable of operating under low power.

The proper operation of individual elements requires coordination at the system level. An embedded processor meets this requirement as it combines extensive I/O capability with processing efficiency. Chapter 3 describes the system electrical design including sensors, channel amplifiers, embedded processor, and LCD display. The thesis includes the construction of a prototype. This chapter also describes the prototype design and fabrication, including images of the completed system. Finally the system software is described including data flow from the sensor inputs, the progressive calculation of variables, and the display of the results.

Chapter 4 presents the data from a variety of tests designed to validate the system operation. Of particular importance are the isolated energy tests which individually

calculate the energy from three different measurement parameters. The chapter finishes with a concluding section and a section that describes future research.

1.2 Related Work

Any measuring instrument must have traceability: traceability of the instrument calibration to a standards bureau, and traceability of the measurement algorithms to accepted procedures in the field of study. To validate this exercise monitoring system, the thesis has relied on information available from the American College of Sports Medicine (ACSM). This institution, founded in 1954, is focused on sports medicine and exercise science, and is recognized around the world as an authority in this field.

To provide a frame of reference for the measurement of the caloric consumption while riding a bicycle, Table I lists the caloric expenditure based on velocity and body weight, over a 30 minute time interval. This table was created from a compendium of physical activities listing the ratio of work metabolic rate to resting metabolic rate. The values shown are the calories expended or energy input [ame 01b].

Bicycle Velocity	Body Weight of Individual			
	100 #	150 #	200 #	250 #
< 10 mph	95 Kcal	143 Kcal	190 Kcal	238 Kcal
10 - 11.9 mph	143 Kcal	214 Kcal	286 Kcal	357 Kcal
12 - 13.9 mph	190 Kcal	286 Kcal	381 Kcal	476 Kcal
14 - 15.9 mph	238 Kcal	357 Kcal	476 Kcal	595 Kcal
15 - 19 mph	286 Kcal	428 Kcal	571 Kcal	714 Kcal
> 20 mph	381 Kcal	571 Kcal	762 Kcal	952 Kcal

Table I: Caloric expenditure

In addition to providing activity measurement data, ACSM also has analyzed the physiology of physical activity to establish guidelines for predicting caloric expenditure

for different repetitive motions. This thesis uses one of these metabolic equations for the calculation of caloric consumption [ame 00].

One of the sensor requirements is the ability to measure air speed. Since the hot-wire technology was selected as the measurement method for air speed monitoring, an amount of heat flow analysis was employed to validate the process. Conduction and convection analysis is covered in significant detail in [inc 96].

A significant breakthrough has been the design of integrated accelerometers. The use of micro-machined devices in this thesis has greatly reduced the power requirement, the cost, and the size of the sensor. Analog Devices has extensive application information on these devices [kit 95], [shu 95], [ana 04].

Table II lists several personal monitoring devices that are commercially available. Some of these are mountable on a bicycle to perform as bicycle computers. The measurement method employed by these devices is different than that employed by the thesis and falls into the following categories.

1) Bicycle mounted device that determines calories by measuring heart rate.
2) Non-bicycle mounted device that determines calories by measuring foot acceleration.
3) Non-bicycle mounted device that determines calories by measuring GPS displacement.

Device	Measurement Method		
	Heart Rate	GPS	Foot Acceleration
Polar S720i	X		
Garmin Forerunner 201		X	
Suunto T6			X

Table II: Personal monitoring devices

There is another commercial product that is more similar to the thesis. This device is a torque measuring instrument incorporated in the rear hub of a bicycle. The device calculates kilojoules as the energy output, based on the measured torque and the wheel speed. To determine energy input or calories consumed, the company assumes a human body efficiency of 25% and directly translates kilojoules to "food" calories at a 1:1 ratio. The product cost when built into a wheel is about $1200.

The design of this thesis is unique and an improvement over the commercial products in the following ways.

1) Designs based on heart rate require wearing a chest strap.
2) Designs based on heart rate can be inaccurate because of the delay in heart rate response to exercise.
3) The thesis design is the only one based on air flow and acceleration.
4) The thesis design does not require a rear hub replacement.
5) The thesis design is potentially much less expensive to manufacture.

Bicycle computers are available for $20-$50. The cost of the temperature sensors and the accelerometer is less than $10. Adding the cost of these components with mounting issues and the bicycle computer might cost $100-$200.

1.3 Exercise Physiology

All physical work performed by the human body comes from the contraction of muscle fiber. The energy for this process comes from the conversion of adenosine triphosphate (ATP) into adenosine diphosphate (ADP). The adenosine triphosphate molecule is shown in Figure 1 and the adenosine diphosphate molecule is shown in Figure 2 [ame 01b].

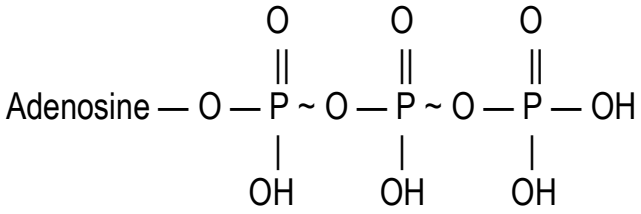


Figure 1: Adenosine triphosphate molecule

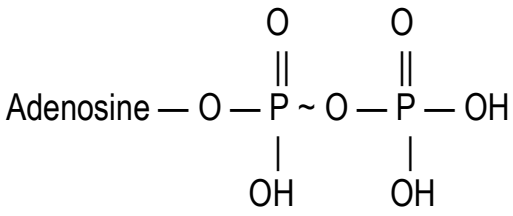


Figure 2: Adenosine diphosphate molecule

There is a release of energy if one phosphate group is cleaved from adenosine triphosphate by means of ATP hydrolysis. The result of this reaction is adenosine diphosphate (ADP), inorganic phosphate (Pi), and an amount of energy in the form of muscle contraction. All muscle activity in the human body depends on the production and availability of ATP in the muscle fiber. The reaction is shown below [ame 01b].

$$ATP + H_2O \rightarrow ADP + P + energy$$

The first few seconds of exercise use the residual ATP that is available locally in the muscle. From that point on however the ATP must be re-synthesized continuously utilizing three different metabolic processes [ame 01a], [gau 89], [cos 03].

1) Creatine-phosphate (CP) System
2) Anaerobic Glycolysis (Lactic Acid System)
3) Oxygen (Aerobic) System

Since the residual ATP is exhausted in a few seconds, the first process is able to immediately utilize creatine phosphate to re-phosphorylate the ADP back into ATP. This only requires one enzymatic process and is rapid because the creatine phosphate is already available in the cells. Since oxygen is not required, this process is classified as anaerobic [ame 01b].

The second process also happens quickly and is called anaerobic glocolysis. This is an enzyme catalyzed conversion of carbohydrate in the form of glycogen or glucose to produce ATP through substrate phosphorylation. If glucose is the substrate, two ATP molecules are produced and if glycogen is the substrate then three ATP molecules are produced. Again since oxygen is not involved, this process is also classified as anaerobic. Like the creatine phosphate process, the duration of the rapid glycolysis process is limited [ame 01b].

The last process is much more involved but is the energy source for the majority of the exercise interval. This process is aerobic in that it does utilize oxygen to implement oxidative phosphorylation which produces ATP and water. Aerobic oxidation is also more useful as it can utilize substrates of carbohydrates, fat, or even protein to produce ATP.

Aerobic oxidation is achieved by a sequence of two complex metabolic processes. The first of these is called the Krebs cycle which essentially removes hydrogen atoms from the reactants. The electrons from these atoms form a chain of cytochromes in the second process where the electrons are transported from molecule to molecule in the mitochondria. The energy from this process can re-phosphorylate ADP back into ATP while the hydrogen combines with oxygen to form water [ame 01b], [gau 89].

The initial utilization of residual ATP and the three processes that produce ATP are not isolated energy sources but overlap to a significant degree. While there is overlap, each of the energy sources is dominant over the following time intervals [ame 01b].

< 5 seconds	-	usage of existing ATP
5 seconds to 10 seconds	-	creatine phosphate production of ATP
10 seconds to 100 seconds	-	rapid glycolysis production of ATP
> 100 seconds	-	oxidative production of ATP

From this data it is clear that even if the duration of exercise is only 20 minutes, the vast majority of the energy produced is from aerobic metabolism.

> "In intense activities lasting 1.5 to 2 minutes, the ATP-CP and lactic acid energy systems generate approximately 50% of the energy, while aerobic metabolism supplies the remainder. A distance runner, on the other hand, derives essentially 98% of his energy from aerobic metabolism during a 50-minute training run." [ame 01b, 142]

This reliance on oxygen for longer training intervals is the basis for predicting the amount of kilocalories consumed during exercise.

1.4 Exercise Calorimetry

The goal of this thesis is to determine the caloric expenditure for a given amount of physical work performed. There are 3 techniques presented here that could be used to measure the caloric expenditure from exercise.

The most basic procedure is direct calorimetry. For an exercising individual this measurement requires enclosing the person and exercise apparatus in a chamber. By carefully monitoring the temperature, the metabolic rate in kilocalories can be precisely determined. Since this process places high demands on the measuring instrumentation and is expensive, it is not frequently used.

A more common process uses indirect calorimetry to determine the caloric expenditure by measuring the oxygen consumption during exercise. This typically uses open-circuit spirometry to measure the volume of inspired oxygen and the volume of expired oxygen and expired CO_2. These measurements can then be used with established relationships to estimate the metabolic rate in kilocalories. While this technique is

commonly used in the lab, and portable volume exchange monitors allow measurements in the field, it is invasive and not practical for monitoring casual exercise.

There is another form of indirect calorimetry that uses the relationship of the exercise power level in Watts to oxygen uptake. This relationship of oxygen consumption to work output has been tested over a large and broad sampling of the adult population. Therefore if the weight of the individual is known and the work intensity is measured, it is possible to estimate the volume of oxygen exchange to a fairly high degree of accuracy. From the level of oxygen uptake the calories expended can be predicted. This is the technique utilized in this thesis.

Since oxygen is part of the fuel for muscle activity during steady state aerobic exercise, the amount of oxygen consumed (oxygen uptake) is proportional to the work produced. The respiratory quotient (RQ) is the ratio of CO_2 to O_2 in the air expelled and varies depending on what fuel is being utilized. Table III compares the respiratory quotient for different fuels and the corresponding energy expended per liter of O_2 exchange [ame 00]. Since the RQ is usually not known, the value of 5 kcal per liter of O_2 is typically used.

Fuel	**RQ**	**Kcal/liter O_2**
Fat	0.7	4.69
Protein	0.8	NA
Carbohydrate	1	5.05

Table III: Respiratory quotient for different fuels

The American College of Sports Medicine has derived a number of metabolic equations that apply to different types of physical activity. These equations predict the volume of O_2 exchange (VO_2) as a function of the power level of the exercise, the mass of the individual, and the length of the exercise period. The equations are only appropriate for long term exercise intervals where the exercise is predominately aerobic. As was previously shown, this is the category of exercise where the energy comes from aerobic metabolism and there is a strong correlation between VO_2 and the energy produced. The equation for leg ergometry is given below and is suitable for riding a bicycle [ame 00].

$$VO_2 = \frac{(10.8 \cdot W)}{M} + 7.0$$

VO_2 volume of O2 exchange expressed as $mL \cdot kg^{-1} \cdot min^{-1}$

W = power in Watts

M = mass of individual in kg

In this formula the 10.8 term is used for scaling, and the 7.0 term represents the resting VO_2 level and the cost of unloaded leg movement at 50-60 rpm. The conversion of this formula from VO_2 to the energy in kcal is shown below [ame 00].

minus	3.5	delete resting VO_2
times	M	scale to the mass of the individual
times	$\frac{1 \text{ minute}}{60 \text{ seconds}}$	convert from minutes to seconds
times	$\frac{1 \text{ liter}}{1000 \text{ mL}}$	convert from ml to liters

times	$\frac{5 \text{ kcal}}{\text{Liter } O_2}$	convert from VO_2 to kcal
times	T	duration in seconds

resulting in

$$\text{kcal} = \left[\frac{\left(\frac{10.8 \cdot W}{M} + 3.5 \right) \cdot M}{12,000} \right] \cdot T$$

In this formula the input variables are the mass of the rider in kilograms, the power level in Watts, and the time duration of the exercise interval in seconds. The output variable is the energy expenditure above resting energy and is expressed in kilocalories. To calculate the energy expended during exercise, this energy monitor will need sensors that can measure the power level in Watts of a person riding a bicycle. This instrumentation methodology is discussed in the next chapter.

CHAPTER II

SENSOR RESEARCH AND APPLICATION

The ability to evaluate a process requires the capability of measuring individual aspects of that process. Choosing the proper measurement parameters can have a significant influence on the system complexity and the overall measurement accuracy. This chapter looks at parameter selection and analyzes sensor design and integration.

2.1 Measurement Basis

The Watt is a unit of power that usually represents electrical power but can also be used to represent the power level in mechanical work. The most direct way to measure mechanical power is to measure the force moving an object and the velocity of that object. For example, the power level of a person riding a bicycle could be measured by sensing the force exerted on the pedals and the cadence or pedaling frequency. Another method would be measuring the tension in the chain or the torque in the drive hub. However, all of these direct techniques violate the non-invasive design requirement. Modifying the bicycle by adding strain gauges to the drive mechanics is not consistent with the stated design goals of this thesis.

An indirect way to determine the drive force can be found by examining the quantities that oppose motion. There is a force required to accelerate the mass, to oppose gravity, and to oppose friction. The mass of the rider and vehicle at rest will wish to stay at rest and will require a force acting on the mass to set it in motion. This force can be determined indirectly by measuring the acceleration of the vehicle. Similarly, if the vehicle is climbing an incline there is a force required to elevate the mass. This force can again be determined indirectly by measuring the acceleration of gravity at that angle of incline. Thirdly, there is friction opposing forward motion. The force opposing friction can be determined by inclusion of all friction components and a measurement of the velocity of the vehicle. In summary then there are three forces to be considered in the energy analysis of the vehicle and rider: the force required to accelerate the mass, the force required to oppose gravity, and the force required to oppose friction. All three of these forces can be determined indirectly, and an analysis of the measurement process follows.

The force causing acceleration is given by [sea 74]

$$F=ma$$

$m = kg$

$a = ms^{-2}$

$F = newtons$

If the mass of the vehicle is known and the mass of the rider is known, these quantities can simply be included in subsequent calculations. The acceleration of the vehicle can be determined by measuring the change in velocity of the vehicle. However, a more direct method is the measurement of the acceleration by an accelerometer. An

accelerometer is a sensing element that determines the acceleration by measuring the displacement of a suspended mass. If the accelerometer is calibrated in meters and seconds, then the accelerometer reading multiplied by the mass of the loaded vehicle is the accelerating force in newtons.

The acceleration of gravity g is 9.8 ms^{-2}. A 1 kg mass has a downward force due to gravity of 9.8 newtons. An exercising individual must exert a force opposing the force of gravity. If the rider were climbing a 90 degree hill this force would be 9.8 newtons for every kg of vehicle weight. However for more moderate slopes the acceleration of gravity acting on the vehicle is $g \sin \theta$ where θ is the angle of the hill. If the accelerometer axis of measurement is parallel to the slope of the hill, and if the accelerometer is calibrated in meters and seconds, then the accelerometer output will equal $g \sin \theta$. As a result, the same accelerometer that is used to calculate the force in newtons due to acceleration of the vehicle, can also be used to calculate the force opposing gravity.

The third force that must be determined is the force opposing friction. Friction is the greatest energy drain over moderately level terrain and the major component is the friction due to air displacement. A person riding a bicycle displaces a significant amount of air and the air shadow is not aerodynamic. Well over a hundred Watts of power is lost at higher velocities due to air displacement. Since the air friction is proportional to the square of the velocity, it is possible to indirectly calculate the drag force due to air friction by measuring the vehicle velocity. However, the surface velocity is only related

to air friction if the air is calm. To properly calculate air friction, the air speed of the vehicle must be known. To measure air speed what is required is an air speed sensor or anemometer.

Additional sources of friction include tire rolling resistance and drivetrain friction. The tire rolling friction for modern high pressure tires is almost zero. Since the air friction signature for the vehicle will be determined by a measurement of the drag force at touring velocities, whatever tire rolling resistance that does exist will be included at this velocity.

The friction due to the drivetrain is fairly small. It is largely independent of air speed and doesn't have any significant air friction component. The source of the friction is the sliding drive members and bearings, and the viscous flow of lubricating fluids. Since the viscosity of lubricants varies with temperature, no load tests were performed at 0 °C and 25 °C in an empirical determination of this friction component. The test configuration is described more fully in Chapter 4.

To summarize, there are 3 major forces that are involved in the energy analysis of a person exercising on a bicycle. The force to accelerate the mass and the force to oppose gravity can be measured indirectly through the use of an accelerometer. An overview of the types of accelerometers and the selection, design, and integration of the acceleration sensor is described in Section 2.3. The third force that opposes friction can be measured indirectly by determining the air speed of the vehicle. This requires the design of an air

speed sensor or anemometer. The anemometer design is a major part of this thesis, and is discussed in the next section.

2.2 Air Velocity Sensor

The friction due to air displacement is the largest component of friction and usually the largest energy drain in a human powered vehicle. Since the drag due to air friction is directly related to air speed, the force against friction can be indirectly determined by a measurement of the air speed. This type of measurement requires an air speed sensor or anemometer. This section details the design of an anemometer that combines measurement accuracy with very low power operation.

2.2.1 Anemometer Design Analysis

Air flow measurements are significant throughout science for everything from sensing flow rates in space shuttle fuel cells to maximizing the efficiency of growing mushrooms [mar 02]. Air flow can be measured using light, as in Laser Doppler Anemometry which measures velocity induced phase shifts [bev 97]. Ultrasonic anemometers are based on sound and measure the change in the speed of sound as a function of air velocity [sto 02]. Other anemometers measure pressure as in the Pitot tube, or the torque induced on rotating cups [kri 99].

Anemometers can also be based on temperature such as the calorimetric flow meter. This device measures the amount of energy required to heat a flowing substance [hsi 95]. Another temperature based flow sensor measures the cooling effect of air on a heated

surface. Since faster moving air will cool a heated surface more than slower moving air, the amount of cooling is a function of the air velocity. To utilize this technique, the temperature of the heated surface must be known as well as the ambient temperature of the air. Accordingly, this system uses two temperature sensors. One, called the hot sensor is heated, and must accurately measure the cooling as a result of the air flow. The other sensor is called the cold sensor, and it measures the ambient air temperature. In effect, the system measures the cooling of the hot sensor from the frame of reference of the cold sensor. This category of air velocity measurement is called hot-wire anemometry and is the technique utilized in this thesis.

The term hot wire comes from the typical use of a thin heated wire as the hot sensor. Since the wire has a known temperature coefficient of resistance, the temperature of the wire can be determined by measuring the voltage across and the current through the wire. In addition, the power in the wire is a function of the voltage and current, and can be used to control the temperature of the wire.

Hot wire anemometers can be operated in a number of modes with advantages and disadvantages of each. Where the hot sensor is a section of resistance wire, the element can have a constant controlled voltage and the current through the wire is a function of temperature. The element can also be operated at constant current, constant power, or even variable controlled temperature [lee 97]. In all of these modes the wire will change temperature with different air flow rates. This has the disadvantage of slower response time since it will take a finite amount of time for the wire to change temperature.

To increase the speed of response, another mode may be used called constant temperature. In this mode the temperature of the sensor above ambient is kept constant by means of closed loop regulation. The measurement parameter for the air flow calculation is the amount of power in the sensor. Higher air speeds will require higher power in the sensor to maintain the constant temperature difference between the sensors. Since there is no temperature change in the sensor, the speed of response can be quite high [keg 00].

This thesis proposes a dual mode sensor to facilitate a wider range of operation. For low to medium air velocities, the hot sensor is operated at constant temperature to provide higher accuracy and speed of response. At higher flow rates the hot sensor is operated at constant power. This prevents an increase in power requirements at high flow rates while still providing accurate measurements. In addition, since the sensor temperature change is significantly faster at higher flow rates, the speed of response is still adequate when operated in this mode.

There are several different types of sensing elements that can be used in the hot wire class of air flow instruments. While the simplest is just a wire, a number of other devices could be used. Thermisters have a higher coefficient of resistivity and can be effectively used to measure very low flow rates [mar 71]. Thermocouples are usable, but require external heating and cold junction compensation. Semiconductor PN junctions have a well established temperature relationship due to carrier mobility and can be designed for

self-heating. In addition, PN junctions can be incorporated in an integrated device where a significant reduction in the overall complexity of the system is achieved. Because these advantages are important to this application, this system will utilize integrated PN junction sensing devices.

The device selected for this design is the LM335 in a TO-46 package [nat 00]. The LM335 is an electronic zener device that regulates to a constant voltage when fed from a reverse polarity current source. This makes the device capable of self-heating since the power can be controlled by varying the reverse current. The reverse voltage is proportional to temperature at the rate of 10 mV/°K. For example, at a temperature of 25 °C the voltage across the device is 2.98 V. The reverse current has an absolute maximum of 15 mA and a recommended operational maximum of 10 mA. This results in a maximum self-heating power of ~30 mW.

Because an integrated package often has complex package geometry and relatively low thermal resistance to air, it requires careful application design. If the entire package is exposed to the air flow, complex patterns of air movement could result in measurement inaccuracies. In addition, the low thermal resistance increases the power requirements for heating the device. For these reasons this thesis focused on exposing only a fraction of the package to the air flow, and achieving laminar air flow over the device. This design approach increased the ratio of heat loss to air flow to the heat loss through the package mounting to ambient.

To increase the thermal resistance to ambient, the sensor package is enclosed in insulating material. The material used for insulation and device mounting is cork. This was chosen for its very low thermal conductivity of 0.039 W/m°K, and for physical ruggedness [inc 96]. While styrofoam has a higher thermal resistivity, the exposed surface would be subject to damage and deterioration. The cork was obtained from a local winery, slit lengthwise, and the flat surfaces machine ground for smoothness. The cork halves were clamped together and mounted in a lathe chuck. A high speed miniature grinder was attached to the lathe carriage. With this equipment it was possible to machine the soft material to the proper dimensions. The two halves were joined with the LM335 sensor using a bead of epoxy adhesive around the circular portion of the sensor package to seal out water. The cold sensor was mounted in a similar fashion. The sensor wires were extended using constantan wire to greatly reduce the heat flow through the leads.

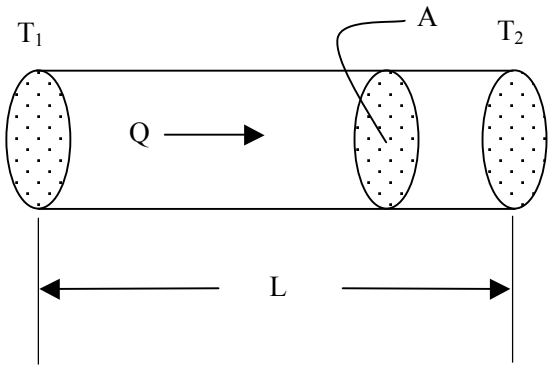
The major concern with using an integrated package in place of a hot wire sensor is the amount of power required to heat the sensor. This design analysis will investigate whether the LM335 can be used as a sensing device with adequate accuracy over the anticipated range of air velocity. The analysis is based on two types of calculations. The first is thermal conduction through a substance. In this analysis the heat conduction is a function of the temperature difference, the dimensions of the substance, and the coefficient of heat conduction. The second is forced air thermal convection. This calculation is based on a number of factors such as the dimensions and geometry of the interface, the temperature difference, the air flow, and the properties of air at the

temperature of operation. The basis of the calculations is presented followed by the calculation of values for the case in study.

2.2.2 Thermal Conduction Analysis

The equations for thermal conduction through a substance may be used to calculate the heat transfer through the length of a material [sco 74]. This analysis technique will be used to determine the heat loss through the leads of the device. This process is shown in Figure 3.

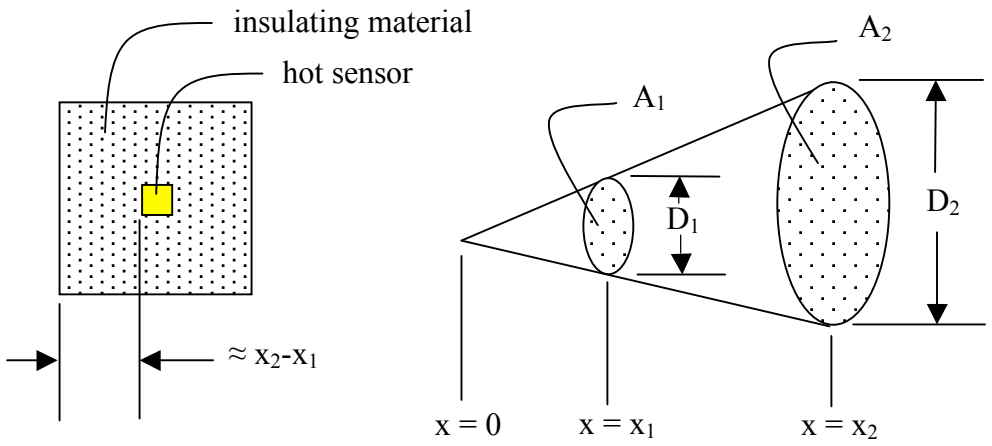
$$Q = k \cdot A \cdot \frac{T1 - T2}{L}$$



Q = heat transfer in Watts
k = thermal conductivity
A = area
$T_1 - T_2$ = temperature difference
L = length of material

Figure 3: Thermal conductivity through a substance

To investigate the heat loss from the sensor through the insulating material, a one-dimensional approximation as shown in Figure 4 can be used [inc 96].



A = area
D = diameter
x = position of slice relative to the vortex of cone

Figure 4: Thermal conductivity through a conical section

This approach uses a circular conical section to simplify the thermal analysis. The area of the section at x_1 equals the surface area of the hot sensor. The area of the section at the right side of the cone at x_2 equals the outer surface of the insulating material enclosing the hot sensor. For this analysis, the distance from x_1 to x_2 equals the nominal thickness of the insulating material. The point where $x = 0$, and the values of x_1 and x_2 do not represent physical positions in the insulating material, but represent positions in the model.

Three equations represent the relationship of these variables

$D_1 = a_1 x_1$ where D_1 = diameter of cone at x_1

$D_2 = a_1 x_2$ a_1 = constant for slope of cone

$x_2 = x_1 +$ (thickness of insulating material)

These equations are solved simultaneously taking the known values of D_1 and D_2 to solve for x_1, x_2, and a_1. To determine the heat transfer, the heat flow equation in the following form is used [inc 96].

$$q_x = k \cdot A \cdot \frac{dt}{dx}$$

where q_x = heat flow in Watts

k = thermal conductivity of material W/m°K

A = area m^2

dt = incremental temperature difference

dx = incremental position

This formula is rearranged, integrated, and solved for q_x yielding [inc 96]

$$q_x = \pi \cdot a_1^2 \cdot k \frac{(T_2 - T_1)}{4 \cdot \left\{ \frac{1}{x_1} - \frac{1}{x_2} \right\}}$$

By inserting values for the dimensions of the sensor configuration, the temperatures, and the thermal conductivity of the insulating material, the value of heat flow in Watts is determined.

2.2.3 Thermal Convection Analysis

For calculating forced air thermal convection, the analysis is based on a flat plate in parallel flow. In this calculation, the exposed surface of the thermal sensor is represented by a flat rectangular plate of dimensions L parallel to air flow, and a dimension W across the air flow. For this calculation, laminar air flow is assumed and the calculation uses the fluid properties of air at 300 °K [inc 96].

$k = 0.0263$ W/m°K	thermal conductivity
$v = 0.00001589$ m^2/s	kinematic viscosity
$a = 0.0000225$ m^2/s	thermal diffusivity

The following additional variables and formulas are used for the forced air convection analysis [inc 96].

L = length of plate in meters

W = width of plate in meters

V = flow velocity in m/s

T_1 = air temperature °C

T_2 = plate temperature °C

$Pr = \frac{v}{a}$	Prandtl number
$Re = \frac{V \cdot L}{v}$	Reynolds number
$Nu = 0.664\ Re^{1/2}\ Pr^{1/3}$	Nusselt number
$h = \frac{Nu \cdot k}{L}$	heat transfer coefficient

$$q = h \cdot (\text{plate area}) \cdot (\text{temperature difference})$$

2.2.4 Temperature Tests

A temperature accuracy test was performed on 4 integrated thermal sensors. The purpose of this test was to determine the characteristics of the specific sensor devices used in the prototype. There are 5 significant sensor errors to be considered.

1) Calibration error - this is the measurement error for a given device at a nominal temperature e.g. 25 °C. This error will be corrected by software calibration.
2) Slope error - this is the measurement error for a given device over the full range of temperatures. If significant, this error could be corrected by a value based on temperature.
3) Linearity error - this is the deviance from a linear temperature slope for a given device. If significant, this could be corrected by a software look-up table.
4) Time stability error - this is an initial drift in the first 1000 hours of measurement.
5) Error due to a change in reverse current - there is a small dependency of the output measurement on the reverse current. This will be eliminated by performing all measurements at the same current.

For the temperature test to be accurate, the temperatures of the sensors and the reference thermometer must all be at the same exact temperature. This test utilized a ½" x 2" square block of copper to tightly couple the devices as shown in Figure 5. The 4 sensors and thermometer were thermally attached to the copper block as shown in Figure 6.

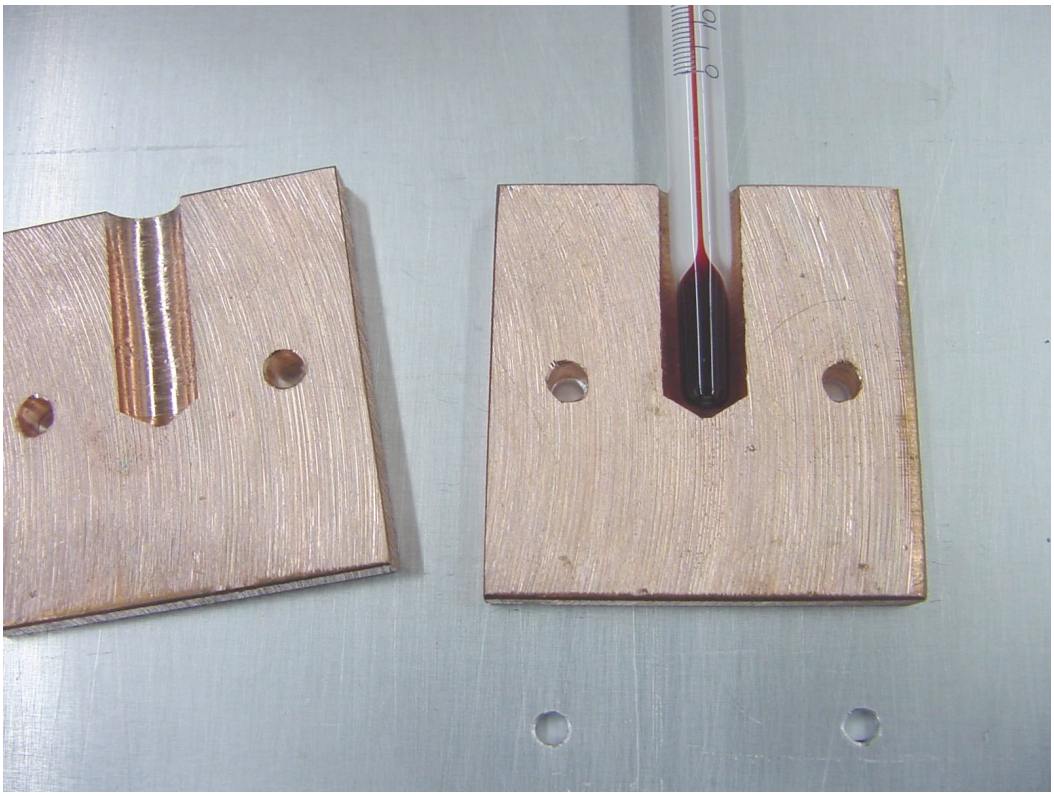


Figure 5: Picture of copper block

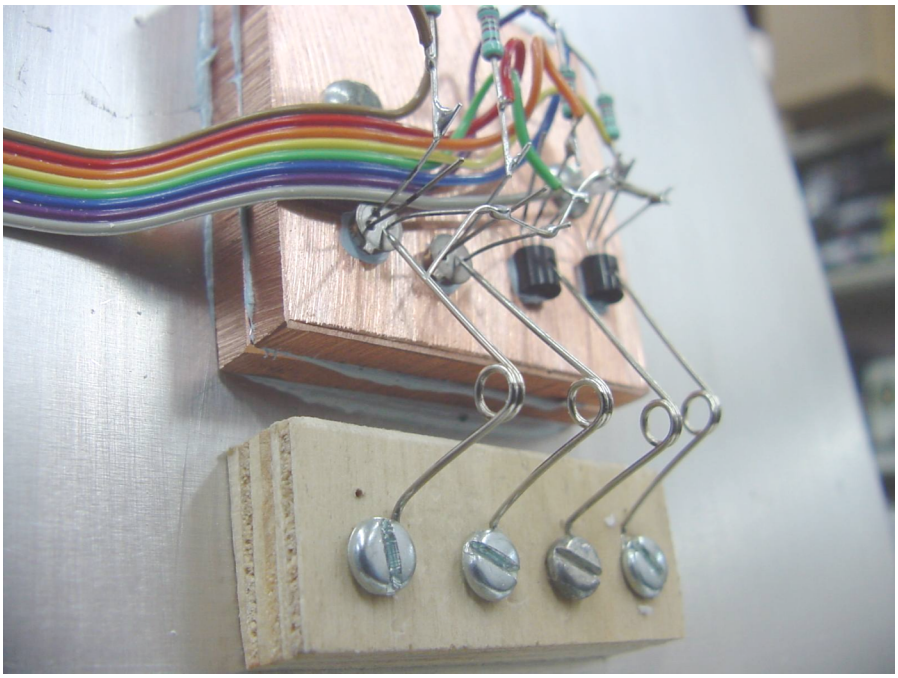


Figure 6: Picture of sensors attached to copper block

The assembly was enclosed in styrofoam and placed in an environmental chamber. The temperatures were recorded at 2 °C increments over a range of 0 °C to 40 °C. Figure 7 is a plot of the temperature readings from the 4 sensors. Figure 8 is from the same raw data with calibration offset values added to correct for the calibration error in each of the devices.

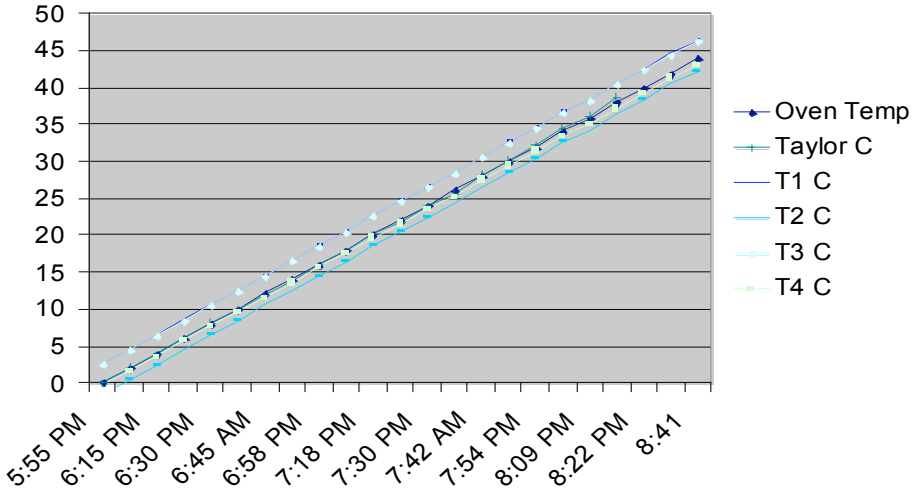


Figure 7: Un-calibrated sensor temperature test

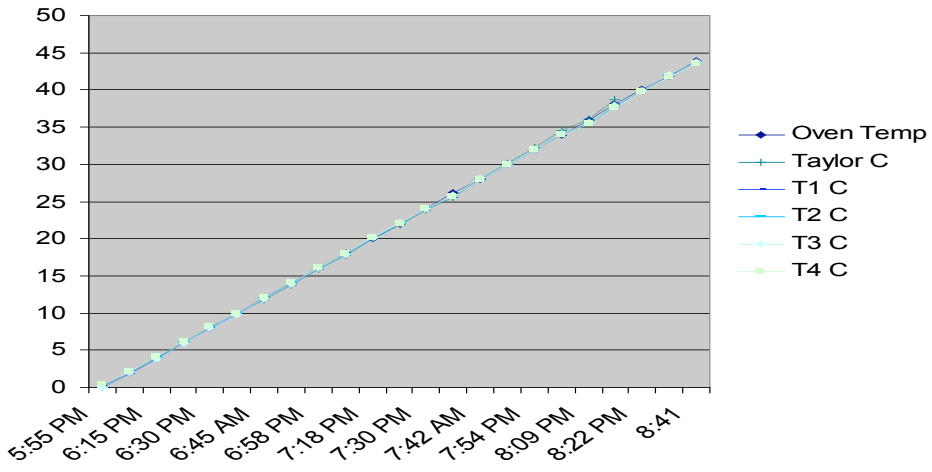


Figure 8: Calibrated sensor temperature test

As can be seen from the data plots, there is no significant linearity error detected in the 4 devices tested. The calibration error is significant but will be corrected in software. The slope error for the two TO-46 devices is very small and no correction is required.

The next test measured the temperature rise under various conditions with constant power in the sensor. For this test the device leads were extended with 36 Ga copper wire to minimize the heat loss through the leads and to focus the test on the heat loss through the insulation. The test was performed with a power input of 30 mW into the sensor with data recorded under 4 different measurement conditions. The data is shown in Table IV which also includes a calculation based on a conical section model of a fully enclosed sensor. Even though the conical section is only an approximation, the calculated rise is very close to the measured data.

Data Source	Condition	Power	Ambient	Measurement	Rise
Mea	Sensor in still air	30mW	293°C	304°C	11°C
Mea	Sensor in cork, end exposed	30mW	293°C	306°C	13°C
Mea	Sensor in cork, end against styrofoam	30mW	293°C	308°C	15°C
Mea	Enclosed in 4" block of styrofoam	30mW	293°C	312°C	19°C
Cal	Sensor fully enclosed in cork, Calculated using circular conical section	30mW	-	-	16°C

Table IV: 30 mW sensor temperature rise test

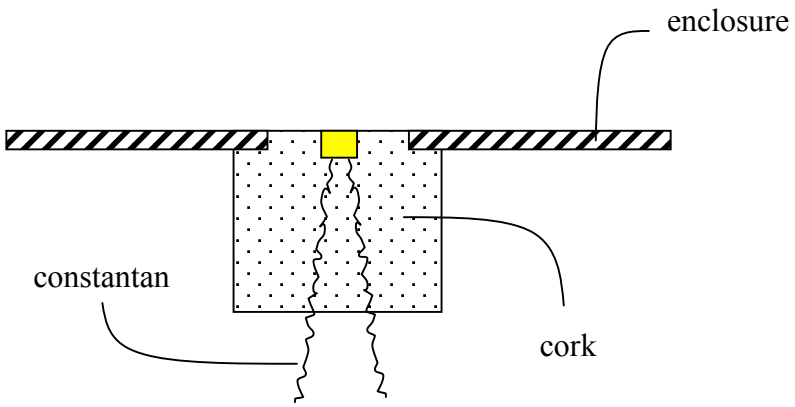


Figure 9: Graphic of sensor mounting

To reduce the heat loss through the leads of the device they were replaced with 30 ga constantan wire (55%Cu, 45% Ni). This material has a thermal conductivity of 23 W/m°K. The calibration terminal was not connected and the two active leads were compressed in a zig-zag pattern to increase the lead length. The mounting configuration for the hot sensor is shown in Figure 9. The cold sensor is mounted in a similar fashion.

The thermal conduction analysis for heat transfer through the mounting insulation and through the device leads may be expressed over a range of temperatures. This is useful to determine the proper temperature difference between the hot sensor and the ambient air temperature. A larger temperature difference leads to greater accuracy but also requires more power. Table V displays the power lost through the insulator, the device leads, and the total power lost to ambient.

Measuring air velocity depends on a measurement of the heat flow through convection to ambient air. Using a calculation based on a flat plate in parallel flow, Table VI lists the power lost through convection. The range of velocities is from 1 to 10 m/s and the range of temperature difference between the hot sensor and ambient is from 1 to 10 °C. A key aspect of hot-wire anemometer design is the operation of the hot sensor. In the constant temperature mode, this sensor is regulated to a fixed temperature above ambient. A lower temperature above ambient requires less power to heat the sensor. A higher temperature usually means higher accuracy. Based on the data in Table V and Table VI, a temperature of 5 °C above ambient was chosen for the hot sensor.

Temperature Difference	Mounting Insulator	Device Leads	Total Power
1°C	1.9 mW	0.1 mW	2.0 mW
2°C	3.7 mW	0.2 mW	3.9 mW
3°C	5.6 mW	0.3 mW	5.9 mW
4°C	7.5 mW	0.4 mW	7.9 mW
5°C	9.4 mW	0.5 mW	9.8 mW
6°C	11.2 mW	0.6 mW	11.8 mW
7°C	13.1 mW	0.6 mW	13.7 mW
8°C	15.0 mW	0.7 mW	15.7 mW
9°C	16.8 mW	0.8 mW	17.7 mW
10°C	18.7 mW	0.9 mW	19.6 mW

Table V: Power flow through insulator

Velocity m/s	Velocity mph	Power at 1°C Δ	Power at 2°C Δ	Power at 3°C Δ	Power at 4°C Δ	Power at 5°C Δ	Power at 6°C Δ	Power at 7°C Δ	Power at 8°C Δ	Power at 9°C Δ	Power at 10°C Δ
1	2.2	1.1	2.1	3.2	4.2	5.3	6.4	7.4	8.5	9.6	10.6
2	4.5	1.5	3.0	4.5	6.0	7.5	9.0	10.5	12.0	13.5	15.0
3	6.7	1.8	3.7	5.5	7.4	9.2	11.0	12.9	14.7	16.6	18.4
4	8.9	2.1	4.2	6.4	8.5	10.6	12.7	14.9	17.0	19.1	21.2
5	11.2	2.4	4.7	7.1	9.5	11.9	14.2	16.6	19.0	21.4	23.7
6	13.4	2.6	5.2	7.8	10.4	13.0	15.6	18.2	20.8	23.4	26.0
7	15.7	2.8	5.6	8.4	11.2	14.0	16.9	19.7	22.5	25.3	28.1
8	17.9	3.0	6.0	9.0	12.0	15.0	18.0	21.0	24.0	27.0	30.0
9	20.1	3.2	6.4	9.6	12.7	15.9	19.1	22.3	25.5	28.7	31.9
10	22.4	3.4	6.7	10.1	13.4	16.8	20.2	23.5	26.9	30.2	33.6

Table VI: Power flow through convection

2.2.5 Static Air Flow Tests

This anemometer is a two mode design, constant temperature when the hot sensor is below the maximum power the system is able to deliver, and constant power above that point. The measurement parameter for this anemometer is power when operated in the constant temperature mode, and the temperature difference between the hot sensor and the 5 degree set point when operated in the constant power mode. These two functions do not have the same slope. In addition, even when the system is operated within one of the modes the measurement output is non-linear. For this reason, a software look-up table was utilized to provide accurate air speed measurement from the input data [jor 96]. The software driver was designed to create a single pointer for the look-up table, based on either the power in the hot sensor or the temperature difference. The range of these pointer values is between 0 and 91. The next tests were designed to measure the value of this table pointer when operated under various air flow conditions.

First a series of static tests were performed on the completed anemometer. The test configuration included a commercial anemometer for air speed reference, an industrial vacuum cleaner for air flow, and a powerstat to adjust the air flow. The software driver for these tests included a sensor calibration correction but no compensation for sensor slope error. The look-up table pointer values were recorded as the measurement output data. The pointer was created from the sum of the power in the hot sensor (duty cycle variable), and optionally the temperature difference between the hot sensor and the target hot sensor temperature. The test was first performed in a lab with the probe of the commercial anemometer hand held in the air flow. The test was repeated in a garage with different equipment and with the probe fastened in a fixed position with respect to the hot sensor. The first test yielded very stable results over 8 measurements. The less stable results in the second test may be due to air turbulence in the air flow path or air r

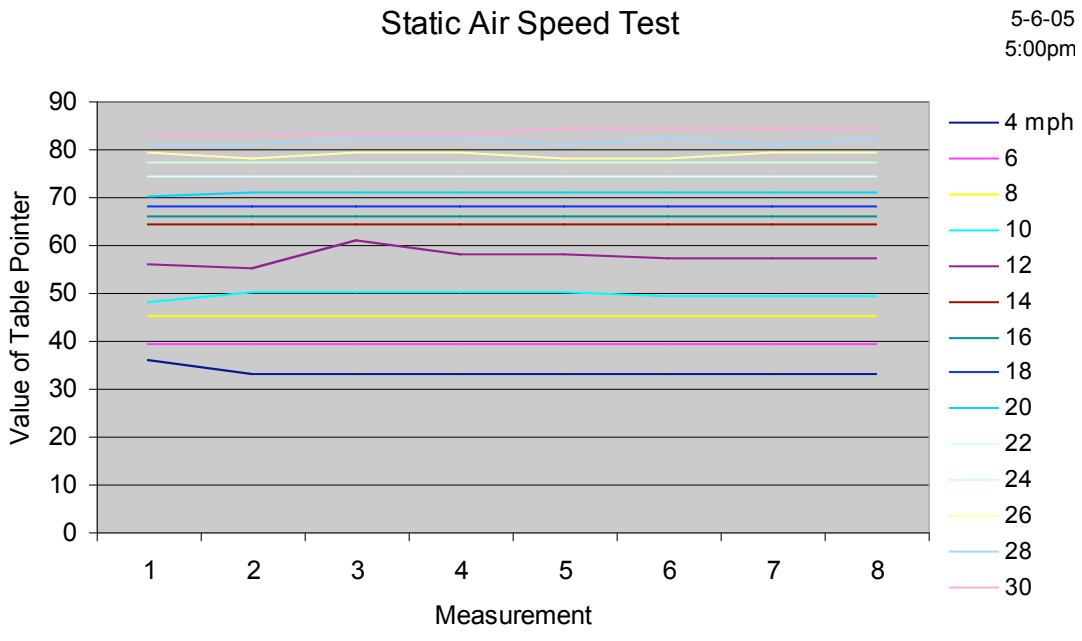


Figure 10: Static air speed test 5/6/05

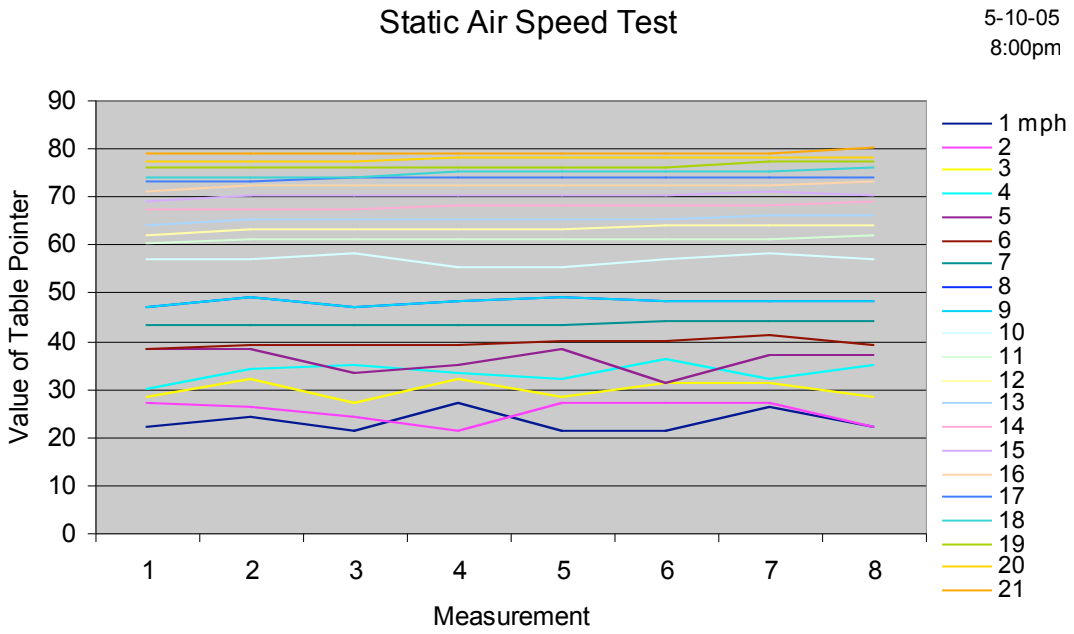


Figure 11: Static air speed test 5/10/05

2.2.6 Outdoor Air Flow Tests

Next a series of outdoor bicycle tests of the anemometer were performed. The tests were conducted at 5, 10, and 15mph with the anemometer mounted to the bicycle frame. For these tests, each data point collected represented an average of 8 consecutive measurements. The data from 5/26/05 (a particularly calm day) was used to assign the look-up table values for the instrument. These table values were then used to convert all of the previously recorded raw data to velocities. This data represents 15 different test sequences and is presented in Figures 15-29.

The data in Figures 19 and 23 looks questionable. For example, all of the 15 mph data from Figure 19 was at the maximum pointer value and even in Figure 23 the 10 and 15 mph data is well above the expected range. Upon further analysis, it was determined

that the cold sensor op-amp was at the negative rail. That is, the actual air temperature was below the cold sensor reading. This caused the target hot sensor temperature to be greater than 5 °C above ambient. As a result, the hot sensor heater was at 100% duty cycle and the measured temperature was well below the target. For this design and these component values, the lower temperature limit of operation is 55 °F. The data from these two tests is not valid and will not be included in subsequent analysis.

Comparison tests were performed utilizing a commercial anemometer, the Omega model HH-600. The test data was recorded in the units displayed on the device, feet per minute, and converted to miles per hour for the plot display. This data is shown in Figures 12 and 13. Figure 14 displays the same data but in blocks of 8 averaged values. This was done to present the Omega data in a form similar to the thesis sensor data, since that sensor also averages 8 readings per data point.

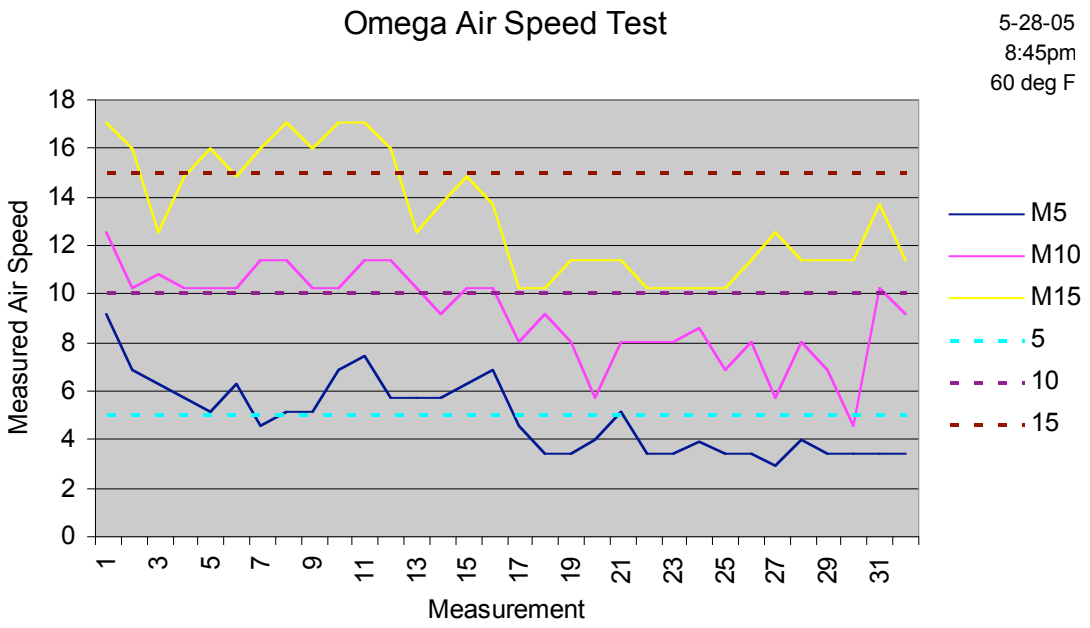


Figure 12: Omega moving air speed test 5/28/05

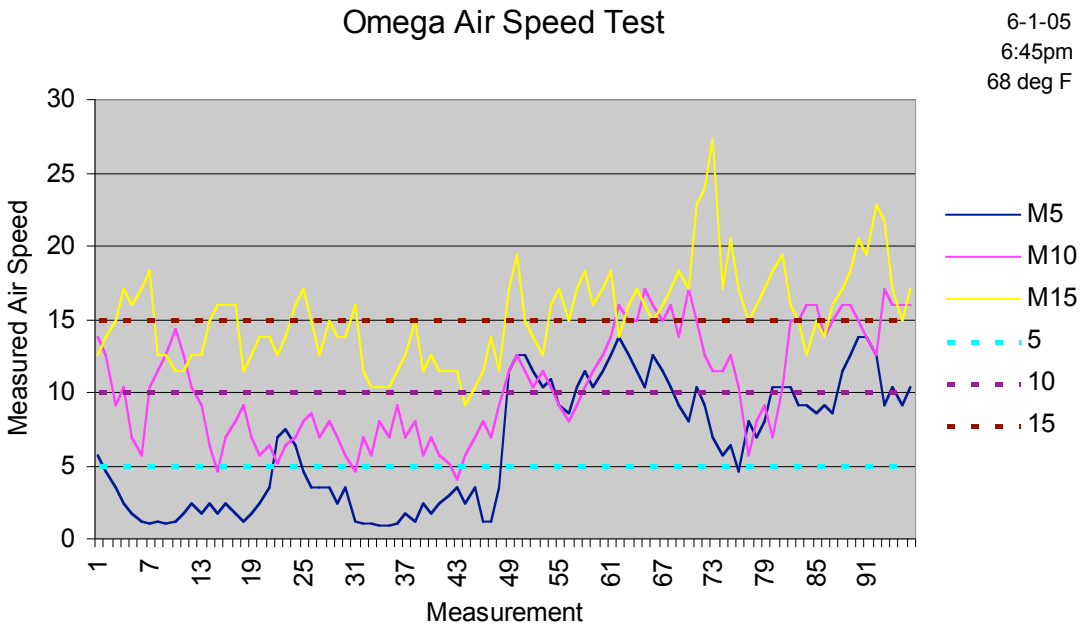


Figure 13: Omega moving air speed test 6/1/05

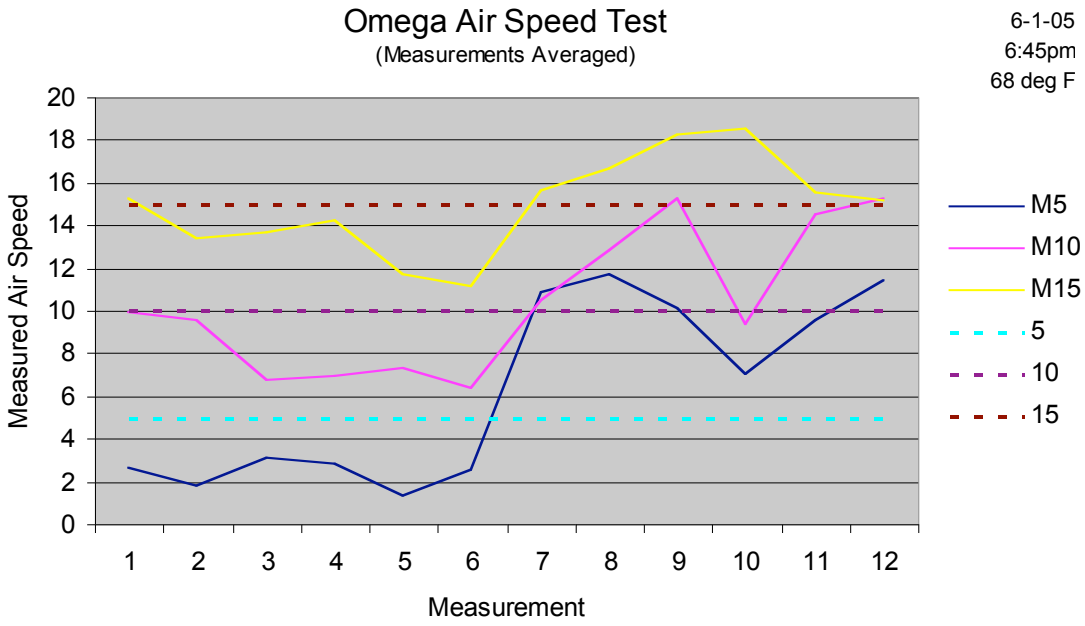


Figure 14: Omega moving air speed test 6/1/05 8 readings averaged

A further perspective can be achieved by averaging the entire set of Omega anemometer measurements to one value for each velocity tested.

Source	Description	5	10	15
Figure 12	average of 5/28/05 data	5.3	7.6	13.2
Figure 14	average of 6/1/05 data	6.2	10.4	14.9

Table VII: Average of Omega data

This can be compared to the average of the entire set of thesis anemometer data

Source	Description	5	10	15
Figure 15-29	average of all data from 5/8/05 to 6/1/05	5.5	10.5	15.1
	(excluding invalid data from 5/12, 5/16)			

Table VIII: Average of thesis data

The averaged data is remarkably accurate given the variances due to wind gusts and the estimated velocity accuracy of +/-10%. This data indicates that the thesis anemometer is comparable in accuracy to a commercial instrument, and has more than adequate accuracy for use as part of an energy monitor. The next section will analyze the accelerometer design of the system.

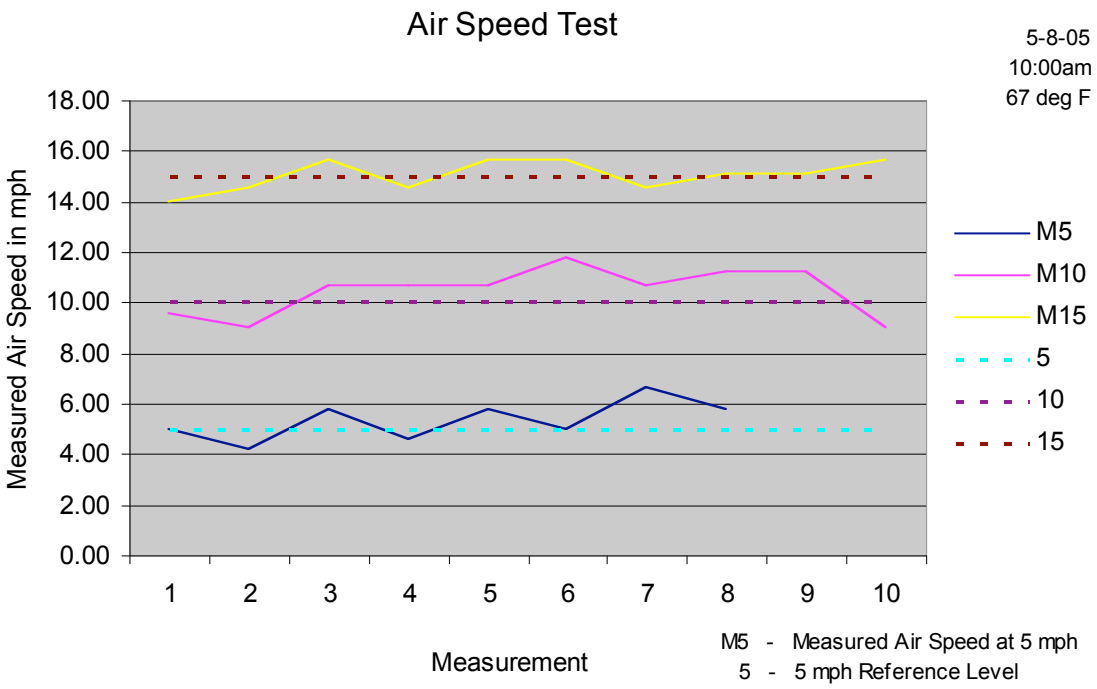


Figure 15: Moving air speed test 5/8/05

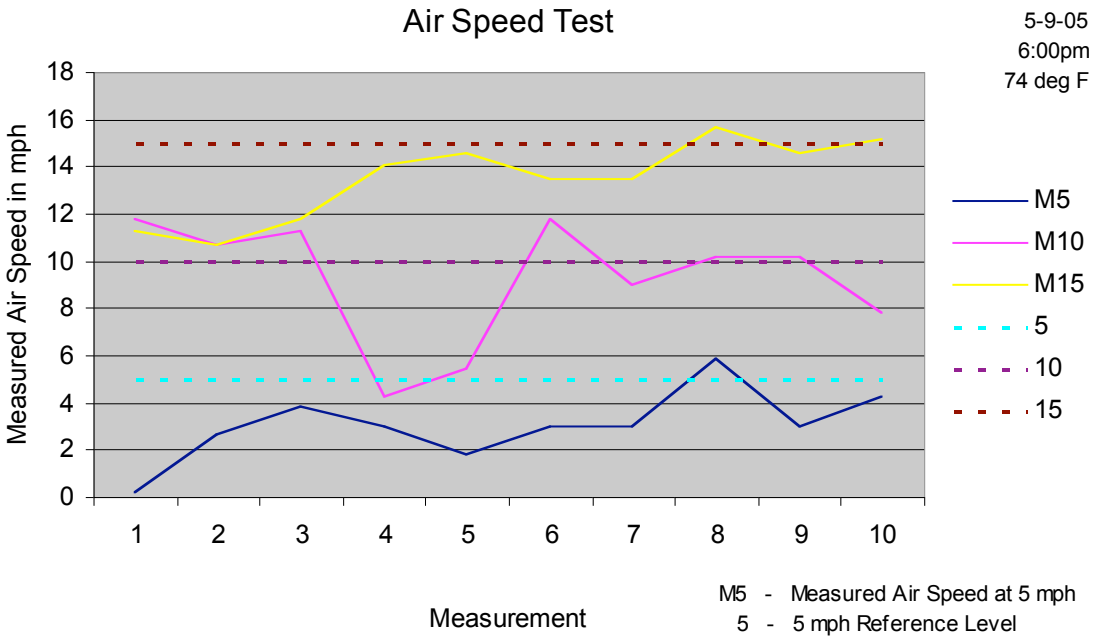


Figure 16: Moving air speed test 5/9/05

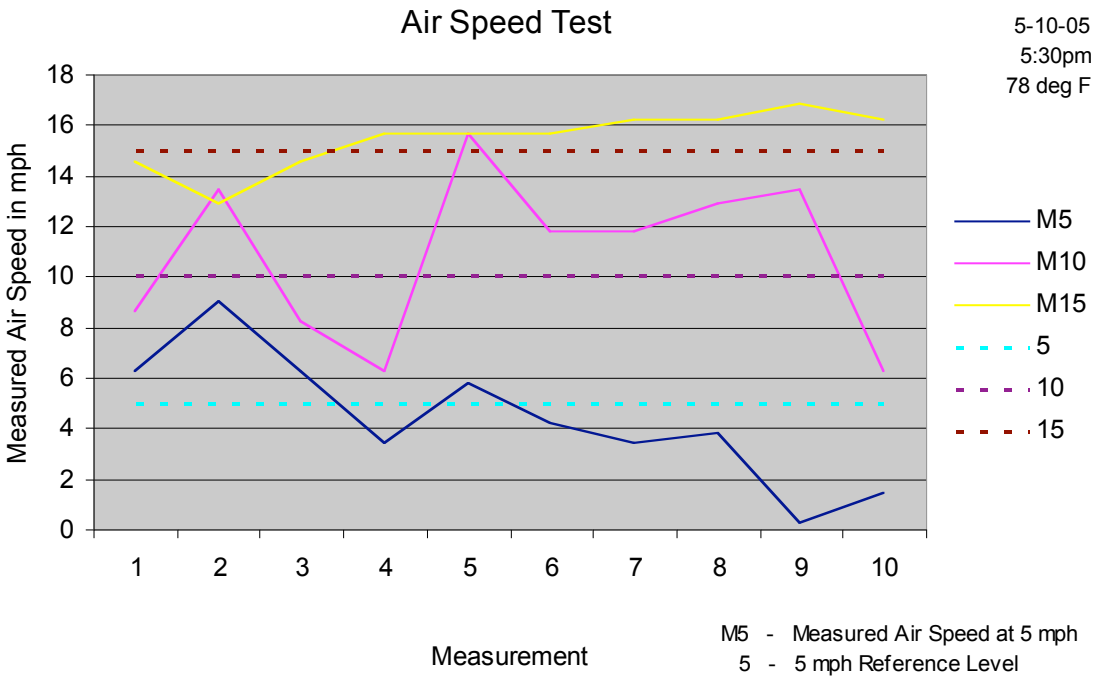


Figure 17: Moving air speed test 5/10/05

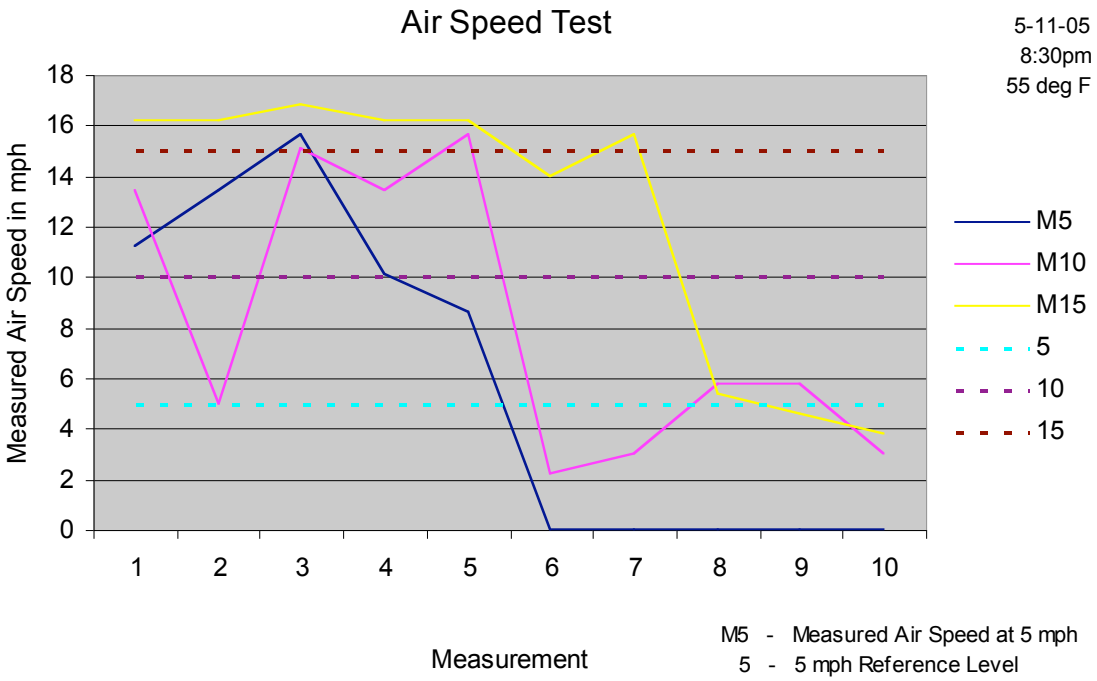


Figure 18: Moving air speed test 5/11/05

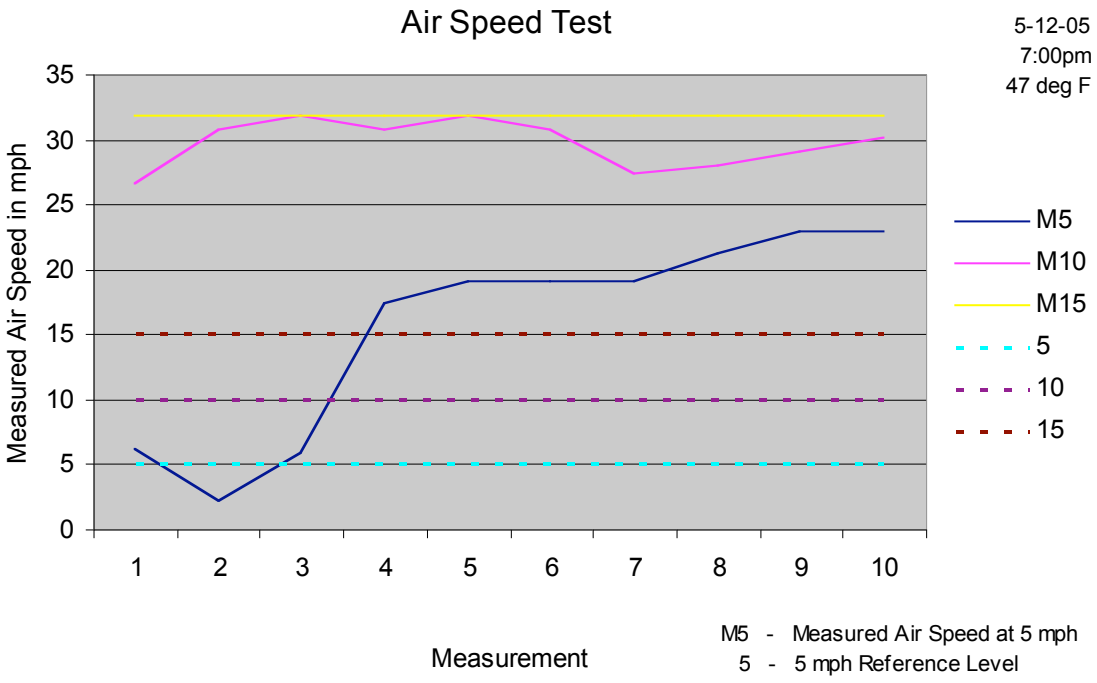


Figure 19: Moving air speed test 5/12/05

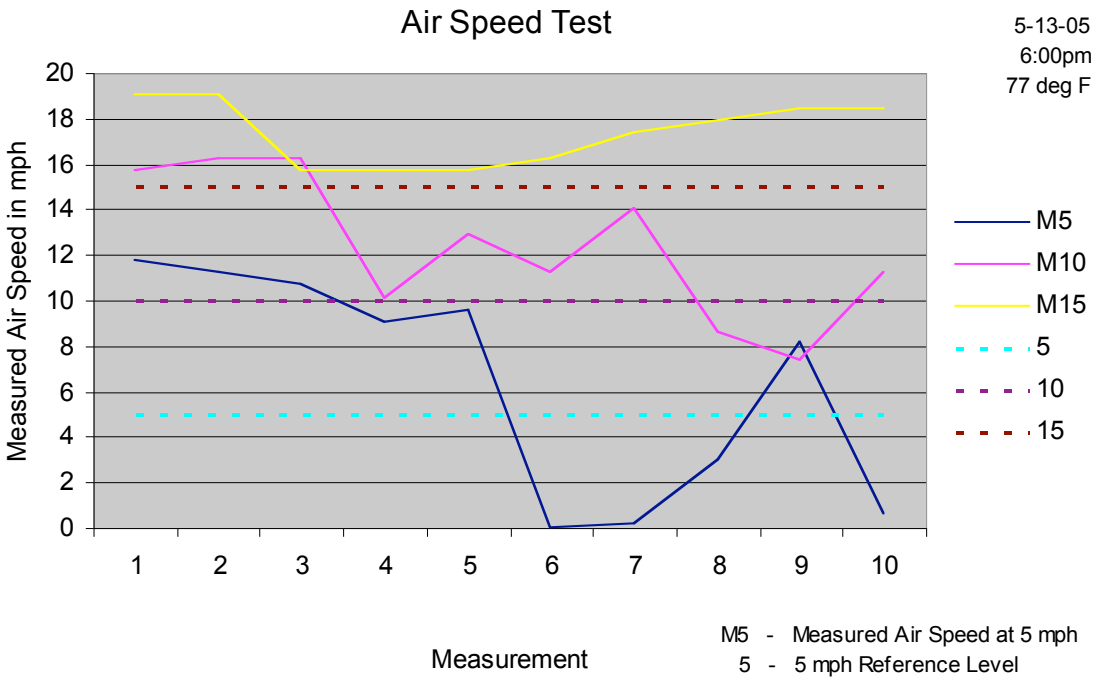


Figure 20: Moving air speed test 5/13/05

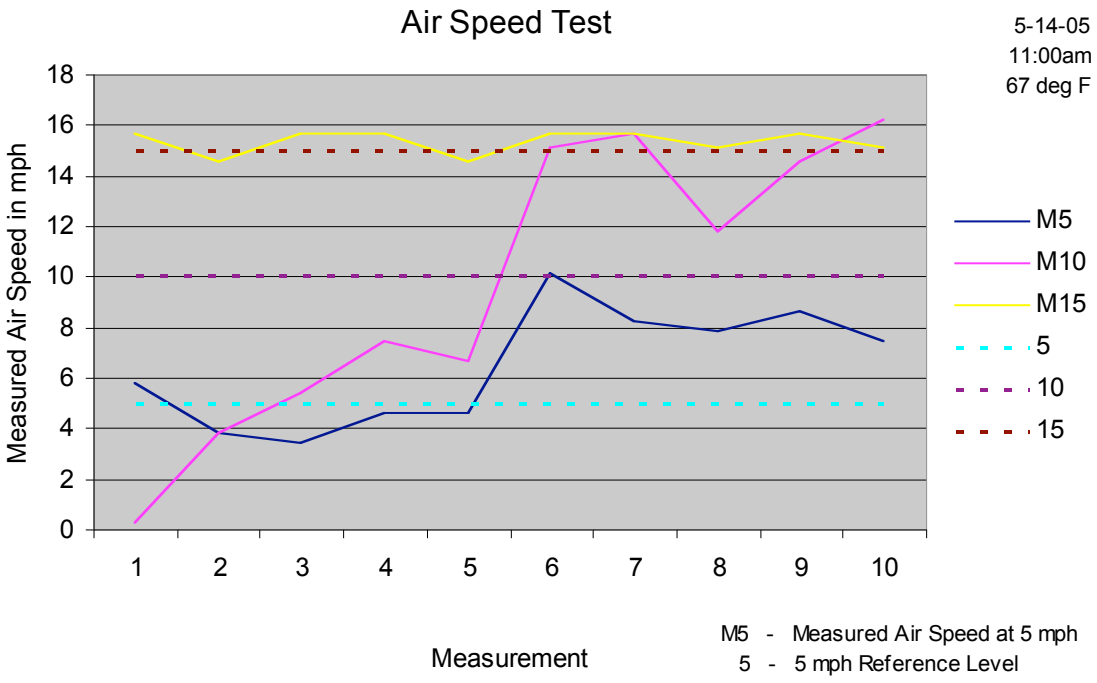


Figure 21: Moving air speed test 5/14/05

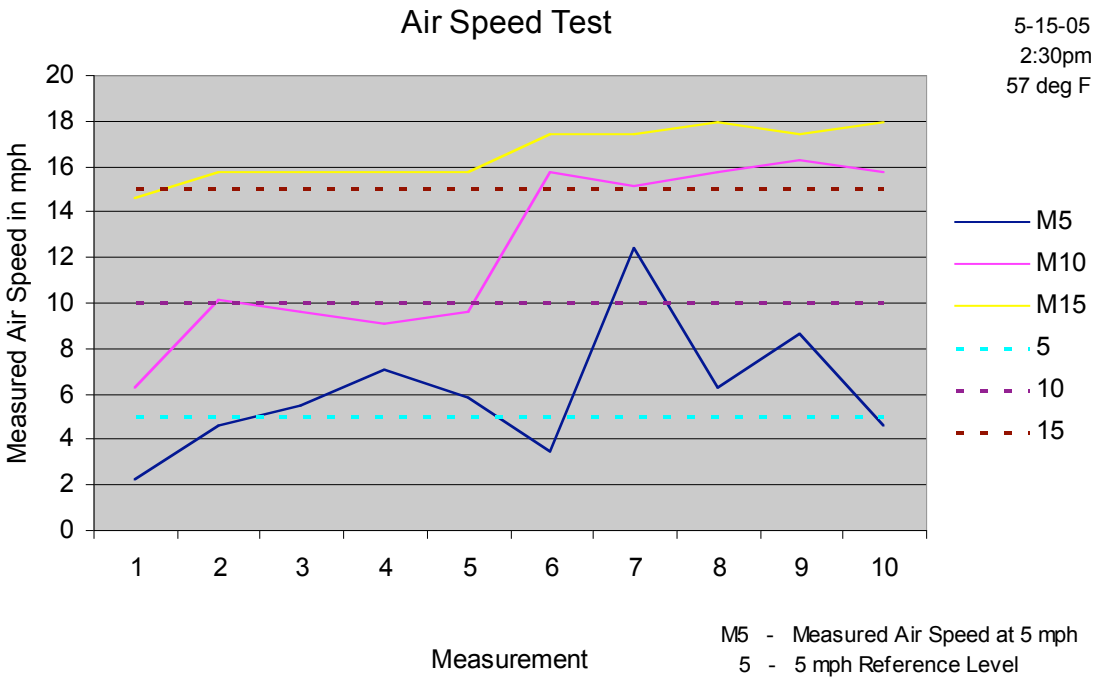


Figure 22: Moving air speed test 5/15/05

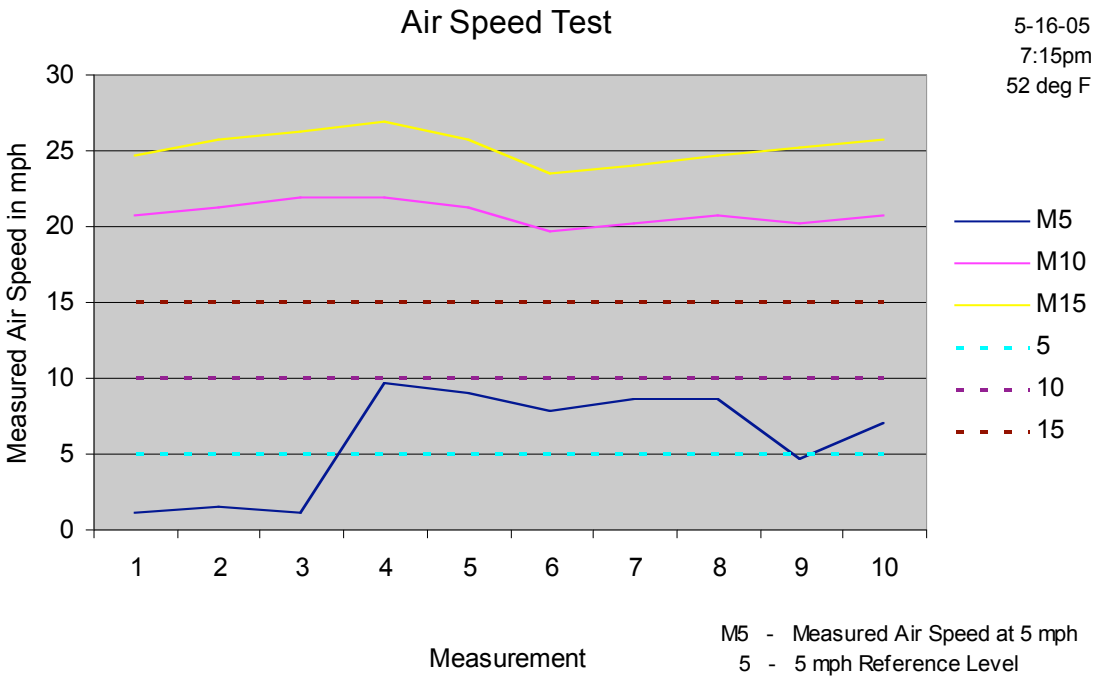


Figure 23: Moving air speed test 5/16/05

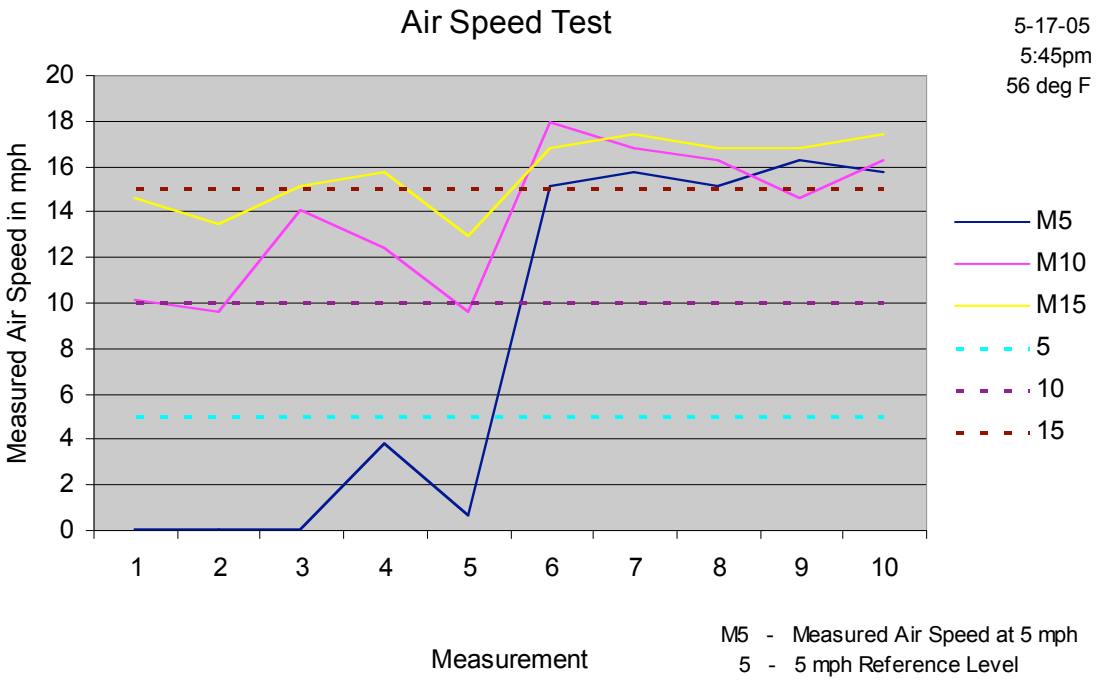


Figure 24: Moving air speed test 5/17/05

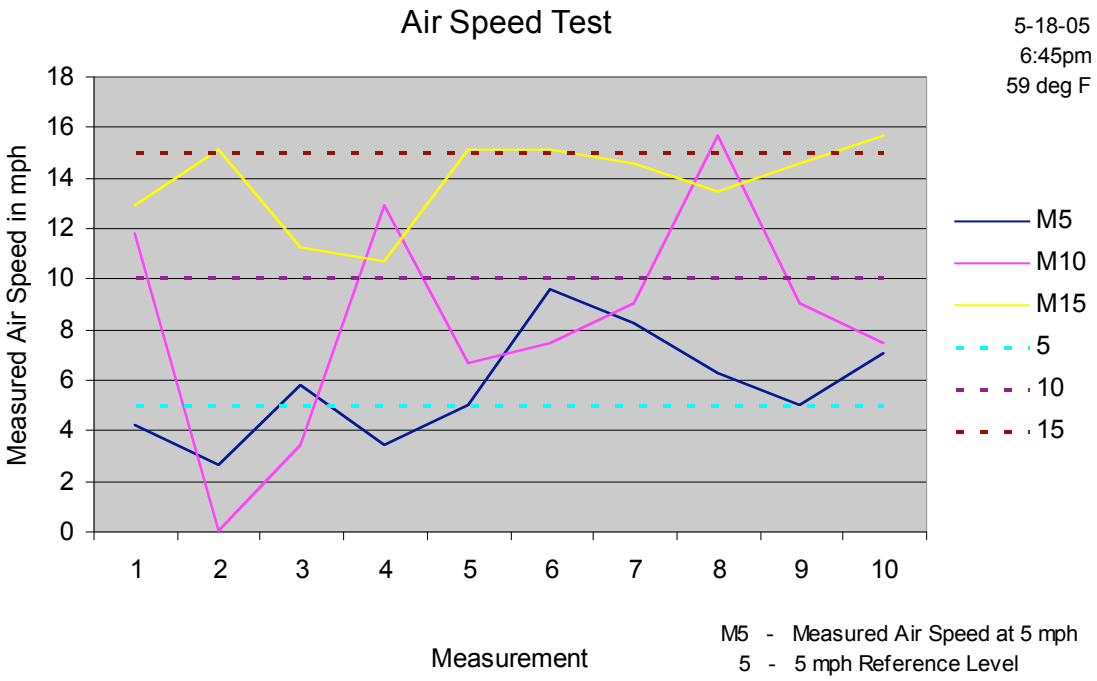


Figure 25: Moving air speed test 5/18/05

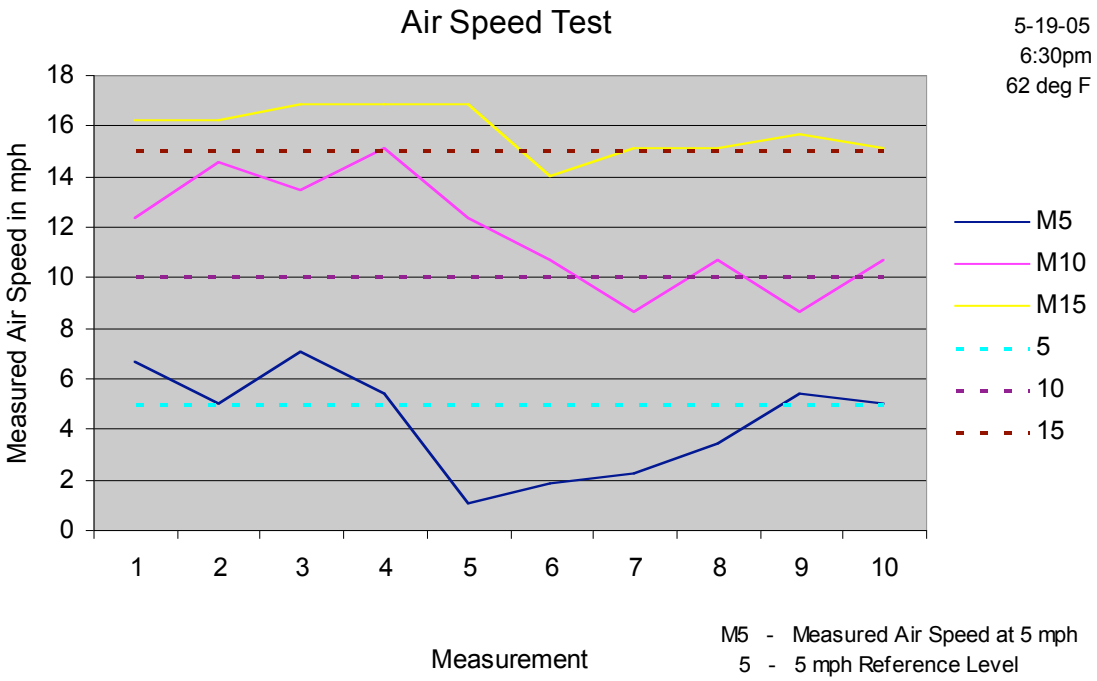


Figure 26: Moving air speed test 5/19/05

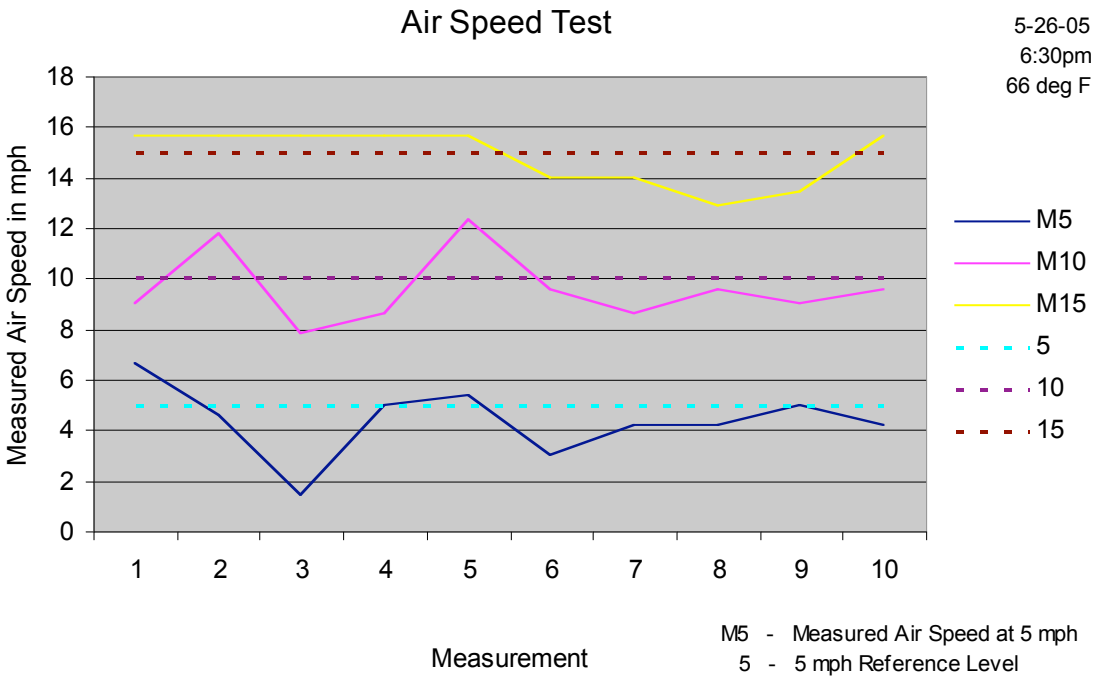


Figure 27: Moving air speed test 5/26/05

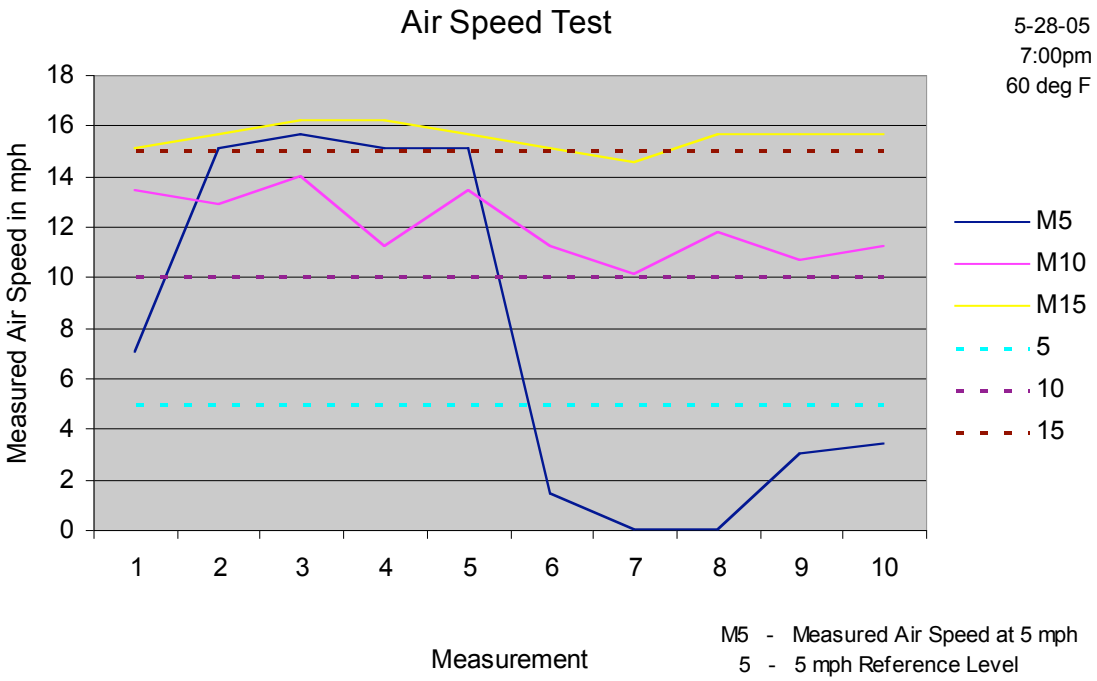


Figure 28: Moving air speed test 5/28/05

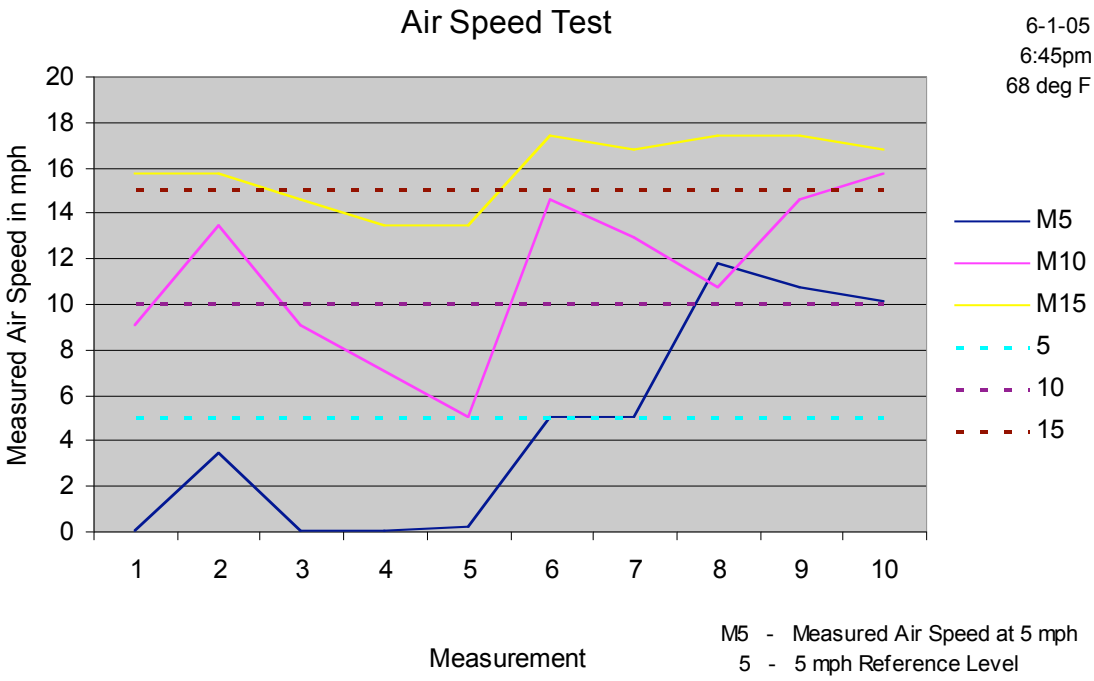


Figure 29: Moving air speed test 6/1/05

2.3 Acceleration Sensor

Accelerometers are sensors that measure the amount of acceleration due to either a change in velocity or the acceleration of gravity. Accelerometers have an axis of measurement and measure the amount of acceleration in that axis. If an accelerometer has response down to DC, the device can be used to measure tilt or angle of incline by measuring the component of gravity in the axis of tilt.

The fundamental structure of most accelerometers is a mass suspended by a spring. The device measures the force required to accelerate the mass as a measure of the amount of acceleration. This acceleration force is a function of the displacement of the spring. To determine the acceleration, the displacement of the spring can be measured or the stress induced in the spring material as a result of the displacement. There are other designs that use a force to prevent displacement which are called servo or closed loop accelerometers.

The accelerometer can be constructed using conventional manufacturing technologies by fabricating a mass, a spring, and a means of measuring displacement. More recently however, the majority of devices are being constructed using polysilicon techniques called micro-electro-mechanical-system (MEMS). These devices have micro geometries and are far less expensive enabling an explosive growth in accelerometer applications. Table IX contains a listing of the different accelerometer structures, the method of measurement, and other accelerometer characteristics.

Type	Structure	Displacement	Stress	Force	Measurement	Attributes	Open Loop Control	Closed Loop Control	Macro Geometry	Micro Geometry	Usage
Liquid	Air Bubble, Fluid Level	X			Fluid Position	Inexpensive	X		X		Carpenter's Level, Shock Sensors
Moving Coil	Mass Spring			X	Magnetic Field	Usually Servo Control		X	X		
Piezoresistive	Mass Spring		X		Resistance Change	Low Output Impedance, Non-Linear Output	X	X	X	X	
Piezoelectric	Mass Spring		X		Induced Charge	Capable of High g Forces, 10,000 g	X	X	X	X	High Bandwidth Vibration Measurements
Capacitive	Mass Spring	X			Capacitance Change	Inexpensive, Low g to 30,000g	X	X		X	
Inductive	Mass Spring	X			Inductance Change	Less Common	X		X		
Magnetostructure	Mass Spring		X		Magnetic Property Change	Less Common	X			X	
Thermal	Mass Spring	X			Heat Transfer, Polysilicon-Aluminum Thermocouple	Less Common	X			X	
FET Displacement	Mass Spring	X			Gate Channel Displacement	High Sensitivity, Voltage Output	X			X	
FET Stress	Mass Spring		X		Stress Modulated Band Gap Energy	Potentially Low Cost	X			X	
Laser Doppler Accelerometer, (LDA)	Light	X			Doppler Shift	No Attachment Required, High g, No Upper Measurement Limit	X		X		Measuring Extremely High Accelerations
Laser Mass Spring	Mass Spring	X			Interferometric Change in Cavity Resonance	High Sensitivity, 100X Piezoelectric	X		X		Gravitational Wave Detector
Superconducting Angular Accelerometer	Flexure Pivot	X			Superconducting Quantum Interference Device Amplifier	Low Noise	X		X		Gravity Gradiometry
Suspended Mass	Mechanically Suspended Mass, Electrostatically Suspended Mass	X		X	Multiple	Extreme Sensitivity 1 fg	X	X	X		Satellite Deceleration While in Orbit

Table IX: Accelerometer table

This energy monitoring system requires a device that will measure the acceleration due to velocity change as well as the partial acceleration due to gravity. Since the driving force is limited and climbing a slope involves a fraction of the acceleration from gravity, the device needed is a low g accelerometer. In addition, since climbing even a slight grade over a period of time will still require a significant amount of energy, the accelerometer must have very high sensitivity. Listed below is a summary of the major accelerometer requirements for the energy monitoring system.

1) integrated device for small size and low power
2) response down to DC
3) maximum g rating above the expected road shock
4) very low bandwidth
5) very high resolution
6) easily interfaced to a uC

The integrated capacitance change sensors are low cost and are available in a variety of ratings. The ADXL320 from Analog Devices is an analog sensor that was tested further to validate the suitability for this application [ana 04].

The acceleration limit for device damage, is usually well above the expected range of operation. However, the acceleration limit for measurement, must be adequate for the g forces expected during operation. The majority of HPV are 2 wheeled bicycles with no suspension. It is plausible that a frame mounted accelerometer could see several g(s) of acceleration from road shock. If the device has adequate range for the forces expected

including shocks and vibrations, then these short term bursts are integrated over a longer sampling period and the average correctly represents the desired measurement data. If however the g force substantially exceeds the rating of the accelerometer, then the device will fail to correctly include the data that exceeds the maximum g rating. This could possibly lead to measurement errors that are significant.

To determine this, a test was conducted where the accelerometer was mounted to the frame of a bicycle and the vehicle ridden over terrain typical of that encountered during an exercise riding interval. The oscilloscope image in Figure 30 shows the accelerometer output using a relatively high bandwidth. The maximum operational rating of the accelerometer was 5 g and it was operated with a bandwidth of 500z. The terrain was over concrete sidewalk and the velocity was 15 mph. Under these conditions the peak voltage is 1.08 V representing a maximum acceleration of 3.46 g. Since the device rating is 5 g, this measurement is well within the rating of the sensor.

Next it was necessary to determine the proper bandwidth of the device. Since the system software will capture the accelerometer output at discrete points of time, it could easily fetch a measurement when the accelerometer voltage was at a peak. Therefore it is necessary for the bandwidth be lowered so the measurement is representative of the vehicle motion, not bumps in the road. In Figure 31 the accelerometer bandwidth has been reduced to 5 Hz. To properly scale the waveform, the sensor output has been amplified with a gain of 10. With this bandwidth the peak g measurement has been reduced to 0.5 g.

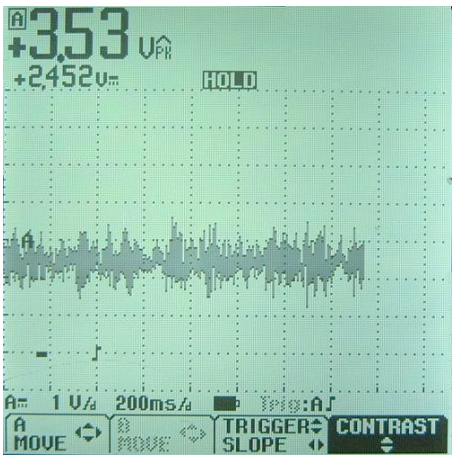


Figure 30: Oscilloscope picture of accelerometer shock test 500 Hz

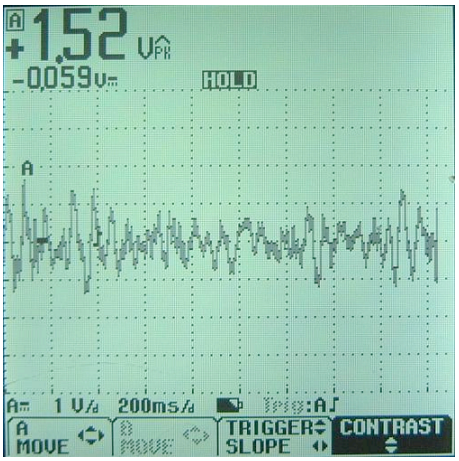


Figure 31: Oscilloscope picture of accelerometer shock test 5 Hz

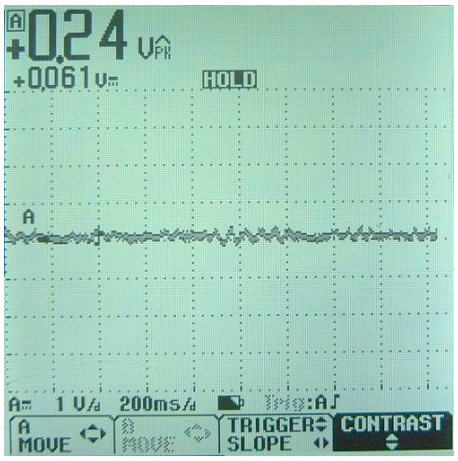


Figure 32: Oscilloscope picture of accelerometer shock test 0.5 Hz

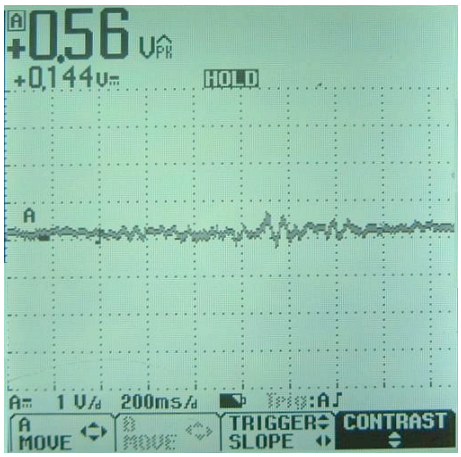


Figure 33: Oscilloscope picture of accelerometer shock test 0.5 Hz with bump

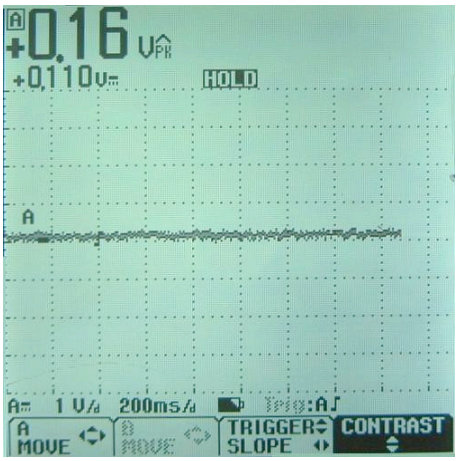


Figure 34: Oscilloscope picture of accelerometer shock test 0.1 Hz

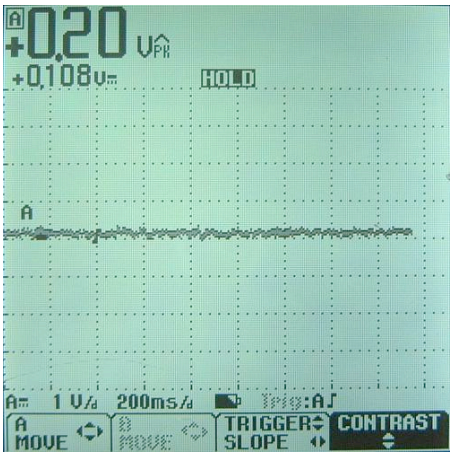


Figure 35: Oscilloscope picture of accelerometer shock test 0.1 Hz with bump

Figure 32 shows a further bandwidth reduction to 0.5 Hz with a peak g measurement of 0.06 g. Figure 33 has the same bandwidth but the test vehicle was ridden over a moderately severe bump. The effect of the shock can be seen as a transient just past the 1 second mark of the trace. The peak g measurement is 0.13 g. Figure 34 shows a further bandwidth reduction to 0.1 Hz with a peak g measurement of 0.06 g. The same bandwidth reduction in Figure 35 also includes the test over the moderately severe bump. The shock from the bump is no longer visible in the trace and the peak g measurement is 0.029 g. This data indicates that the bandwidth of the accelerometer must be reduced to a fraction of 1 Hz to insure proper sensor measurements.

The accelerometer must also have very high resolution to detect subtle differences in the grade or slope of the terrain. First the relationship between the measured acceleration and slope must be defined.

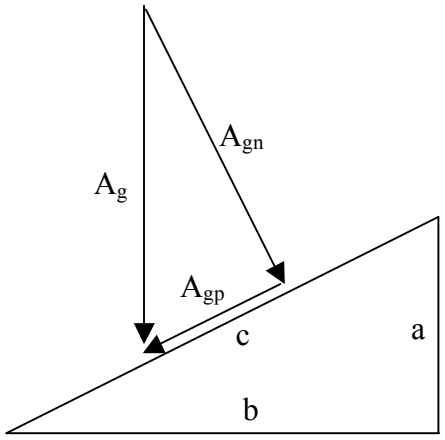


Figure 36: Vector diagram of gravity

Referring to vector diagram of Figure 36, if the acceleration due to gravity A_g is represented by a force vector it may be broken down into a component normal to the

slope A_{gn}, and a component parallel to the slope A_{gp}. If the accelerometer measurement axis is aligned to the direction of travel or parallel to the slope, the accelerometer will measure A_{gp}.

Now two of three sides of this triangle are known and the third is given by

$$A_{gn} = \sqrt{A_g^2 - A_{gp}^2}$$

Since ΔA_g, A_{gn}, A_{gp} and Δ a,b,c are similar triangles, the slope of the hill is given by the opposite side divided by the adjacent side.

$$\text{Slope} = \frac{a}{b} = \frac{A_{gp}}{A_{gn}} = \frac{A_{gp}}{\sqrt{A_g^2 - A_{gp}^2}}$$

The device resolution is determined assuming a 10 bit A/D converter at 5 V reference levels. The voltage change per LSB change is

$$5V/1023 = 4.88 \text{ mV}$$

For the ADXL320, the un-calibrated sensor output voltage change for 1 g is 0.312 V. Including a gain of 50 for the channel amplifier, the A/D resolution is

$$4.88 \text{ mV}/(0.312 \times 50) = 313 \text{ ug}$$

Note that with a gain of 50 the maximum measurable acceleration is << 1 g. Entering this value into the formula above yields

$$A_g = 9.81 \text{ ms}^{-2}$$

$$A_{gp} = 9.81 \times 313 \text{ ug} = 0.003 \text{ ms}^{-2}$$

$$\text{Slope} = 0.003/\sqrt{(9.81)^2 - (0.003)^2} = 0.0003$$

This is a very gradual slope. For example, in 12 inches the elevation change is

$$12” \times 0.0003 = 0.0037”$$

This is about the thickness of a piece of paper and is more than adequate resolution for the energy monitoring system. The next section briefly covers the method of measuring the velocity and distance traveled.

2.4 Velocity and Distance Sensor

The velocity of a vehicle has a major impact in the power required to sustain forward motion. Nearly all of the energy expended when riding is required to oppose friction and overcome gravity. Power is a function of force, distance, and time and is the primary method employed in this system to monitor energy. To calculate the power the system must know the velocity which can be determined by measuring the time elapsed and the distance traveled. This section describes a simple sensor that is combined with software to provide significant information about the motion of the vehicle.

A variety of sensing devices have been used to monitor rotating elements where some signal is generated for each revolution. If the wheel circumference of a bicycle is known, the distance traveled can be determined by counting the wheel revolutions. To calculate the velocity, the time of each revolution is measured resulting in an incremental velocity for each wheel revolution. Likewise, the change in velocity from 1 revolution to the next can be used to determine the incremental vehicle acceleration. A series of these incremental measurements can be averaged when necessary.

This sensor uses a permanent magnet mounted on the front wheel and a reed switch mounted on the front fork to generate a digital signal for each wheel revolution. The passing magnet causes a momentary switch closure that is connected to a uC digital input bit. In the uC the wheel revolution data is processed to provide the distance, velocity, and acceleration of the vehicle. Additional information on the wheel interrupt interface and the method of calculation is covered in the next chapter.

CHAPTER III

MICRO-COMPUTER IMPLEMENTATION

A moderately complex instrument such as an energy monitor requires a significant amount of data processing and computation. The values captured at the sensing elements must be converted to variables based on standardized units of measurement. These variables require the use of specific algorithms which prepare the data for optimum user presentation. In addition, the entire monitoring system must be light, compact, and operate off of a long life battery. This chapter describes various aspects of the system including the embedded processor, the circuit design, the prototype fabrication, the software management, and the overall system operation.

3.1 Circuit Design

The initial task of the processor is to continuously collect data from the 4 sensor inputs:

1) Cold Temperature Sensor
2) Hot Temperature Sensor
3) Acceleration Sensor

4) Wheel Revolution Sensor

The first 3 sensors are analog devices and each channel interface requires A/D conversion with corresponding range, calibration, and resolution requirements. These requirements are addressed by the design of the channel interface which includes the selection of the type of sensor, the amplification or attenuation of channel signal gain prior to conversion, the type of A/D conversion, the design tradeoff between channel resolution and bandwidth, and the software drivers that complete the channel interface. The last channel is the wheel revolution sensor which is essentially a single bit digital input. This channel design must address switch de-bounce and the designed range of operation.

Both the temperature and acceleration sensors are available as digital devices. No separate A/D conversion is required for these devices which simplifies the interface to the uC. There are design tradeoffs however that made these choices unattractive for use in this system. While the LM335 is an analog current sinking device that permits self-heating, a digital temperature sensor requires external heating. The digital accelerometers are PWM devices and require timing the duty cycle of a PWM output. Because 2 uC timers are already being used for wheel revolution timing, process event timing, and clock timing, it is not possible to capture the PWM duty cycle with the required accuracy. For these reasons, analog sensors were used and A/D conversion was done in the uC.

The major consideration in A/D conversion is the LSB voltage change relative to the signal being acquired. One technique available to increase resolution is to compress the A/D voltage reference levels. A problem with this approach is that the range of operation for the temperature sensor output is different than the voltage range for the accelerometer. However, by using a resistor divider network in conjunction with a digital output from the uC, it is possible to dynamically change the vref+ and vref- voltage levels. This makes it possible to set the vref voltage inputs to optimum levels for making temperature sensor A/D measurements, and change the levels when making an A/D measurement from the accelerometer. Since the A/D vref inputs are relatively low impedance, low resistance divider networks or buffer amplifiers are required. In addition, the minimum vref+ to vref- voltage difference for the 16F877 is 2V. This limits the resolution enhancement that can be achieved with this technique. For these reasons, adjusting the voltage reference levels was not a design approach utilized in this thesis.

The temperature sensor output for the LM335 is a voltage proportional to the temperature in degrees Kelvin. The voltage change with temperature is 10 mV/degC. If the A/D reference inputs are connected to +5V and ground, the LSB voltage is 5 mV. This gives a resolution of 0.5 degC/bit. If a channel amplifier with a gain of 10 is used the resolution is improved to 0.05 degC/bit. The LM11 op-amp was selected for the channel amplifier due to its low supply current and very low offset voltage. A DC offset was included in the amplifier to center the amplifier output over the expected temperature operating range. Figure 37 shows the temperature channel amplifiers and the rest of the circuit design of the energy monitoring system.

The accelerometer sensor output is a voltage proportional to the acceleration in g expressed as m/s^2. The voltage change with acceleration is 312 mV/g. Directly connected the resolution is 16 mg/bit. Using a channel amplifier with a gain of 50 improves the resolution to 313 ug/bit. Again the LM11 op-amp was used for this A/D channel amplifier.

The wheel revolution sensor is connected to the RB0/INT pin as an external edge triggered interrupt. A small capacitor is added to provide an adequate level of de-bounce for this signal.

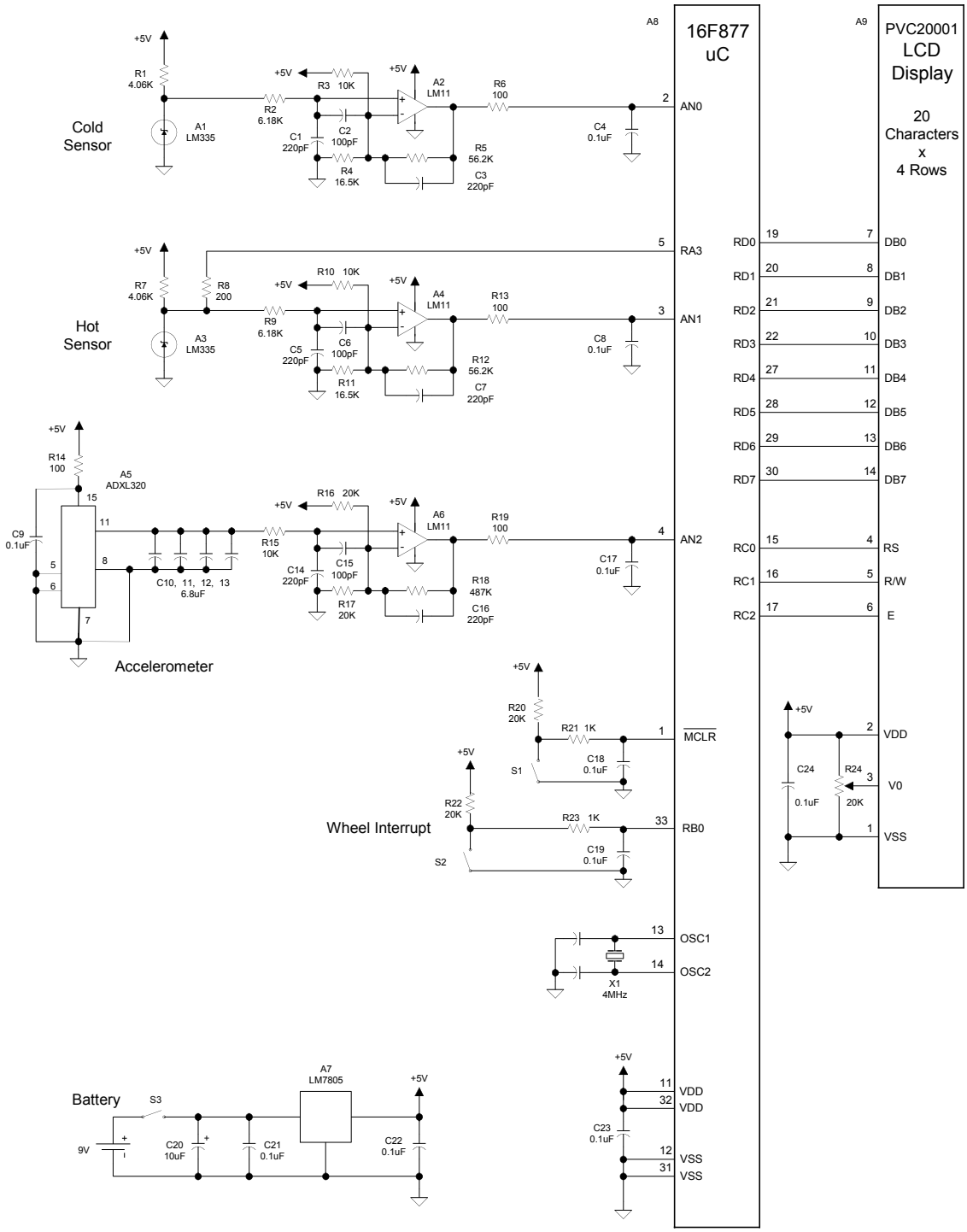


Figure 37: System schematic

The measurement system uses a 20 character by 4 line alphanumeric LCD display with a parallel data interface. The processor converts each of the floating point measurement values to an ASCII character string with appropriate display formatting. This display value is transferred to the display device one character at a time over an 8 bit data bus with 3 wire handshaking.

3.2 Construction of the Prototype

For signal integrity of the monitoring system, the circuit layout must process the measurement signals while rejecting noise from internal and external sources. In addition, the entire assembly must be rugged and protected from the elements. To achieve this the circuit components are mounted on a 1/16" G10 epoxy board shown in Figures 38 and 39. Since the assembly is tilted for viewing, the accelerometer is mounted at a 27 deg angle to maintain horizontal alignment of the measurement axis as shown in Figure 40. The LCD display components are mounted on a separate PC board, and the display and main boards are connected with a flat ribbon cable.

The circuit boards, the battery, and the thermal sensors are mounted in a two piece die cast box as shown in Figure 41. To access the air flow the tops of the thermal sensors are exposed on one side of the box. A sheet metal tunnel covers the thermal sensors to minimize the influence of a cross wind. On the bottom of the housing is a modular connector for connection to the wheel pulse switch. The entire assembly weighs 227g. The next section covers the system software.

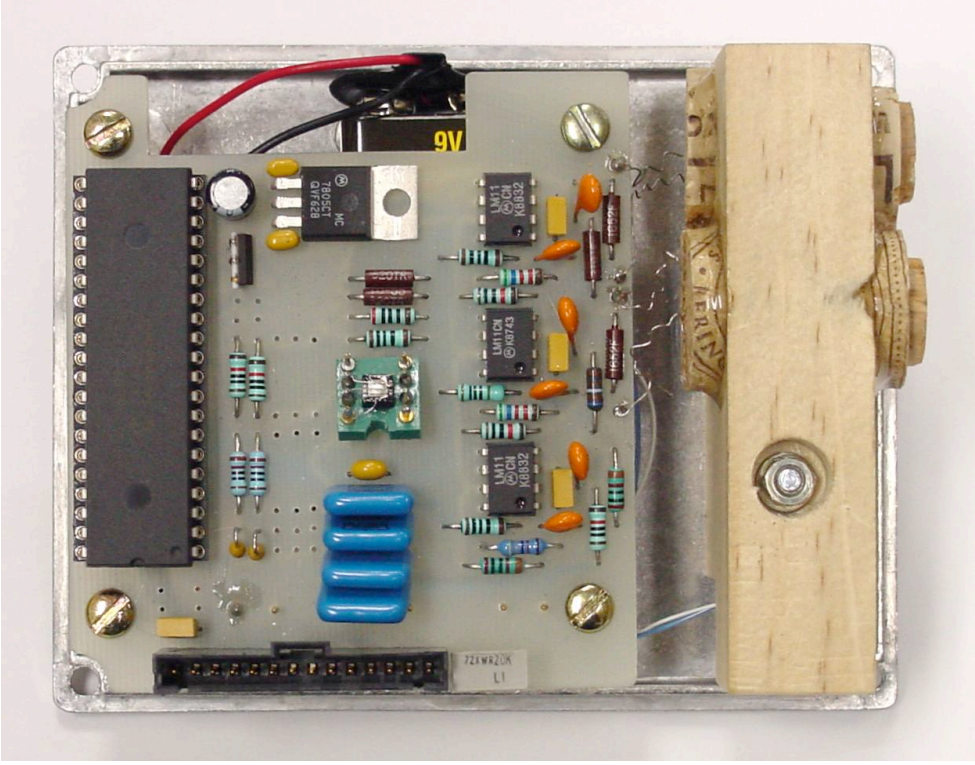


Figure 38: Picture of top of circuit assembly

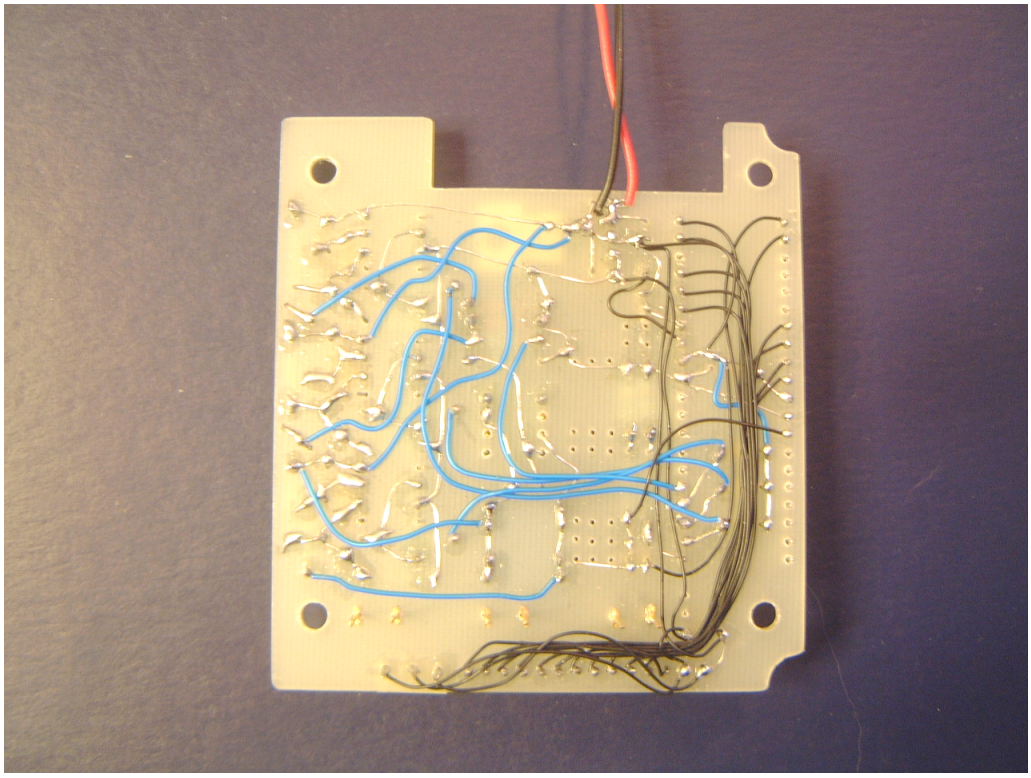


Figure 39: Picture of bottom of circuit assembly

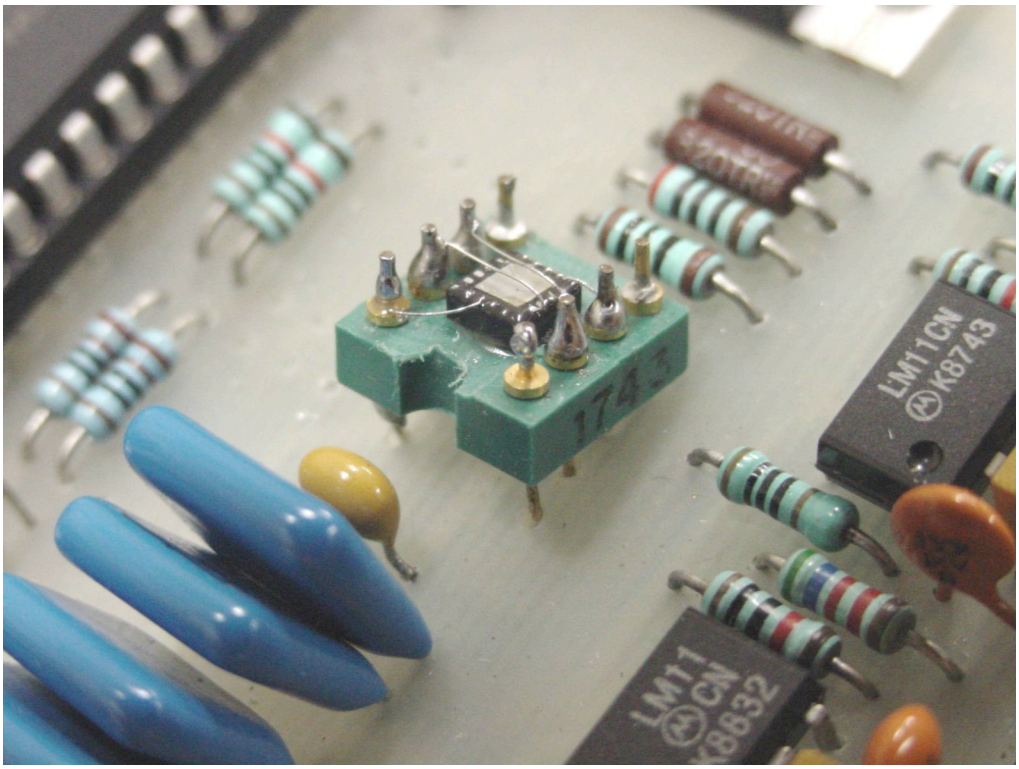


Figure 40: Picture of accelerometer

Figure 41: Picture of enclosure

3.3 System Software

In any real time monitoring system the software structure is critical. The measurement data typically must be captured at specific times and each data channel must not interfere with other data gathering. In addition, the time available for data processing is limited; as new data becomes available a measurement must be made and the data entered into the system. The software for this system combines interrupt and main program loop structures that collect data over 3 different time intervals. The length of each interval corresponds directly to the particular data being captured and pertinent data can be exchanged between the sections of code in each interval.

The first loop sequence executes at 10 mS intervals and is driven by a timer interrupt. The primary function of this code is to serve as the measurement loop for the hot thermal sensor. Since the sensing element is also the heating element, excellent temperature control is achieved with this time interval. In addition to the hot sensor, the accelerometer measurement is also captured in this interval. This is necessary to prevent A/D measurement contention and makes the most recent reading available to the main program loop.

There are several calculations that must be performed in the 10 mS interval. The most important of these is the heater control. A comparison is made between the 5 degree above ambient target and the latest hot sensor temperature. If the sensor is too hot the heater is turned off and if it's too cold the heater is turned on. To capture the duty cycle of the heater, the T_Hpower variable is incremented every time the heater is turned

on. This value is used in the main program loop to calculate the power in the hot sensor and the corresponding air flow when the air speed monitoring system is in the constant temperature mode. The 16 bit variable T_Hdif contains the binary magnitude of the temperature difference between the hot sensor and the 5 degree target. This variable is the controlling parameter for air speed when the system is in the constant power mode. Prior to reading the hot sensor the heater is turned off to minimize any influence on the temperature reading.

The next interval follows in sequence from the first and occurs at 1 second intervals. This interval is created by a branch controlling loop counter with a value of 100. It is locked in sequence with the first interval and will never execute without the 10mS interval occurring first. There are several important functions performed in this interval. The temperature of the cold sensor is acquired and made available to the main program loop. From the cold temperature measurement the 5 degree above ambient target for the hot sensor is calculated. This is done in a few microseconds by adding the binary magnitude equivalent of 5 degrees to the 10 bit binary magnitude reading of the cold sensor. An additional offset is added at this point to correct for calibration errors in the temperature sensors.

The next section of code in this interval prepares the look-up table pointer. This pointer is created primarily from the duty cycle variable (T_Hpower), which represents the power in the hot sensor. The duty cycle value is shifted by subtracting 40 to optimize the look-up table range. If the duty cycle is 100%, the temperature difference T_Hdif is

compressed and added to extend the table pointer. This block of table pointer values corresponds to air speed sensor operation in the constant power mode. The pointer is protected for under or over range and averaged over 8 readings. This smoothing of the air speed pointer is done primarily to add stability to the wind speed display.

The last section of code in the 1 second interval updates the time counters. These are simply variables that are coded to increment as a time display in hours, minutes, and seconds. Since this section of code is executed at 1 second intervals, the seconds counter is incremented here. This concludes the 10mS and 1 second code intervals.

The last time interval is triggered by an interrupt but actually executes in the main program. The execution of this section of code is initiated by a wheel pulse interrupt and the time interval corresponds to the time required for 1 wheel revolution. Since this time varies, the execution of this code sequence is asynchronous to the 10 mS and 1 second intervals. Essentially the interrupt loops capture the data and the main program processes and displays the data. It is important to note that the three different time intervals are adequately non-competitive. If the 10 mS interrupt is delayed by a wheel pulse interrupt, the timer continues to run and the next 10 mS interrupt will occur on time. If the wheel pulse interrupt is delayed by a 10 mS interrupt, the delay is a small fraction of the wheel pulse interval and appears as noise in the data flow. There is not an issue with the 1 second interval as it is synchronized to the 10 mS interval.

The wheel pulse event triggers the execution of the main program. That is, every time there is a wheel pulse interrupt, the entire section of code in the main program is executed to completion.

The following is a list of the steps of main program execution.

Calculate the time of the last wheel revolution
Update the distance variable
Update the velocity variable
Calculate the acceleration (based on changes in wheel speed)
Calculate the acceleration (based on the accelerometer reading)
Calculate the acceleration (based only on the slope of the vehicle)
Calculate the grade
Calculate the elevation change
Update the elevation total
Calculate the cold sensor temperature in degrees Fahrenheit
Calculate the power in the drive train
Calculate the air speed
Calculate the force due to the air speed
Calculate the head wind
Calculate the total Watts being expended
Calculate the kilocalories in this time interval
Update the kilocalorie total
Display all data
Go back and wait for the next wheel pulse interrupt

The first part of the main program sequence contains code to control the loop branching. There is a timeout if no wheel pulse has occurred to allow an update of system variables. In addition, two consecutive wheel pulse interrupts are required for normal execution of the main program. This prevents a wheel pulse conflict with the overflow counter and assures proper velocity calculation.

The majority of the main program is a linear sequence of variable calculations designed to progressively acquire the desired display data. All of this calculation is done in 32 bit floating point math. Three routines greatly facilitate the movement of data for

these calculations, To_AARG, To_BARG, and AARG_to. In the floating point routines, AARG is the A argument location and BARG is the B argument location. Since indirect addressing is utilized, to transfer data to the AARG floating point calculation registers, the address of the data is loaded in the W register and a call made to the To_AARG routine. Using these routines make this section of code compact and more readable. Once all of the desired data has been calculated, the last section of code transfers the data variables to the LCD display.

The main program sequence of code takes less than 40 mS to execute. Since the wheel interrupts occur at 120 mS intervals at a vehicle speed of 30 mph, there is ample time for this code to complete execution prior to the next interrupt. Listed below are the subroutines that facilitate the overall code execution.

Init_FP
Float_ascii
To_AARG
To_BARG
AARG_to
Ten_ms
Disp
LCD_labels
LCD_init
LCD_cmd
LCD_data
LCD_busy
FPA32
FPS32
FPM32
FPD32
INT3232
FXD3216U

With the exception of the fast computations done in the 10 mS interval, all variable calculation is done in 32 bit floating point arithmetic. AN575 describes the floating point package that was utilized. AN670 is a conversion program that creates an ASCII string from a floating point variable. This routine was modified to suppress leading zeros and to relocate the sign character to the immediate left of the first nonzero character. These modifications improved the readability of the display. Figures 42-45 display the flow chart of the system software and in appendix A is the source code. The next chapter covers system testing and analysis.

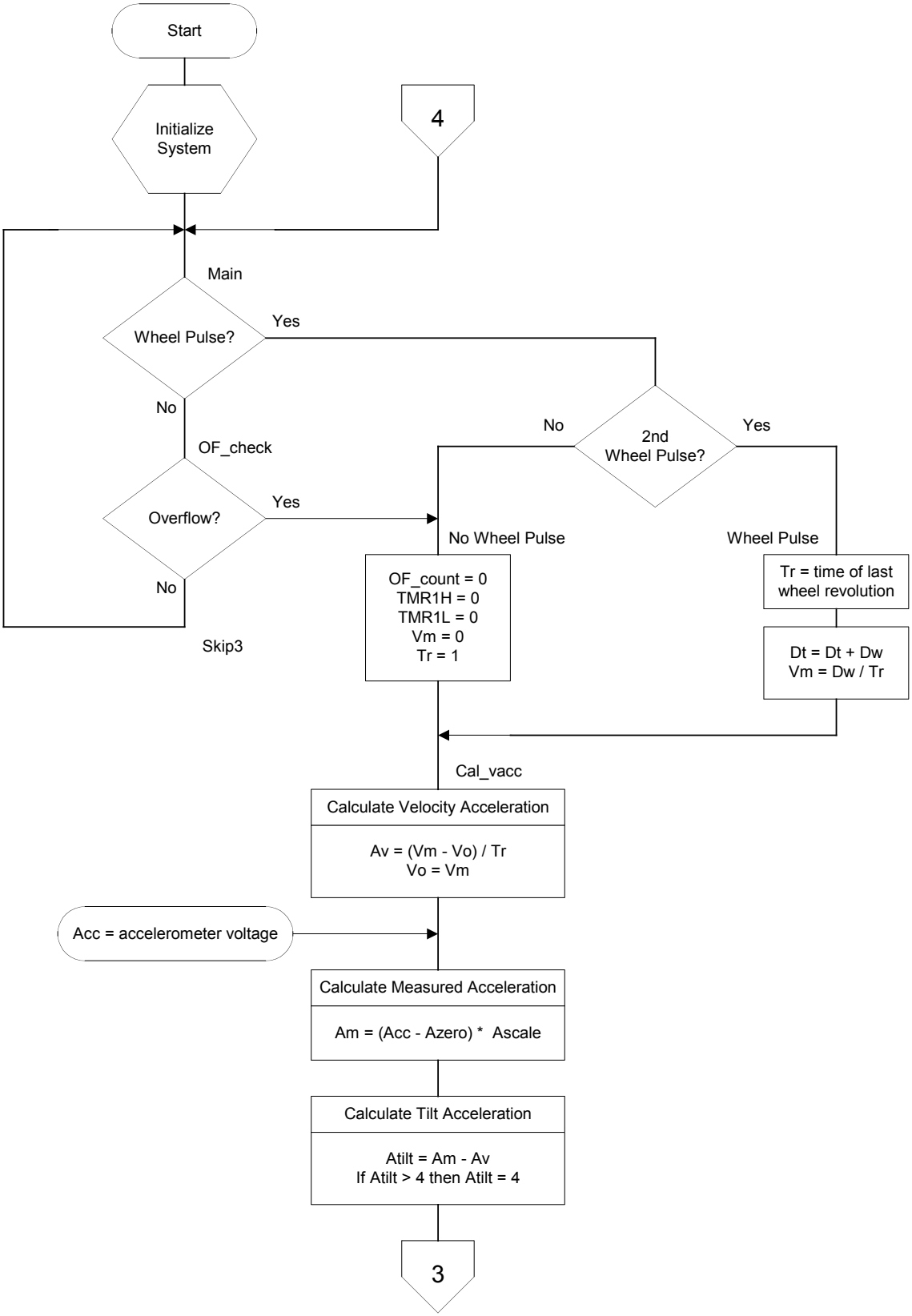


Figure 42: Main program flow chart upper half

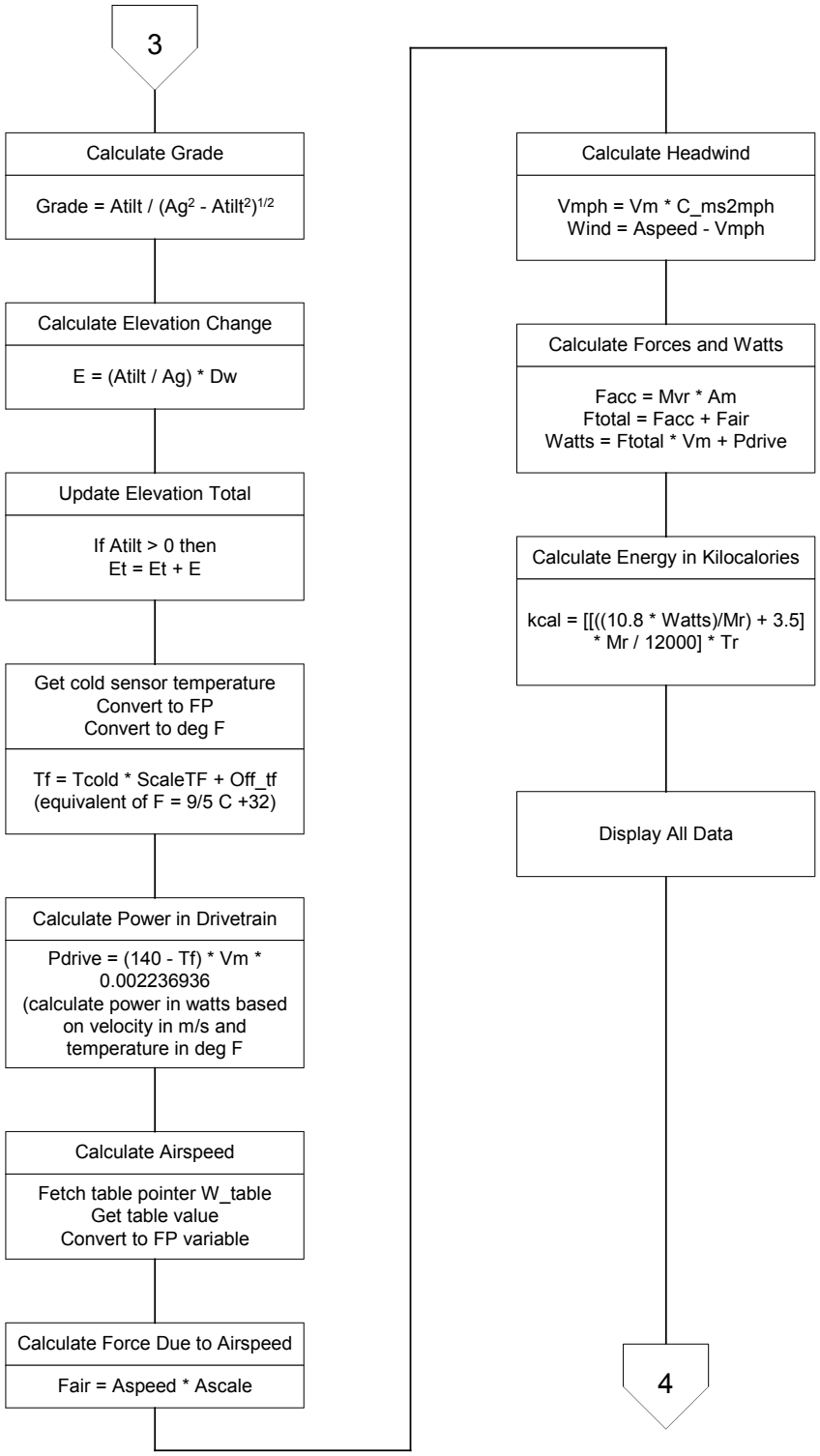


Figure 43: Main program flow chart lower half

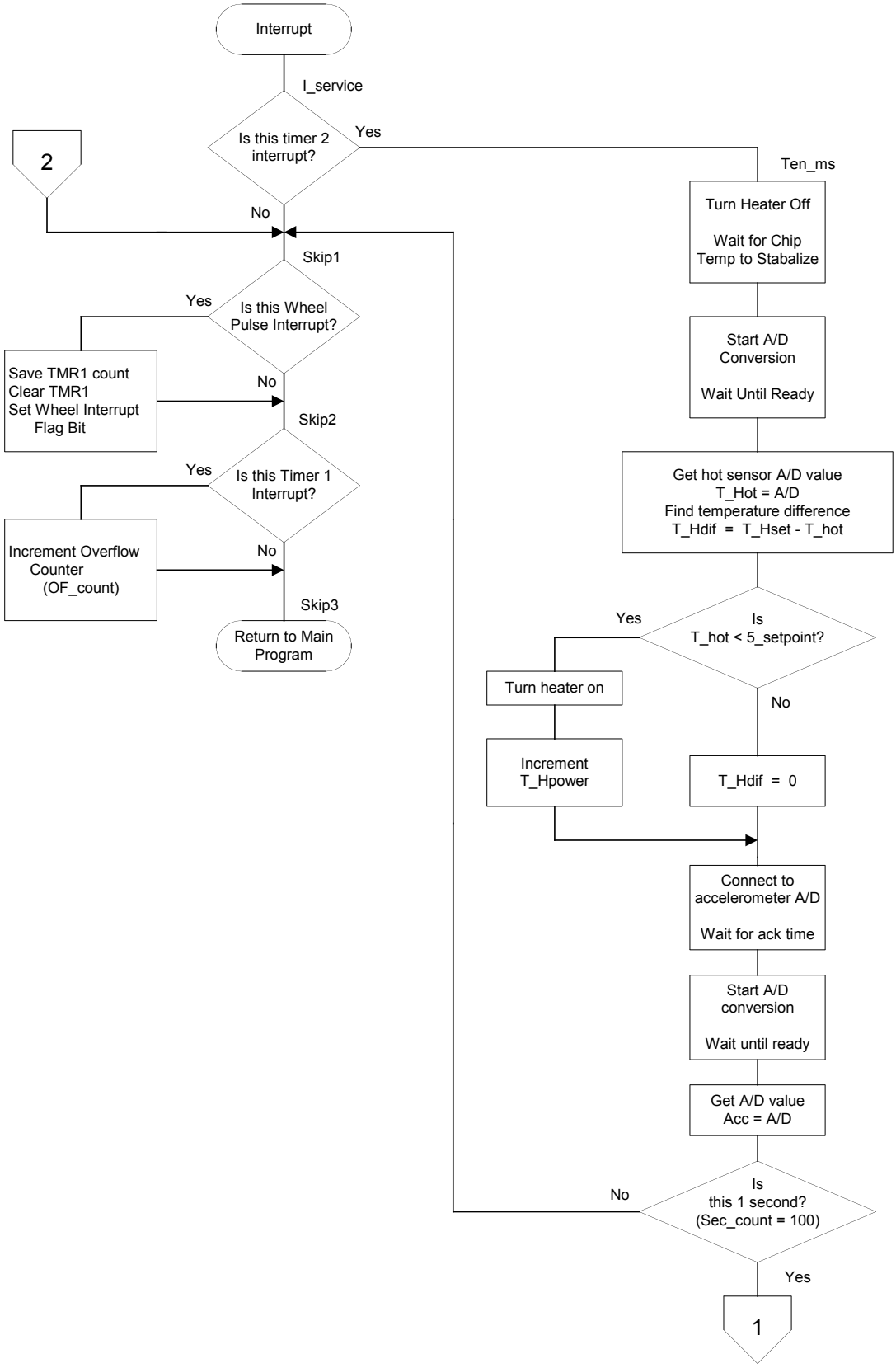


Figure 44: Interrupt program flow chart upper half

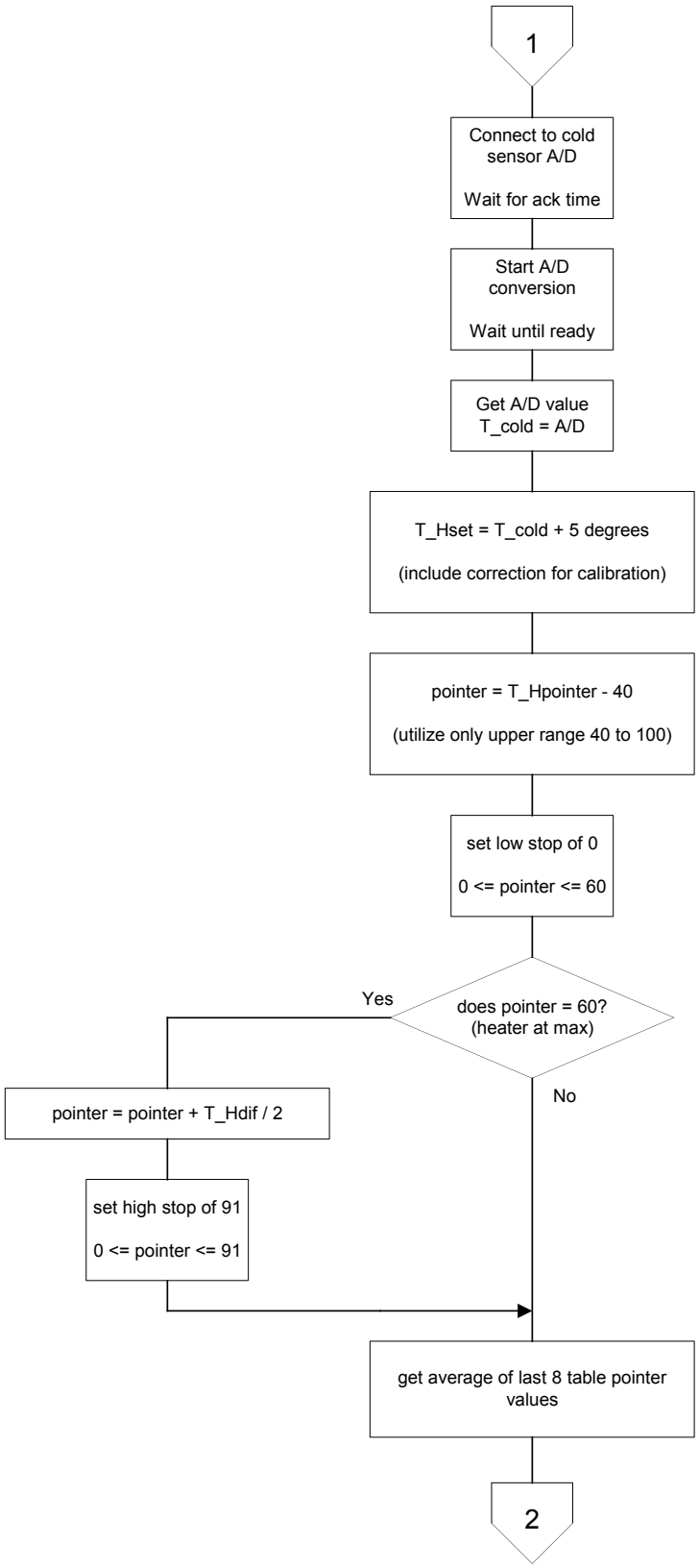


Figure 45: Interrupt program flow chart lower half

CHAPTER IV

ANALYSIS AND CONCLUSION

To validate the proper application of the individual elements, and to critique the overall system operation, requires a process of testing and evaluation. In this final chapter, the thesis overviews performance testing of the system with an evaluation and appraisal of the results. A summary is presented of the energy monitoring system from research and development to implementation and operation. As research efforts are never complete, the thesis concludes with a look at specific elements of the system that require further investigation.

4.1 Evaluation Testing

This section presents the results from a number of tests designed to measure particular aspects of the energy monitoring system. The tests are

Drivetrain Friction Test
Air Speed Friction Test
Static Accelerometer Test
Distance Measurement Test
Velocity Measurement Test
Isolated Air Speed Energy Test
Isolated Elevation Gain Energy Test
Isolated Velocity Increase Energy Test

4.1.1 Drivetrain Friction Test

A test was conducted to measure the drivetrain friction of the vehicle. This is a small but measurable component of the overall energy calculation. The typical bicycle has crank arms, pedals, front chain rings, and a rear derailleur assembly. While there is some friction in the crank bearing, the pedal bearings, and the wheel bearings, the main source of friction is in the drive chain and the associated components. The chain is routed around a front sprocket, through a pair of idler pulleys in the derailleur, and around a rear sprocket. The friction is largely due to the small radius of the sprockets and the 3 changes of direction in the rear derailleur assembly.

To measure the power loss in the drivetrain required measuring the force needed to rotate the crank axel and associated chain drive components at a speed equivalent to riding. A standard road bike was utilized for this test. One test included the rear wheel for comparison, while for the other tests the spokes tire and rim were removed from the rear hub. An electric drill was mounted with bearings in the axis of the drive chuck, such that the drill was free to rotate along this axis. In the drill chuck a swing arm was mounted that would apply pressure to the right pedal of the bicycle as shown in Figure 46. The swing axis of the electric drill was aligned to be concentric with the axis of the crank axel such that the swing arm could easily rotate the crank assembly. To measure the torque, a spring scale was attached to a lever arm of the electric drill. Since the drill was free to rotate, the torque required to rotate the crank was the same as the torque required to prevent rotation of the drill. The displacement distance is equal to the

equivalent circumference of the lever arm radius times the rotational speed. The power is equal to the force times the displacement velocity. Listed in Table X are the results of this test.

Figure 46: Picture of drivetrain test configuration

Condition	Temperature ºF	Temperature ºC	Crank Cadence RPM	Lever Radius cm	Equivalent Velocity m/s	Opposing Force Grams	Opposing Force Newtons	Power Watts
40/14 Gear, No Wheel	36	2.2	40	19.5	0.82	96	0.94	0.77
	36	2.2	50	19.5	1.02	107	1.05	1.07
	36	2.2	58	19.5	1.18	118	1.16	1.37
	36	2.2	70	19.5	1.43	118	1.16	1.65
	36	2.2	83	19.5	1.69	118	1.16	1.96
	36	2.2	90	19.5	1.84	121	1.19	2.18
	36	2.2	102	19.5	2.08	125	1.23	2.55
40/14 Gear, No Wheel	72	22.2	40	19.5	0.82	68	0.67	0.54
	72	22.2	51	19.5	1.04	71	0.70	0.73
	72	22.2	61	19.5	1.25	75	0.74	0.92
	72	22.2	70	19.5	1.43	75	0.74	1.05
	72	22.2	81	19.5	1.65	78	0.77	1.27
	72	22.2	90	19.5	1.84	78	0.77	1.41
	72	22.2	100	19.5	2.04	82	0.80	1.64
32/11 Gear, With Wheel	77	25	91	19.5	1.86	260	2.55	4.74

Table X: Drivetrain data

4.1.2 Air Speed Friction Test

This test is to determine the friction force coefficient of air speed. The friction force is the force opposing the air friction drag and can be determined by measuring the force needed to sustain forward motion at a constant speed. Since this force is proportional to the square of air speed [gil 94], the form of the calculation is

$$F\_air = F\_ascale \times (A\_speed)^2$$

A_speed = air speed
F_ascale = friction force coefficient
F_air = friction force due to air speed

To minimize unnecessary calculations in the embedded processor, A_speed is in mph, while F_air is in newtons.

The test consisted of a towing vehicle, a tow wire, a spring scale, and the test bicycle. The test was conducted 5 times and proved to be particularly difficult to perform. The first 3 tests were conducted using an automobile as the tow vehicle and were unsatisfactory. There was an oscillation between slack in the tow wire and a maximum spring scale reading. The last 2 tests were conducted with another bicycle as the tow vehicle and were somewhat more consistent. The test was repeated in the opposite direction to discount the effects of slope and wind speed. The test results are

12 mph south = 1200 g

12 mph north = 900 g

12 mph average = 1050 g

Yielding

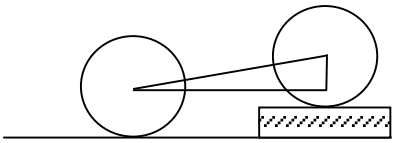
$$F\_ascale = 0.071506 \text{ newtons/(mph)}^2$$

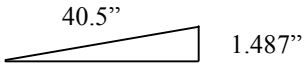
4.1.3 Static Accelerometer Test

This test was designed to validate the calibration of the accelerometer. A section of 2x6 wood spacer was placed under the rear and front wheel of the test vehicle to measure the slope or incline. The measured values were compared with the calculated slope.

Spacer = 1.487”

Wheel Base = 40.5”





Slope = $1.487/\sqrt{(40.5)^2 - (1.487)^2}$ = 3.674%

The test was performed in 2 directions to discount the slope of the test surface. The resolution of the display is 0.01% grade. Table XI lists 4 individual measurements corrected for the offset error, and the average of those 4 measurements.

Alignment	Condition	Display	Correction For Offset	Average % Grade	Calculated % Grade	Error
North	No Space	-0.06				
North	Space Rear	-3.88	3.82			
North	Space Front	3.82	3.88			
South	No Space	0.13				
South	Space Rear	-3.59	3.72			
South	Space Front	3.88	3.75	3.79	3.674	3.16%

Table XI: Accelerometer test

4.1.4 Distance Measurement Test

The distance measurement test was designed to validate the distance sensor and associated software. The test was conducted over a 1500' calibration course certified by USA Track & Field (USATF). The course is located at Lorain County Community College. The distance odometer readings are (calibrated in 0.01 miles)

Start = 0.01

Finish = 0.29

The front wheel of the test vehicle was rotated until the odometer incremented to 0.01 miles. The front wheel was then aligned to the start mark of the calibration course. The finish increment to 0.29 miles occurred approximately 10' before the finish mark on the calibration course. The total measured distance was

(0.28) x (5280) + 10' = 1488'

1488 / 1500 x 100 = 99.2% => 0.8% error

4.1.5 Velocity Measurement Test

The velocity measurement test was designed to validate the velocity sensor and associated software. The test was conducted between 1 mile distance marks on a Metropark bike path. The velocity of the vehicle between the marks was 12 mph and the start and finish times were recorded.

Start Time = 24:05

Finish Time = 29:12

307 / 300 x 100 = 102.3% => 2.3% error

4.1.6 Isolated Air Speed Energy Test

To validate the energy calculation based on air speed, a test was performed with both the drivetrain power calculation and the accelerometer power calculation disabled. By operating in this mode, the force of air speed friction becomes the only input to the kilocalorie measurement. The test was performed on the road with the direction of travel reversed at frequent intervals to minimize any influence due to wind. The energy for a 30 second interval was collected 20 times over 10 minutes at 5, 10, and 15 mph. Figure 47 shows the measurement data under these test conditions. The data is questionable since the values at 15 mph show lower energy than most of the 10 mph values. It was determined that the system battery voltage had decayed below the regulation limit causing improper operation. In particular, the drop in the 10 mph measurements in intervals 12-15 appears to indicate the actual point of failure, since the 15 mph measurements were collected after the 10 mph data. The test was repeated with a fresh battery and the results are shown in Figure 48.

To analyze the measurements more closely, the values of all data points were averaged over the three velocities and compared with an energy calculation based on force over distance. The results are shown in Table XII.

	Input Power	Output Power	
Velocity	Energy Monitor Measurement	Energy Calculation Force x Distance	Efficiency
5	18.9 kcal	0.572 kcal	3.0%
10	43.2 kcal	4.577 kcal	10.6%
15	77.9 kcal	15.45 kcal	19.8%

Table XII: Isolated air speed energy calculation

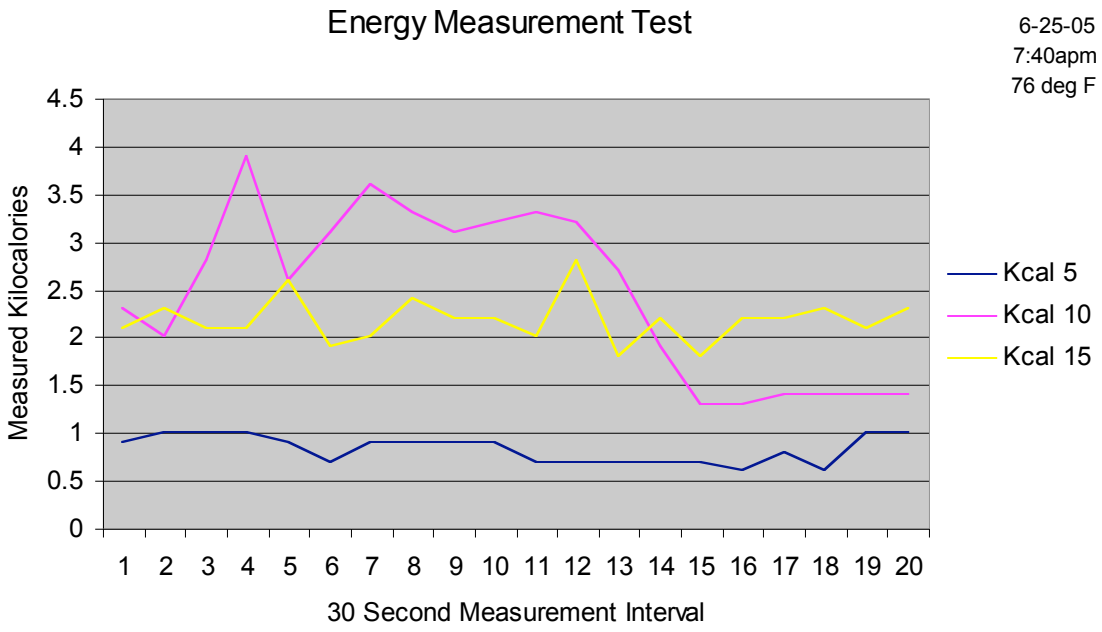


Figure 47: Air speed interval measurement 6/25/05

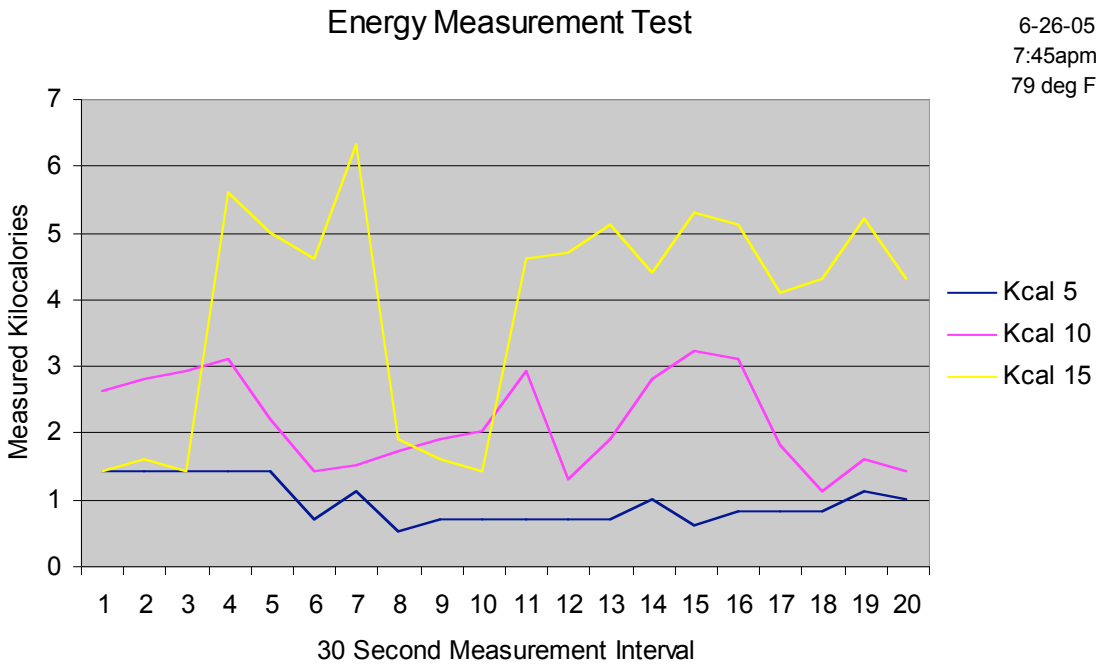


Figure 48: Air speed interval measurement 6/26/05

For comparison, the human body is 20-23% efficient [plo 03] in producing output power from calories expended. This table shows good correlation at 15 mph between the input energy from burning calories to output energy in the form of work performed. The lower efficiency at 5 and 10 mph is due to the fixed term (3.5 $mLkg^{-1}min^{-1}$) representing leg cycling movement at 60 rpm in the leg ergometry formula.

4.1.7 Isolated Elevation Gain Energy Test

To validate the energy calculation based on elevation gain, a test was performed with both the drivetrain power calculation and the air speed power calculation disabled. In this mode, the force accelerating the mass of the vehicle and the rider is the only input to the kilocalorie measurement. This acceleration can be due to either increasing the velocity of the vehicle or an increase in potential energy. In this case it is the increase in potential energy that is to be measured. To isolate the measurement as much as possible, the test measurement was conducted with the vehicle in motion at the beginning of the test and the velocity was held constant.

The test was conducted on an underpass where the measured vertical distance from the bridge surface to the park road below was 8.76 meters. The park drive sloped up on either side of the bridge to the same level as the road above. The north slope was steeper than the south slope and this difference was included in the measurement data. To ensure accuracy, the alignment of the accelerometer to the bike frame was checked before and after this test. The test was conducted at 5 and 10 mph. Since the increase in potential

energy is the same regardless of the slope or the velocity, all 4 measurements should measure the same energy. Table XIII shows the results.

	Input Power	Output Power	
Velocity & Direction	Energy Monitor Measurement	Energy Calculation Ep = mgy	Efficiency
5 N	7.7 kcal	1.73 kcal	22.5%
5 S	8.3 kcal	1.73 kcal	20.8%
10 N	7.5 kcal	1.73 kcal	23.1%
10 S	7.7 kcal	1.73 kcal	22.5%

Table XIII: Isolated potential energy calculation

4.1.8 Isolated Velocity Increase Energy Test

The last test is a validation of the energy calculation based on actual vehicle acceleration. Again both the drivetrain power calculation and the air speed power calculation were disabled for this test. In this mode, the force of accelerating the mass of the vehicle and the rider is the only input to the kilocalorie measurement. This test measures the acceleration due only to an increase in the velocity of the vehicle.

The test was conducted in two directions to take into account any slope in the road surface. The test measured the energy required to accelerate the vehicle from a dead stop to 5, 10, and 15 mph. In addition, the test was conducted at an approximation of three different acceleration rates: slow, medium, and fast. There were two measurements for each set of conditions. Figure 49 shows the results of the isolated acceleration test.

An average of the data for the two measurements is compared to an energy calculation based on an increase in potential energy. This results are shown in Table XIV.

Velocity Acceleration	Input Power Energy Monitor Measurement	Output Power Energy Calculation $E_k = ½ mv^2$	Efficiency
5 Slow	0.3 kcal	0.05 kcal	16.7%
5 Medium	0.25 kcal	0.05 kcal	20.0%
5 Fast	0.63 kcal	0.05 kcal	7.9%
10 Slow	1.05 kcal	0.20 kcal	19.0%
10 Medium	0.98 kcal	0.20 kcal	20.4%
10 Fast	1.4 kcal	0.20 kcal	14.3%
15 Slow	2.0 kcal	0.45 kcal	22.5%
15 Medium	2.2 kcal	0.45 kcal	20.5%
15 Fast	2.8 kcal	0.45 kcal	16.1%

Table XIV: Isolated kinetic energy calculation

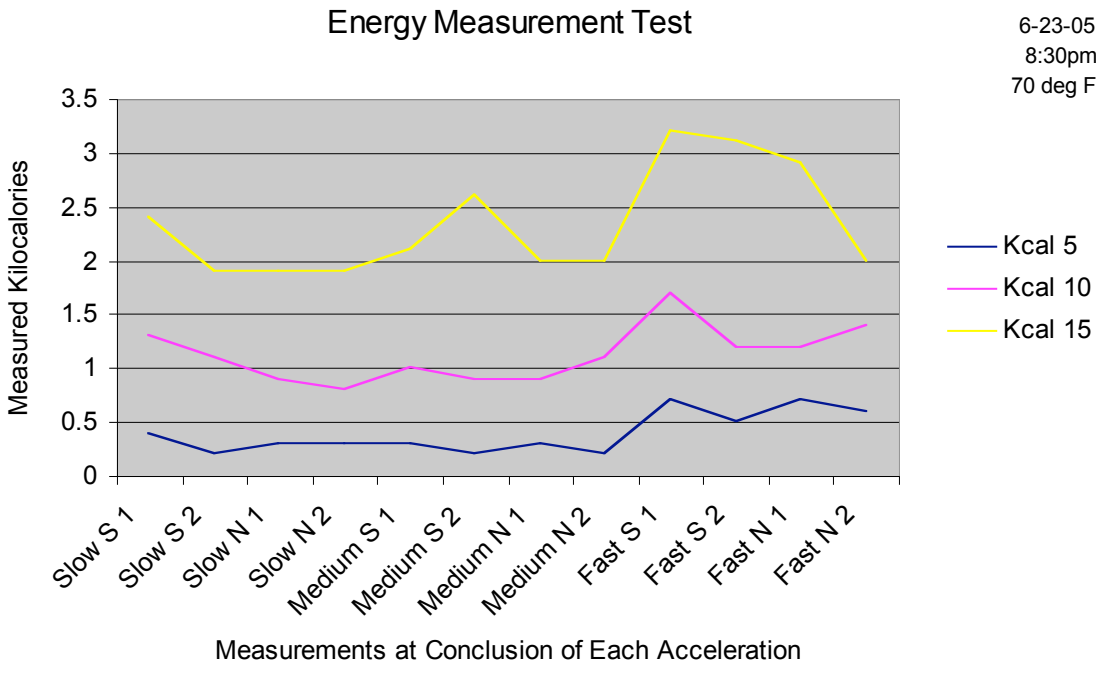


Figure 49: Velocity change energy measurement

4.2 Conclusion

This thesis has described an energy monitor for a human powered vehicle. The design objective was a device that would not require modification of the vehicle or attachments to the body. Accordingly, the sensors monitor air speed and acceleration to indirectly determine the exercise power level. Established relationships in exercise physiology translate this power level to a prediction of calorie consumption for the exercise interval.

The device incorporates three sensors and a microcomputer to gather and process the measurement data. The air speed sensor uses hot wire anemometer technology and is designed for low power operation. The accelerometer sensor captures the power to accelerate and to elevate the vehicle mass. A third sensor monitors the velocity and distance traveled. Together with a microcomputer, the sensors monitor the exercise intensity and determine the kilocalories consumed. This value is calculated and updated continuously on an LCD display.

A significant amount of data was collected to validate this measurement approach and to test the system accuracy. One of the liabilities of air speed measurement is the variability due to wind gusts. The data however indicates a high correlation to ground speed when averaged over extended periods. While a similar device for a completely independent comparison was not available, the sensors were tested individually, and the kilocalorie measurement was compared against the calculated energy output. The results

demonstrate that this system approach is a viable measurement technique that has considerable usefulness in human powered navigation and exercise.

4.3 Future Work

This thesis has a number of areas that deserve further study. Many of these issues relate to a tradeoff between accuracy and simplicity. A major circuit simplification would be the elimination of the channel amplifiers. This could be done by the use of higher bit count A/D converters. In addition, higher sensitivity accelerometers would also be helpful. Related to this is how much A/D resolution is actually required.

The air friction test is inaccurate and needs to be repeated in a wind tunnel or with an automated measurement process. In addition, multiple air friction coefficients need to be collected for different types of vehicles and rider positions. This system did not need correction for temperature sensor slope error. However, some devices could require calibration at individual temperatures with an additional look-up table. Finally, the physical size needs to be greatly reduced with smaller components. This potentially has the additional benefit of lower power consumption if the temperature sensor package size can be reduced.

BIBLIOGRAPHY

[ame 00] American College of Sports Medicine. ACSM'S Guidelines for Exercise Testing and Prescription. Philadelphia: Lippincott Williams & Wilkins, 2000.

[ame 01a] American College of Sports Medicine. ACSM'S Health & Fitness Certification Review. Philadelphia: Lippincott Williams & Wilkins, 2001.

[ame 01b] American College of Sports Medicine. ACSM'S Resource Manual for Guidelines for Exercise Testing and Prescription. Philadelphia: Lippincott Williams & Wilkins, 2001.

[ana 04] Analog Devices. ADXL320 Data Sheet. www.analog.com. 2004.

[bev 97] Beversdorff, M., W. Forster, R. Schodl, and H. W. Jentink. In-flight Laser Anemometry for Aerodynamic Investigation on an Aircraft" Optics and Lasers in Engineering 27 (1997): pg. 571-586.

[cos 03] Costanzo, Linda S. Physiology. Baltimore: Lippincott Williams & Wilkins, 2003.

[gau 89] Gaudin, Anthony J., Kenneth C. Jones, James G. Cotanche, and Josephine Ryan. Human Anatomy and Physiology. San Diego: Harcourt Brace Javanovich, 1989.

[gil] Giles, R. V., J. B. Evett, and C. Liu. Fluid Mechanics and Hydraulics. New York:McGraw-Hill, 1994.

[his 95] Hsieh, H. Y., and Jay N. Zemel. “Pyroelectric Anemometry: Frequency, Geometry and Gas Dependence”. Sensors and Actuators 49 (1995): pg. 133-140.

[inc 96] Incropera, Frank P. and David P. DeWitt. Fundamentals of Heat and Mass Transfer. New York: John Wiley & Sons, 1996.

[jor 96] Jorgensen, F. E. “The Computer-controlled Constant-temperature Anemometer. Aspects of Set-up, Probe Calibration, Data Acquisition and Data Converson.” Measurement Science Technology 7 (1996): pg. 1378-1387.

[keg 00] Kegerise, M. A., and E. F. Spina. “A Comparative Study of Constant-voltage and Constant-temperature Hot-wire Anemometers.” Experiments in Fluids 29 (2000): pg. 154-164.

[kit 95] Kitchin, Charles. “Using Accelerometers in Low g Applications.” Analog Devices Application Note AN-374. www.analog.com. 1995.

[kri 99] Kristensen, L. The Perennial Cup Anemometer. Wind Energy 2 (1999): pg. 59-75.

[lee 97] Lee, S. P., and S. Ken Kauh. “A New Approach to Enhance the Sensitivity of a Hot-wire Anemometer and Static Response Analysis of a Variable-temperature Anemometer.” Experiments in Fluids 22 (1997): pg. 212-219.

[mar 02] Martin, D. P., J. J. Grant, and J. V. Ringwood. “Evaluation of a Prototype Thermal Anemometer for use in Low Airspeed Drying Measure Calculations.” Flow Measurement and Instrumentation 12 (2002): pg. 385-396.

[mar 71] Martino, Francisco S. “A Thermister Anemometer for the Measurement of Very Low Air Velocities” The Review of Scientific Instruments (1971): Volume 42

Number 5

[nat 00] National Seminconductor LM335 Data Sheet. www.national.com 2000.

[plo 03] Plowman, Sharon A. and Denise L. Smith. Exercise Physiology for Health, Fitness, and Performance. San Francisco: Benjamin Cummings, (2003).

[sco 74] Scott, Allan W. Cooling of Electronic Equipment. New York: John Wiley & Sons, (1974).

[sea 70] Sears, Francis Weston, and Mark W. Zemansky. University Physics. Reading: Addison-Wesley, (1970).

[shu 95] Shuster, Mike, Bob Briano, and Charles Kitchin. "Mounting Considerations for ADXL Series Accelerometers." Analog Devices Application Note AN-379. www.analog.com. 1995.

[sto 02] Stock, Chris. "WindSonic – A New Ultrasonic Anemometer." Sensor Review (2002): Volume 22, Number 3, pg. 218-222.

APPENDIX A

SOFTWARE LISTING

```
;*************************************************************
; Calorie.asm
; Gary Siegmund
; Cleveland State University
;
;
;
;
;
;
;
;
;
;*************************************************************

      list p=16f877

#define P16_MAP1 0            ; don't use map 1
#define P16_MAP2 1            ; do use map 2

include "p16f877.inc"

      nolist
            include "math16.inc"    ;constants and varible definitions
            include "MATH16.MAC"
      list




;*************************************************************
; Macro definitions

push  macro
      movwf   WTemp           ; save W
      swapf   STATUS,W        ; get STATUS in W without affecting
STATUS
      movwf   StatusTemp      ; save STATUS
      movf    PCLATH,W        ; get PCLATH
      movwf   PclathTemp      ; save PCLATH
      clrf    STATUS          ; set to bank 0
      clrf    PCLATH          ; set to page 0
      endm

pop   macro
      movf    PclathTemp,W    ; get PCLATH
      movwf   PCLATH          ; restore PCLATH
      swapf   StatusTemp,W    ; unswap STATUS nibbles into W
      movwf   STATUS          ; restore STATUS
      swapf   WTemp,F         ; swap W nibbles, keep in file register
      swapf   WTemp,W         ; restore W without affecting STATUS
      endm
```

```
;*****************************************************
; User-defined variables

      cblock  0x60            ;           bank 0 assignnments
              ASCII4          ; reserve five bytes of data RAM for each digit
              ASCII3          ;
              ASCII2          ;
              ASCII1          ;
              ASCII           ;

              digit_count     ; counter used to cycle through each digit
              OF_count        ; overflow counter,
                              ; used as a timer 1 overflow counter
              C_bits          ; used as a register to hold various
                              ; status control bits
                              ;     bit 0 wheel pulse
                              ;     bit 1 A_tilt test utility
                              ;     bit 2 one second flag
                              ;     bit 3 previous wheel pulse
                              ;     bit 4
                              ;     bit 5
                              ;     bit 6 display toggle
                              ;     bit 7       "
              T1H             ; used to save previous timer 1 value
              T1L             ;     "     "     "     "
              N_char          ; used to set number of characters to display
              Temp            ; temporary variable
      endc

      cblock  0x78            ;           bank 0 assignnments
              WTemp           ; for push and pop
              StatusTemp      ;     "     "
              PclathTemp      ;     "     "
      endc

                              ; start at 0xA3   A_tilt-3  EXP  0xA0
                              ;                 A_tilt-2  B0   0xA1
                              ;                 A_tilt-1  B1   0xA2
                              ;                 A_tilt    B2   0xA3
                              ;
      cblock  0xA3            ;           bank 1 assignnments
              A_tilt:4        ; acceleration representing tilt
              A_g:4           ; acceleration of gravity
              A_g2:4          ;     "     "     "   squared
              A_m:4           ; measured frame acceleration
              A_scale:4       ; calibration constant for accelerometer
              C_100:4         ; constant  FP value of 100
              A_v:4           ; acceleration due to change in velocity
              A_zero:4        ; calibration zero reference contstant
              D_w:4           ; calibration constant - dist wheel rev
              D_t:4           ; total distance traveled
              E:4             ; last elevation change
              E_t:4           ; total elevation change
              T_r:4           ; wheel index time
```

```
        V_m:4           ; measured velocity
        V_o:4           ; previous velocity
        Grade:4         ; % grade

        T1_scale:4      ; time scale correction factor = 1x10^-6
        Eight:4         ; correction for 1:8 prescale in timer 1
        C_00223:4       ; constant 0.002236936 for P_drive calculation
        C_140:4         ; constant   140       "     "         "
endc


cblock  0x113           ;           bank 2 assignnments
        Scale_TF:4      ; convert from C to F scale factor
        Off_TF:4        ; 32 degree add to convert from C to F
        C_ms2mph:4      ; constant  m/s to mph
        C_m2mi:4        ;     "     meters to miles
        C_kcal1:4       ;     "     10.8
        C_kcal2:4       ;     "     3.5
        C_kcal3:4       ;     "     1/12000
        M_v:4           ; mass of vehicle
        M_r:4           ; mass of rider
        M_vr:4          ; mass of vehicle and rider
        V_mph:4         ; velocity in mph
        A_speed:4       ; air speed in mpg
        F_ascale:4      ; conversion factor
                        ;      from airspeed to force in newtons
        F_air:4         ; force due to air friction in newtons
        Wind:4          ; head wind in mph
        K_cal:4         ; accumulated kilo calories burned
                        ;      in this riding interval
        P_drive:4       ; power due to drive friction
        T_f:4           ; temperature in deg F

        Ten_h           ; 10 hour counter
        One_h           ; 1 hour counter
        Ten_m           ; 10 minute counter
        One_m           ; 1 minute counter
        Ten_s           ; 10 second counter
        One_s           ; 1 second counter
        Sec_count       ; 100 counter from 10ms to 1s intervals
        T_HsetH         ; temperature set point for hot sensor
        T_HsetL         ;     "     "     "     "
        T_HdifH         ; temperature difference from set point to actual T
        T_HdifL         ;     "     "     "     "
        T_Hpower        ; power in hot sensor
        W_table         ; table pointer
        W_tablea        ; table pointer
        W_tableH        ; table pointer
        W_tableL        ; table pointer
        Count           ; counter
        Acc_H           ; accelerometer A/D reading
        Acc_L           ;     "     "     "
        T_coldH         ; cold sensor A/D reading
        T_coldL         ;     "     "
        T_hotH          ; hot sensor A/D reading
        T_hotL          ;     "     "
endc
```

```
      cblock  0x190           ;           bank 3 assignnments
              Table:91        ; air speed look-up table
      endc                    ;

;*********************************************************
; Start of executable code

      org     0x000
      nop
      goto    Initialize


;*********************************************************
; Interrupt vector

      org     0x004
      push                    ; save W, STATUS, and PCLATH in temp data registers
      goto    I_service       ; jump to the interrupt service routine


;*********************************************************
; Initialize System

Initialize
      banksel 0               ;0 **** initialize A/D converter ****
      movlw   B'01000001'     ; Fosc/8, A/D enabled
      movwf   ADCON0          ;
      banksel PIE1            ;1
      movlw   B'10000010'     ; A/D data right justified, 3 analog channels
      movwf   ADCON1          ; VDD and VSS references

      banksel PIE1            ;1 ****** initialize serial port *******
                              ;      (available for testing if needed)
      movlw   D'12'           ; This sets the baud rate to 19200
      movwf   SPBRG           ; assuming BRGH=1 and Fosc=4.000 MHz
      banksel 0               ;0
      bsf     RCSTA, SPEN     ; Enable the serial port
      banksel PIE1            ;1
      bcf     TXSTA, SYNC     ; Set up the port for asynchronous operation
      bsf     TXSTA, TXEN     ; Transmit enabled
      bsf     TXSTA, BRGH     ; High baud rate

      movlw   B'00000000'     ; reset option
      movwf   OPTION_REG      ;

      banksel EECON1          ;3
      clrf    EECON1          ; reset EECON1
      banksel PIE1            ;1

                              ; ***** initialize I/O port states *****
      movlw   B'00000111'     ;
      movwf   TRISA           ;
      movlw   B'11001001'     ;
      movwf   TRISB           ;
      movlw   B'11000000'     ;
```

```
        movwf   TRISC           ;
        movlw   B'00000000'     ;
        movwf   TRISD           ;
        movlw   B'00000000'     ;
        movwf   TRISE           ;
        banksel 0               ;0
        bsf     PORTA,3         ; set power bit on,
                                ; change TRISA to toggle this bit on/off


                                ; ******* initialize timer 1 & 2 *******
        banksel PIE1            ;1
        movlw   B'00000011'     ; timer 1 & 2 interrupt enable
        movwf   PIE1            ;
        movlw   B'01010000'     ; enable peripheral,
                                ;  and RB0 interrupts, leave global off
        movwf   INTCON          ;

        banksel 0               ;0
        movlw   B'00110001'     ; timer 1 prescale = 1:8
        movwf   T1CON           ;
        movlw   B'01001101'     ; timer 2 prescale = 1:4, postscale = 1:10
        movwf   T2CON           ;
        banksel PIE1            ;1
        movlw   D'250'          ; timer 2 match = 250 x 4 x 10  >
                                ;  interrupt every 10mS
        movwf   PR2             ;

        bsf     PCLATH,3        ; select page 1
        call    Init_FP         ; * initialize floating point variables *
        bcf     PCLATH,3        ; select page 0

        banksel 0               ;0
        movlw   B'01000000'     ; initialize C_bits
        movwf   C_bits          ;
        clrf    OF_count        ;

        call    LCD_init        ; initialize the LCD display
        call    LCD_labels      ; display the LCD fixed labeling

        bsf     INTCON,GIE      ; enable interrupts
```

```
;*****************************************************************
;*****************************************************************
;*****************************************************************
; Main Routine     the next 10 lines of code are the main program loop
control

Main
```

```
      banksel 0               ;0
      btfss   C_bits,0        ; is there a wheel pulse?
      goto    OF_check        ; NO, check for OF = overflow

      btfsc   C_bits,3        ; was there a last wheel pulse?
      goto    Wheel_pulse     ; YES, goto start of time calculation
      bsf     C_bits,3        ; set last WP bit
      goto    No_wheelpulse   ;

OF_check
      btfss   OF_count,1      ; is there an OF_count overflow?
      goto    Main            ; NO, goto wait loop
      bcf     C_bits,3        ; reset last WP bit

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

No_wheelpulse
      clrf    OF_count        ;   " if timed out, then reset counter
      clrf    TMR1H           ;   "
      clrf    TMR1L           ;   "

      banksel PIE1            ;1   "
      clrf    V_m             ;   "       Vm = 0
      clrf    V_m-1           ;   "
      clrf    V_m-2           ;   "
      clrf    V_m-3           ;   "

      movlw   0x7F            ;   "       Tr = 1 second
      movwf   T_r             ;   "
      clrf    T_r-1           ;   "
      clrf    T_r-2           ;   "
      clrf    T_r-3           ;   "
      banksel 0               ;0   "

      goto  Cal_vacc          ;   " branch to
                              ; "update wheel acceleration"



;
*********************************************************************
; calculate time of last wheel revolution

Wheel_pulse
      movf    OF_count,W      ;
      movwf   AARGB0          ; load highest byte with overflow count
      clrf    OF_count        ;
      movf    T1H,W           ;
      movwf   AARGB1          ; get Timer1 high byte
      movf    T1L,W           ;
      movwf   AARGB2          ; get Timer1 low byte
      call    FLO2432         ; convert to FP

      movlw   T1_scale        ; convert from microseconds to seconds
      call    To_BARG         ; and correct for prescaler 1:8
```

```
      call    FPM32           ;

      movlw   T_r             ; put final result back in Tr
      call    AARG_To         ;


;
*********************************************************************
; update distance and velocity

      movlw   D_t             ;     Dt = Dt + Dw
      call    To_AARG         ;
      movlw   D_w             ;
      call    To_BARG         ;
      call    FPA32           ;
      movlw   D_t             ;
      call    AARG_To         ;

      movlw   D_w             ;     Vm = Dw / Tr
      call    To_AARG         ;
      movlw   T_r             ;
      call    To_BARG         ;
      call    FPD32           ;
      movlw   V_m             ;
      call    AARG_To         ;



;
*********************************************************************
; calculate velocity acceleration

Cal_vacc
      movlw   V_m             ;     Av = (Vm - Vo) / Tr
      call    To_AARG         ;
      movlw   V_o             ;
      call    To_BARG         ;
      call    FPS32           ;
      movlw   T_r             ;
      call    To_BARG         ;
      call    FPD32           ;
      movlw   A_v             ;
      call    AARG_To         ;

      banksel PIE1            ;1
      movf    V_m,W           ;     Vo = Vm
      movwf   V_o             ;
      movf    V_m-1,W         ;
      movwf   V_o-1           ;
      movf    V_m-2,W         ;
      movwf   V_o-2           ;
      movf    V_m-3,W         ;
      movwf   V_o-3           ;


;
*********************************************************************
```

```
; calculate measured acceleration

      banksel 0               ;0    Acc = A/D converter result
      clrf    AARGB0          ;
      banksel EEADR           ;2
      movf    Acc_H,W         ;
      banksel 0               ;0
      movwf   AARGB1          ;
      banksel EEADR           ;2
      movf    Acc_L,W         ;
      banksel 0               ;0
      movwf   AARGB2          ;
      call    FLO2432         ; convert to FP

      movlw   A_zero          ;     Am = (Acc - Azero) * Ascale
      call    To_BARG         ;
      call    FPS32           ;
      movlw   A_scale         ;
      call    To_BARG         ;
      call    FPM32           ;
      movlw   A_m             ;
      call    AARG_To         ;


;
*************************************************************************
; calculate tilt acceleration

      movlw   A_v             ;     Atilt = Am - Av
      call    To_BARG         ;
;     call    FPS32           ;  commented out for test
      movlw   A_tilt          ;
      call    AARG_To         ;

      bcf     C_bits,1        ; safety check,   *
                              ;      if Atilt > 4 then make Atilt = 4
      btfsc   AARG,7          ;           *
      bsf     C_bits,1        ;           *
      bsf     AARG,7          ;           *
      movlw   Eight           ;           *
      call    To_BARG         ;           *
      decf    BEXP,F          ; change 8 to 4   *
      call    FPA32           ;           *
      btfss   AARG,7          ;           *
      goto    OK              ;           *
      movlw   Eight           ;           *
      call    To_AARG         ;           *
      decf    AEXP,F          ; change 8 to 4   *
      movlw   A_tilt          ;           *
      call    AARG_To         ;           *
OK    movlw   A_tilt          ;           *
      call    To_AARG         ;           *
      btfsc   C_bits,1        ;           *
      bsf     AARG,7          ;           *
      movlw   A_tilt          ;           *
      call    AARG_To         ;           *
```

```
;
*********************************************************************
; calculate grade

      movlw   A_tilt          ; Grade = 100 x Atilt / Ag - (32 x (Atilt/Ag)^4)
      call    To_AARG         ;
      movlw   A_g             ;
      call    To_BARG         ;
      call    FPD32           ;
      movlw   AEXP            ;
      call    To_BARG         ;
      call    FPM32           ;
      movlw   AEXP            ;
      call    To_BARG         ;
      call    FPM32           ;
      movlw   0x84            ;
      movwf   BEXP            ;
      clrf    BARGB0          ;
      clrf    BARGB1          ;
      clrf    BARGB2          ;
      call    FPM32           ;
      bsf     AARG,7          ; change to negative number for subtraction
      movlw   A_g             ;
      call    To_BARG         ;
      call    FPA32           ;
      movlw   AEXP            ;
      call    To_BARG         ;
      movlw   A_tilt          ;
      call    To_AARG         ;
      call    FPD32           ;
      movlw   C_100           ;
      call    To_BARG         ;
      call    FPM32           ;
      movlw   Grade           ;
      call    AARG_To         ;


;
*********************************************************************
; calculate elevation change

      movlw   A_tilt          ;      E = (Atilt / Ag) x Dw
      call    To_AARG         ;
      movlw   A_g             ;
      call    To_BARG         ;
      call    FPD32           ;
      movlw   D_w             ;
      call    To_BARG         ;
      call    FPM32           ;
      movlw   E               ;
      call    AARG_To         ;


;
*********************************************************************
; update elevation total
```

```
      banksel PIE1            ;1
      btfsc   A_tilt-1,7      ;    If Atilt < 0 then don't add to elevation total
      goto    No_add          ;

      banksel 0               ;0
      btfss   C_bits,0        ;    If no wheel pulse, don't add
      goto    No_add          ;

      movlw   E_t             ;     Et = Et + E
      call    To_BARG         ;
      call    FPA32           ;
      movlw   E_t             ;
      call    AARG_To         ;
No_add                        ;
      banksel 0               ;0
      bcf     C_bits,0        ;


;*****************************************************************
**
; get cold sensor temperature, convert to deg F

      banksel 0               ;0
      clrf    AARGB0          ; clear this byte
      banksel EEADR           ;2
      movf    T_coldH,W       ; ADRESH in AARGB1
      banksel 0               ;0
      movwf   AARGB1          ;
      banksel EEADR           ;2
      movf    T_coldL,W       ; ADRESL in AARGB2
      banksel 0               ;0
      movwf   AARGB2          ;
      call    FLO2432         ; convert to FP

      movlw   Scale_TF        ; get scale factor
      bsf     STATUS,IRP      ; set indirect to bank 2,3
      call    To_BARG         ;
      bcf     STATUS,IRP      ; set indirect to bank 0,1
      call    FPM32           ; convert to F scale

      movlw   Off_TF          ; add 32 to get deg F
      bsf     STATUS,IRP      ; set indirect to bank 2,3
      call    To_BARG         ;
      bcf     STATUS,IRP      ; set indirect to bank 0,1
      call    FPA32           ;

      movlw   T_f             ; save temperature
      bsf     STATUS,IRP      ;
      call    AARG_To         ;
      bcf     STATUS,IRP      ;


;*********************************************************
; Calculate power in drive train

      bsf     AARG,7          ; Pdrive = (140 - deg F) * Vm * 0.002236936
```

```
      movlw   C_140           ;
      call    To_BARG         ;
      call    FPA32           ;

      movlw   V_m             ;
      call    To_BARG         ;
      call    FPM32           ;

      movlw   C_00223         ;
      call    To_BARG         ;
      call    FPM32           ;

      movlw   P_drive         ; save drive train power
      bsf     STATUS,IRP      ;
      call    AARG_To         ;
      bcf     STATUS,IRP      ;


;*********************************************************
; Calculate Airspeed

      banksel EEADR           ;2
      movf    W_table,W       ; put air speed pointer in EEPROM address register
      movwf   EEADR           ;
      banksel EECON1          ;3
      bsf     EECON1,RD       ; initiate a read operation
      banksel EEADR           ;2
      movf    EEDATA,W        ; get table value
      banksel 0               ;0

      clrf    AARGB0          ; save as FP variable
      clrf    AARGB1          ;
      movwf   AARGB2          ;
      call    FLO2432         ; convert to FP variable

      movlw   Eight           ; scale table value
      call    To_BARG         ;
      call    FPD32           ;

      movlw   A_speed         ; store airspeed
      bsf     STATUS,IRP      ;
      call    AARG_To         ;
      bcf     STATUS,IRP      ;


 ;*********************************************************
; Calculate Force Due to Airspeed

      movlw AEXP              ; get air speed squared
      call  To_BARG           ;
      call  FPM32             ;

      movlw F_ascale          ; scale to a force in newtons
      bsf   STATUS,IRP        ;
      call  To_BARG           ;
      bcf   STATUS,IRP        ;
      call  FPM32             ;
```

```
      movlw F_air             ; save force due to airspeed
      bsf   STATUS,IRP        ;
      call  AARG_To           ;
      bcf   STATUS,IRP        ;


;*********************************************************
; Calculate Headwind

      movlw   V_m             ; get velocity in meters/second
      call    To_AARG         ;
      movlw   C_ms2mph        ; convert to miles per hour
      bsf     STATUS,IRP      ;
      call    To_BARG         ;
      bcf     STATUS,IRP      ;
      call    FPM32           ;

      movlw   V_mph           ; save Vmph
      bsf     STATUS,IRP      ;
      call    AARG_To         ;
      bcf     STATUS,IRP      ;

      movlw   AEXP            ; move to BARG
      call    To_BARG         ;

      movlw   A_speed         ; retrive airspeed
      bsf     STATUS,IRP      ;
      call    To_AARG         ;
      bcf     STATUS,IRP      ;
;     call    FPS32           ; calculate headwind
                              ;    commented out for test
      movlw   Wind            ; save headwind
      bsf     STATUS,IRP      ;
      call    AARG_To         ;
      bcf     STATUS,IRP      ;


;*********************************************************
; Calculate Force Due to Acceleration, add forces, calculate Watts

      movlw   A_m             ; get measured acceleration in m/ss
      call    To_AARG         ;

      movlw   M_vr            ; get mass of loaded vehicle in Kg
      bsf     STATUS,IRP      ;
      call    To_BARG         ;
      bcf     STATUS,IRP      ;
      call    FPM32           ; this is force in newtons    F=ma
                              ; from:
                              ; the acceleration due to velocity change
                              ; the acceleration due to gravity

      movlw   F_air           ; retrive force due to air friction
      bsf     STATUS,IRP      ;
      call    To_BARG         ;
      bcf     STATUS,IRP      ;
```

```
        call    FPA32           ; add air friction force

        movlw   V_m             ; retrive velocity in m/s
        call    To_BARG         ;
        call    FPM32           ; this is the power in Watts  P=FV

        movlw   P_drive         ; retrive power due to drive train
        bsf     STATUS,IRP      ;
        call    To_BARG         ;
        bcf     STATUS,IRP      ;
        call    FPA32           ; add drive train power

;*********************************************************
; Calculate Energy in Kilo Calories
            ; Kcal = ((((10.8 * Watts)/Mass) + 3.5) * Mass/12000) * Tr

        movlw   C_kcal1         ; get constant    (10.8)
        bsf     STATUS,IRP      ;
        call    To_BARG         ;
        bcf     STATUS,IRP      ;
        call    FPM32           ; times Watts

        movlw   M_r             ; get mass of rider
        bsf     STATUS,IRP      ;
        call    To_BARG         ;
        bcf     STATUS,IRP      ;
        call    FPD32           ; divide by mass of rider

        movlw   C_kcal2         ; get constant    (3.5)
        bsf     STATUS,IRP      ;
        call    To_BARG         ;
        bcf     STATUS,IRP      ;
        call    FPA32           ; add constant

        movlw   M_r             ; get mass of rider
        bsf     STATUS,IRP      ;
        call    To_BARG         ;
        bcf     STATUS,IRP      ;
        call    FPM32           ; times mass of rider

        movlw   C_kcal3         ; get constant    (1/12000)
        bsf     STATUS,IRP      ;
        call    To_BARG         ;
        bcf     STATUS,IRP      ;
        call    FPM32           ; times constant

        movlw   T_r             ; get elapsed time in seconds
        call    To_BARG         ;
        call    FPM32           ; times elapsed time

        movlw   K_cal           ; add to total
        bsf     STATUS,IRP      ;
        call    To_BARG         ;
        bcf     STATUS,IRP      ;
        call    FPA32           ;

        movlw   K_cal           ; save kilo-calories
```

```
      bsf     STATUS,IRP      ;
      call    AARG_To         ;
      bcf     STATUS,IRP      ;


;*****************************************************************
;*****************************************************************
;*****************************************************************
; Display all Data


;*********************************************************
; display speed

      movlw   V_mph           ; get FP variable
      bsf     STATUS,IRP      ;
      call    To_AARG         ;
      bcf     STATUS,IRP      ;

      movlw   0x82            ;    multiply by 10 for testing
      movwf   BEXP            ;
      movlw   0x20            ;
      movwf   BARGB0          ;
      movlw   0x00            ;
      movwf   BARGB1          ;
      movlw   0x00            ;
      movwf   BARGB2          ;
      call    FPM32           ;

;     movlw   0x3             ; set # characters
;     movwf   N_char          ;
;     movlw   0xC7            ; set cursor position
;     call    Disp            ; display character string

      call    float_ascii     ; convert to ASCII string
      movlw   0xC6            ; set location
      call    LCD_cmd         ;
      movf    ASCII-2,W       ;
      call    LCD_data        ;
      movf    ASCII-1,W       ;
      call    LCD_data        ;
      movlw   0x2E            ; add decimal
      call    LCD_data        ;
      movf    ASCII,W         ;
      call    LCD_data        ;


;*********************************************************
; display distance

      movlw   D_t             ; get distance in meters
      call    To_AARG         ;
      movlw   C_m2mi          ; get conversion factor
      bsf     STATUS,IRP      ;(conversion factor from meters to 0.01 miles)
      call    To_BARG         ;
      bcf     STATUS,IRP      ;
```

```
      call    FPM32           ; distance in 0.01 miles

      call    float_ascii     ; convert to ASCII string
      movlw   0x99            ; set location
      call    LCD_cmd         ;
      movf    ASCII-3,W       ;
      call    LCD_data        ;
      movf    ASCII-2,W       ;
      call    LCD_data        ;
      movlw   0x2E            ; add decimal
      call    LCD_data        ;
      movf    ASCII-1,W       ;
      call    LCD_data        ;
      movf    ASCII,W         ;
      call    LCD_data        ;


;*********************************************************
; display elevation gained

      movlw   E_t             ;
      call    To_AARG         ;
      movlw   0x80            ;          convert from meters to feet
      movwf   BEXP            ;
      movlw   0x51            ;
      movwf   BARGB0          ;
      movlw   0xF9            ;
      movwf   BARGB1          ;
      movlw   0x44            ;
      movwf   BARGB2          ;
      call    FPM32           ;

      movlw   0x4             ; set # characters
      movwf   N_char          ;
      movlw   0xDA            ; set cursor position
;     call    Disp            ; display character string
                              ;   (commented out for testing)


;*********************************************************
; display deg F

      movlw   T_f             ; get FP variable
      bsf     STATUS,IRP      ;
      call    To_AARG         ;
      bcf     STATUS,IRP      ;

      movlw   0x03            ; set # characters
      movwf   N_char          ;
      movlw   0x91            ; set location
      call    Disp            ; display character string


;*********************************************************
; display slope

      movlw   Grade           ; get FP variable
```

```
      call    To_AARG         ;
                              ;
      movlw   0x82            ;          multiply by 10
      movwf   BEXP            ;
      movlw   0x20            ;
      movwf   BARGB0          ;
      movlw   0x00            ;
      movwf   BARGB1          ;
      movlw   0x00            ;
      movwf   BARGB2          ;
      call    FPM32           ;

      movlw   0x82            ;          multiply by 10
      movwf   BEXP            ;        temporary test code
      movlw   0x20            ;     used for higher resolution
      movwf   BARGB0          ;
      movlw   0x00            ;
      movwf   BARGB1          ;
      movlw   0x00            ;
      movwf   BARGB2          ;
      call    FPM32           ;


      movlw   0x4             ; set # characters
      movwf   N_char          ;
      movlw   0xD0            ; set cursor position
      call    Disp            ; display character string


;*********************************************************
; display headwind

      movlw   Wind            ; get FP variable
      bsf     STATUS,IRP      ;
      call    To_AARG         ;
      bcf     STATUS,IRP      ;
      movlw   0x2             ; set # characters
      movwf   N_char          ;
      movlw   0xA6            ; set cursor position
      call    Disp            ; display character string


;*********************************************************
; display kilo calories

      movlw   K_cal           ; get FP variable
      bsf     STATUS,IRP      ;
      call    To_AARG         ;
      bcf     STATUS,IRP      ;

      movlw   0x82            ;          multiply by 10
      movwf   BEXP            ;
      movlw   0x20            ;
      movwf   BARGB0          ;
      movlw   0x00            ;
      movwf   BARGB1          ;
      movlw   0x00            ;
```

```
      movwf   BARGB2          ;
      call    FPM32           ;


      movlw   0x4             ; set # characters
      movwf   N_char          ;
      movlw   0xE4            ; set cursor position
      call    Disp            ; display character string




      goto    skp             ;


;***** the next 3 sections of code are for testing *****
;******************************************** display hot sensor
      banksel EEADR           ;2
      movf    T_hotL,W        ; get ADRESL
      banksel 0               ;0
      movwf   AARGB3
      banksel EEADR           ;2
      movf    T_hotH,W        ; get ADRESH
      banksel 0               ;0
      movwf   AARGB2
      clrf    AARGB1
      clrf    AARGB0
      call    Int_ascii

      movlw   0xE4            ; set location
      call    LCD_cmd         ;

      movf    ASCII-3,W       ;
      call    LCD_data        ;
      movf    ASCII-2,W       ;
      call    LCD_data        ;
      movf    ASCII-1,W       ;
      call    LCD_data        ;
      movf    ASCII,W         ;
      call    LCD_data        ;
;************
;******************************************** display cold sensor
      banksel EEADR           ;2
      movf    T_coldL,W       ; get ADRESL
      banksel 0               ;0
      movwf   AARGB3
      banksel EEADR           ;2
      movf    T_coldH,W       ; get ADRESH
      banksel 0               ;0
      movwf   AARGB2
      clrf    AARGB1
      clrf    AARGB0
      call    Int_ascii

      movlw   0xE4            ; set location
      banksel 0
      call    LCD_cmd         ;
```

```
      movf    ASCII-3,W       ;
      call    LCD_data        ;
      movf    ASCII-2,W       ;
      call    LCD_data        ;
      movf    ASCII-1,W       ;
      call    LCD_data        ;
      movf    ASCII,W         ;
      call    LCD_data        ;
;************
skp
;***********************************************display W_table
      banksel EEADR           ;2
      movf    W_table,W       ; get ADRESL
      banksel 0               ;0
      movwf   AARGB3
      clrf    AARGB2
      clrf    AARGB1
      clrf    AARGB0
      call    Int_ascii

      movlw   0xDA            ; set location
      banksel 0
      call    LCD_cmd         ;

      movf    ASCII-3,W       ;
      call    LCD_data        ;
      movf    ASCII-2,W       ;
      call    LCD_data        ;
      movf    ASCII-1,W       ;
      call    LCD_data        ;
      movf    ASCII,W         ;
      call    LCD_data        ;
;************


;skp




;*********************************************************************
; update time display

      banksel 0               ;0
      movlw   0x85            ; set location
      call    LCD_cmd         ;

      banksel EEADR           ;2
      movf    Ten_m,w         ; 10s minutes
      banksel 0               ;0
      call    LCD_data        ;

      banksel EEADR           ;2
      movf    One_m,w         ; 1s minutes
```

```
      banksel 0               ;0
      call    LCD_data        ;

      banksel EEADR           ;2
      movlw   0x3A            ; colon
      banksel 0               ;0
      call    LCD_data        ;

      banksel EEADR           ;2
      movf    Ten_s,w         ; 10s seconds
      banksel 0               ;0
      call    LCD_data        ;

      banksel EEADR           ;2
      movf    One_s,w         ; 1s seconds
      banksel 0               ;0
      call    LCD_data        ;



      goto    Main            ; go back and wait
                              ; for next wheel index interrupt
;*****************************************************************
;*****************************************************************
;*****************************************************************
; end of main program loop









;*****************************************************************
;*****************************************************************
;*****************************************************************
; beginning of subroutines



;*********************************************************
; Interrupt Service Routine


I_service
      btfss   PIR1, TMR2IF          ; ***check for timer2 interrupt***
      goto    Skip1                 ;
      bcf     PIR1,TMR2IF           ;      *** if timer2 interrupt: ***

      call  Ten_ms                  ; ten millisecond interrupt code


Skip1
      banksel 0                     ;0
```

```
      btfss   INTCON,INTF           ; check for wheel interrupt *****
      goto    Skip2                 ;
      bcf     INTCON,INTF           ; clear the interrupt flag bit

      movf    TMR1H,W               ;     *** if wheel interrupt: ***
      movwf   T1H                   ; get TMR1H & TMR1L
      movf    TMR1L,W               ;     "     "
      movwf   T1L                   ;     "     "
      clrf    TMR1H                 ; reset TMR1H
      clrf    TMR1L                 ;   "   TMR1L
      bsf     C_bits,0              ; set wheel pulse flag bit


Skip2
      btfss   PIR1, TMR1IF          ; check for timer1 interrupt *****
      goto    Skip3                 ;
      bcf     PIR1,TMR1IF           ;     *** if timer1 interrupt: ***

      incf    OF_count,F            ; increment overflow counter

Skip3
      pop                           ; retrieve W, STATUS, and PCLATH

      retfie                        ; return from interrupt




;*********************************************************
; Convert Floating Point to ASCII Routine   from AN670  (modified)

float_ascii
      movlw   0x20            ; is number negative?
      btfss   AARG,7          ;
      goto    Pos             ;
      movlw   0x2D            ; if yes, put "-" in front
      bcf     AARG,7          ; remove sign bit
Pos   movwf   ASCII4          ;

      call    INT3232         ;AARG  <--  INT( AARG )

Int_ascii
      movlw   ASCII
      movwf   FSR             ;pointer = address of smallest digit
      movlw   d'4'            ;load counter with the number of
      movwf   digit_count     ;significant figures in the decimal number
flo_asclp
      clrf    BARGB0          ;Make the divisor 10.
      movlw   d'10'
      movwf   BARGB1

      bsf     PCLATH,3        ; select page 1
      call    FXD3216U        ;divide (32 bit fixed) / 10 (to get remainder)
      bcf     PCLATH,3        ; select page 0
```

```
      movf    REMB1,w         ;put remainder in w register
      movwf   INDF            ;put number into appropriate digit position

      movlw   0x30
      addwf   INDF,f          ;add 30H to decimal number to convert to
ASCII

      decf    FSR,f           ;move pointer to next digit

      decfsz  digit_count,f
      goto    flo_asclp


      movlw   ASCII3          ; suppress leading zeros
      movwf   FSR

      movf    INDF,W          ; if 1000s digit a zero?
      sublw   0x30            ;
      btfss   STATUS,Z        ;
      goto    out             ;
      movlw   0x20            ; if so, put in a space character
      movwf   INDF            ;

      movf    ASCII4,W        ; justify sign character
      movwf   ASCII3          ;
      movlw   0x20            ;
      movwf   ASCII4          ;

      incf    FSR,F           ; if 100s digit a zero?
      movf    INDF,W          ;
      sublw   0x30            ;
      btfss   STATUS,Z        ;
      goto    out             ;
      movlw   0x20            ; if so, put in a space character
      movwf   INDF            ;

      movf    ASCII3,W        ; justify sign character
      movwf   ASCII2          ;
      movlw   0x20            ;
      movwf   ASCII3          ;

      incf    FSR,F           ; if 10s digit a zero?
      movf    INDF,W          ;
      sublw   0x30            ;
      btfss   STATUS,Z        ;
      goto    out             ;
      movlw   0x20            ; if so, put in a space character
      movwf   INDF            ;

      movf    ASCII2,W        ; justify sign character
      movwf   ASCII1          ;
      movlw   0x20            ;
      movwf   ASCII2          ;

out
      return
```

```
;*********************************************************
; Move FP value to AARG Routine

To_AARG
      movwf   FSR             ; call this routine with address in W reg
      movf    INDF,W          ;
      movwf   AEXP            ; move 1st byte to AEXP

      decf    FSR,F           ;
      movf    INDF,W          ;
      movwf   AARGB0          ; move 2nd byte to AARGB0

      decf    FSR,F           ;
      movf    INDF,W          ;
      movwf   AARGB1          ; move 3rd byte to AARGB1

      decf    FSR,F           ;
      movf    INDF,W          ;
      movwf   AARGB2          ; move 4th byte to AARGB2

      return


;*********************************************************
; Move FP value to BARG Routine

To_BARG
      movwf   FSR             ; call this routine with address in W reg
      movf    INDF,W          ;
      movwf   BEXP            ; move 1st byte to BEXP

      decf    FSR,F           ;
      movf    INDF,W          ;
      movwf   BARGB0          ; move 2nd byte to BARGB0

      decf    FSR,F           ;
      movf    INDF,W          ;
      movwf   BARGB1          ; move 3rd byte to BARGB1

      decf    FSR,F           ;
      movf    INDF,W          ;
      movwf   BARGB2          ; move 4th byte to BARGB2

      return



;*********************************************************
; Move FP value from AARG to destination Routine
```

```
AARG_To
      movwf   FSR             ; call this routine with "to" address in W reg
      movf    AEXP,W          ;
      movwf   INDF            ; move 1st byte from AEXP to destination

      decf    FSR,F           ;
      movf    AARGB0,W        ;
      movwf   INDF            ; move 2nd byte from AARGB0 to destination

      decf    FSR,F           ;
      movf    AARGB1,W        ;
      movwf   INDF            ; move 3rd byte from AARGB1 to destination

      decf    FSR,F           ;
      movf    AARGB2,W        ;
      movwf   INDF            ; move 4th byte from AARGB2 to destination

      return




;*****************************************************************
;*****************************************************************
;*****************************************************************
; Ten Millisecond Interrupt Routine
;
                              ;regulate hot sensor 5 deg C above cold sensor
                              ;calculate heater power
                              ;calculate temperature drop

Ten_ms
      banksel PIE1            ;1
      bsf     TRISA,3         ; turn heater off
      banksel 0               ;0


      movlw   0x40            ; delay to allow heater to cool
      movwf   Temp            ;
Loop5b
      decfsz  Temp,F          ;
      goto    Loop5b          ;


      bsf     ADCON0,GO       ; start A/D conversion
WaitAD1
      btfsc   ADCON0,GO       ; wait until done
      goto    WaitAD1


      banksel PIE1            ;1
      movf    ADRESL,W        ; get ADRESL
      banksel EEADR           ;2
      movwf   T_hotL          ;
```

```
      subwf   T_HsetL,W       ; get difference from T_HsetL
      movwf   T_HdifL         ; save low byte difference


      banksel 0               ;0
      movf    ADRESH,W        ; get ADRESH
      banksel EEADR           ;2
      movwf   T_hotH          ; save T_hotH   (only used for testing)


      banksel 0               ;0
      btfss   STATUS,C        ; if carry, add 1 to ADRESH
      incf    ADRESH,F        ;

      movf    ADRESH,W        ; get ADRESH
      banksel EEADR           ;2

      subwf   T_HsetH,W       ; get difference from T_HsetH
      movwf   T_HdifH         ; save high byte difference

      btfss   STATUS,C        ; if < T_Hset, then turn on heater
      goto    Loop8           ;

      banksel PIE1            ;1
      bcf     TRISA,3         ; turn on heater
      banksel EEADR           ;2
      incf    T_Hpower        ; increment power counter
      goto    Get_Acc         ;

Loop8 clrf    T_HdifL         ; if Thot > Tset, return 0
      clrf    T_HdifH         ;




;*****************************************************************
*
; capture acceleration sensor

Get_Acc
      banksel 0               ;0
      movlw   b'01010001'     ; connect to AN2
      movwf   ADCON0          ;

      movlw   0x04            ; delay for acquisition time
      movwf   Temp            ;
Loop5
      decfsz  Temp,F          ;
      goto    Loop5           ;

      bsf     ADCON0,GO       ; start conversion
Loop5a
      btfsc   ADCON0,GO       ; delay for conversion time
      goto    Loop5a          ;

      banksel PIE1            ;1
```

```
      movf    ADRESL,W        ; get ADRESL
      banksel EEADR           ;2
      movwf   Acc_L           ; save low byte
      banksel 0               ;0
      movf    ADRESH,W        ; get ADRESH
      banksel EEADR           ;2
      movwf   Acc_H           ; save high byte

      banksel 0               ;0
      movlw   b'01001001'     ; connect back to AN1,
;     (for next measurement of hot sensor)
      movwf   ADCON0          ;




;*****************************************************************
; calculate one second intervals

Second_timer
      banksel EEADR           ;2
      decfsz  Sec_count       ; decrement seconds counter,
                              ; if 0 then update seconds
      return                  ; if not 1 second yet, then continue
      movlw   d'100'          ; re-initialize seconds counter to 100d
      movwf   Sec_count       ;

      banksel 0               ;0
      bsf     C_bits,2        ; set 1 second flag bit




;*************************************************************
; capture temperature of cold sensor

Get_Tcold
      banksel 0               ;0
      movlw   b'01000001'     ; connect to AN0
      movwf   ADCON0          ;

      movlw   0x04            ; delay for acquisition time
      movwf   Temp            ;
Loop6
      decfsz  Temp,F          ;
      goto    Loop6           ;

      bsf     ADCON0,GO       ; start conversion
Loop6a
      btfsc   ADCON0,GO       ; delay for conversion time
      goto    Loop6a          ;

      banksel PIE1            ;1
```

```
      movf    ADRESL,W        ; get ADRESL
      banksel EEADR           ;2
      movwf   T_coldL         ; save low byte
      banksel 0               ;0
      movf    ADRESH,W        ; get ADRESH
      banksel EEADR           ;2
      movwf   T_coldH         ; save high byte

      banksel 0               ;0
      movlw   b'01001001'     ; connect back to AN1,
;     (for next measurement of hot sensor)
      movwf   ADCON0          ;




;*************************************************************
; calculate 5 deg elevated set point for hot sensor

      banksel PIE1            ;1
      movlw   d'162'          ; add 102 (5 degrees C)
                              ; to cold sensor temperature
      addwf   ADRESL,W        ; plus correction for sensor error 60
      banksel EEADR           ;2
      movwf   T_HsetL         ;

      banksel 0               ;0
      btfsc   STATUS,C        ; if carry, add 1 to T_HsetH
      incf    ADRESH,F
      movf    ADRESH,W        ;
      banksel EEADR           ;2
      movwf   T_HsetH         ;




;*************************************************************
; calculate pointer to wind speed table

      banksel EEADR           ;2
      movlw   d'40'           ; shift to upper half of range
      subwf   T_Hpower,W      ;

      btfss   STATUS,C        ;
      clrw                    ; set low end stop of 0
      movwf   W_tablea        ; save in W_tablea



      movlw   d'60'           ; test, is it at power max?
      subwf   W_tablea,W      ;   "         "         "
      btfss   STATUS,Z        ;   "         "         "
```

```
      goto    Continue        ; continue

                              ;      if at power max,
                              ;      then add extension to table
                              ;      for temperature drop of hot sensor
      bcf     STATUS,C        ; make sure carry bit is 0
      rrf     T_HdifL,W       ; divide by 2,
                              ;     to adjust range for table extension
      addwf   W_tablea,F      ; add extension to table pointer

Continue
      movlw   d'91'           ; set high end stop of d'90'
      subwf   W_tablea,W      ;
      movf    W_tablea,W      ;
      btfsc   STATUS,C        ;
      movlw   d'90'           ;
      movwf   W_tablea        ;

      clrf    T_Hpower        ; reset power counter



;************************************************************
; calculate average of pointer

      decf    Count,F         ; average 8 readings
      btfsc   STATUS,Z        ;
      goto    Avg1            ;

      movf    W_tablea,W      ; add last 8 values
      addwf   W_tableL,F      ;
      btfsc   STATUS,C        ;
      incf    W_tableH,F      ;
      goto    Avg2            ;

Avg1  bcf     STATUS,C        ; divide by 8
      rrf     W_tableH,F      ;
      rrf     W_tableL,F      ;
      bcf     STATUS,C        ;
      rrf     W_tableH,F      ;
      rrf     W_tableL,F      ;
      bcf     STATUS,C        ;
      rrf     W_tableH,F      ;
      rrf     W_tableL,F      ;

      movf    W_tableL,W      ; save averaged table pointer
      movwf   W_table         ;

      movlw   d'9'            ; reset
      movwf   Count           ;
      clrf    W_tableH        ;
      clrf    W_tableL        ;

Avg2
```

```
;*************************************************************
; update time counters,   return to the rest of the interrupt when done

      banksel EEADR           ;2
      incf    One_s           ; update 1s seconds
      btfss   One_s,3         ;
      return                  ;
      btfss   One_s,1         ;
      return                  ;
      movlw   0x30            ;
      movwf   One_s           ;

      incf    Ten_s           ; update 10s seconds
      btfss   Ten_s,2         ;
      return                  ;
      btfss   Ten_s,1         ;
      return                  ;
      movlw   0x30            ;
      movwf   Ten_s           ;

      incf    One_m           ; update 1s minutes
      btfss   One_m,3         ;
      return                  ;
      btfss   One_m,1         ;
      return                  ;
      movlw   0x30            ;
      movwf   One_m           ;

      bsf     Ten_m,4         ; first time through change space to #0
      incf    Ten_m           ; update 10s minutes
      btfss   Ten_m,2         ;
      return                  ;
      btfss   Ten_m,1         ;
      return                  ;
      movlw   0x30            ;
      movwf   Ten_m           ;

      bsf     One_h,4         ; first time through change space to #0
      incf    One_h           ; update 1s hours
      btfss   One_h,3         ;
      return                  ;
      btfss   One_h,1         ;
      return                  ;
      movlw   0x30            ;
      movwf   One_h           ;

      bsf     Ten_h,4         ; first time through change space to #0
      incf    Ten_h           ; update 10s hours
      btfss   Ten_h,2         ;
      return                  ;
      btfss   Ten_h,1         ;
      return                  ;
      movlw   0x20            ;
      movwf   Ten_h           ;


      return                  ; go back to the rest of the interrupt handler
```

```
;*********************************************************************
;*********************************************************************
;*********************************************************************




;***********************************************************
; Routine to create and display an ASCII character string from a FP
variable
;
                              ; call with:      number of characters in
N_char
                              ;           cursor location in W
                              ;           FP variable in AARG
Disp
      call    LCD_cmd         ; set cursor position
      call    float_ascii     ; convert to ASCII string

      movf    N_char,W        ; display 5 characters
      sublw   0x04            ;
      btfsc   STATUS,C        ;
      goto    Disp1           ;
      movf    ASCII-4,W       ;
      call    LCD_data        ;

Disp1
      movf    N_char,W        ; display 4 characters
      sublw   0x03            ;
      btfsc   STATUS,C        ;
      goto    Disp2           ;
      movf    ASCII-3,W       ;
      call    LCD_data        ;

Disp2
      movf    N_char,W        ; display 3 characters
      sublw   0x02            ;
      btfsc   STATUS,C        ;
      goto    Disp3           ;
      movf    ASCII-2,W       ;
      call    LCD_data        ;

Disp3
      movf    N_char,W        ; display 2 characters
      sublw   0x01            ;
      btfsc   STATUS,C        ;
      goto    Disp4           ;
      movf    ASCII-1,W       ;
      call    LCD_data        ;

Disp4
      movf    ASCII,W         ; display 1 character
      call    LCD_data        ;

      return
```

```
;*************************************************************
; Routine to load labels in LCD display
;
LCD_labels
      movlw 0x80              ; set Time
      call  LCD_cmd
      movlw 0x54
      call  LCD_data
      movlw 0x69
      call  LCD_data
      movlw 0x6D
      call  LCD_data
      movlw 0x65
      call  LCD_data


      movlw 0xC0              ; set Speed
      call  LCD_cmd
      movlw 0x53
      call  LCD_data
      movlw 0x70
      call  LCD_data
      movlw 0x65
      call  LCD_data
      movlw 0x65
      call  LCD_data
      movlw 0x64
      call  LCD_data

      movlw 0x94              ; set Dist
      call  LCD_cmd
      movlw 0x44
      call  LCD_data
      movlw 0x69
      call  LCD_data
      movlw 0x73
      call  LCD_data
      movlw 0x74
      call  LCD_data


      movlw 0xD4              ; set Elev
      call  LCD_cmd
      movlw 0x45
      call  LCD_data
      movlw 0x6C
      call  LCD_data
      movlw 0x65
      call  LCD_data
      movlw 0x76
      call  LCD_data
```

```
      movlw 0x8B              ; set Temp
      call  LCD_cmd
      movlw 0x54
      call  LCD_data
      movlw 0x65
      call  LCD_data
      movlw 0x6D
      call  LCD_data
      movlw 0x70
      call  LCD_data

      movlw 0xCB              ; set Grade
      call  LCD_cmd
      movlw 0x47
      call  LCD_data
      movlw 0x72
      call  LCD_data
      movlw 0x61
      call  LCD_data
      movlw 0x64
      call  LCD_data
      movlw 0x65
      call  LCD_data

      movlw 0x9F              ; set Wind
      call  LCD_cmd
      movlw 0x57
      call  LCD_data
      movlw 0x69
      call  LCD_data
      movlw 0x6E
      call  LCD_data
      movlw 0x64
      call  LCD_data

      movlw 0xDF              ; set Cal
      call  LCD_cmd
      movlw 0x43
      call  LCD_data
      movlw 0x61
      call  LCD_data
      movlw 0x6C
      call  LCD_data

      return




;*********************************************************
; Routine to initialize the LCD display interface
;
LCD_init                      ; LCD initialization
      banksel 0               ;0
```

```
      clrf    PORTC           ; make RS=0, R/W=0, E=0

      banksel TRISD           ;1
      clrf    TRISD           ; make PORTD an output
      bcf     TRISC,0         ; make port D, port C <0:2> outputs
      bcf     TRISC,1         ;
      bcf     TRISC,2         ;
      banksel 0               ;0

      movlw   0x38            ; set LCD function
      call    LCD_cmd

      movlw   0x0C            ; set LCD display
      call    LCD_cmd

      return




;*********************************************************
; Routine to send commands to LCD controller
;
LCD_cmd
      call    LCD_busy        ; wait until ready
      movwf   PORTD           ; output command
      bsf     PORTC,2         ; cycle E
      bcf     PORTC,2
      return




;*********************************************************
; Routine to send ASCII characters to LCD controller
;
LCD_data
      call    LCD_busy        ; wait until ready
      bsf     PORTC,0         ; RS = 1, set for data write
      movwf   PORTD           ; output data
      bsf     PORTC,2         ; cycle E
      bcf     PORTC,2
      return




;*********************************************************
; Routine to delay until LCD controller is not busy
;
LCD_busy
      banksel TRISD           ;1
```

```
      comf    TRISD           ; set PORTD for input
      banksel 0               ;0

      bcf     PORTC,0         ; RS = 0
      bsf     PORTC,1         ; R/W = 1
Busy1
      bsf     PORTC,2         ; E = 1
      btfss   PORTD,7         ; is BF set?
      goto    Done1           ; if not, then done

      bcf     PORTC,2         ; E = 0
      goto    Busy1           ; still busy
Done1
      bcf     PORTC,2         ; E = 0
      bcf     PORTC,1         ; R/W = 0

      banksel TRISD           ;1
      clrf    TRISD           ; set PORTD for output
      banksel 0               ;0

      return




;*****************************************************************
;*****************************************************************
;*****************************************************************
; Attach Math Routines
;

      include "fp32.a16"      ;32 bit float routines from AN575

      org     0x800
      include "fxd26.a16"     ;fixed point 32/16 divide from AN617

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


;*********************************************************
; Routine to initialize floating point variables
;
Init_FP                       ;
************************************************
      banksel PIE1            ;1            Initialize bank 1 registers
                              ;
************************************************

      movlw   0x82            ; A_g = 9.81 m/S
      movwf   A_g             ;     gravity constant
      movlw   0x1C            ;
      movwf   A_g-1           ;
```

```
movlw   0xF5            ;
movwf   A_g-2           ;
movlw   0xC3            ;
movwf   A_g-3           ;

movlw   0x85            ; C_100 = 100
movwf   C_100           ;     used for scaling
movlw   0x48            ;
movwf   C_100-1         ;
movlw   0x00            ;
movwf   C_100-2         ;
movlw   0x00            ;
movwf   C_100-3         ;

movlw   0x76            ; A_scale = 0.00323275
movwf   A_scale         ;     scale factor for accelerometer
movlw   0x53            ;
movwf   A_scale-1       ;
movlw   0xDC            ;
movwf   A_scale-2       ;
movlw   0x8C            ;
movwf   A_scale-3       ;

movlw   0x87            ; A_zero = 284
movwf   A_zero          ;     zero offset for accelerometer
movlw   0x0E            ;
movwf   A_zero-1        ;
movlw   0x00            ;
movwf   A_zero-2        ;
movlw   0x00            ;
movwf   A_zero-3        ;

movlw   0x80            ; D_w = 2.124075 m
movwf   D_w             ;     wheel circumference
movlw   0x07            ;
movwf   D_w-1           ;
movlw   0xF0            ;
movwf   D_w-2           ;
movlw   0xD8            ;
movwf   D_w-3           ;

clrf    Grade           ; Grade = 0
clrf    Grade-1         ;
clrf    Grade-2         ;
clrf    Grade-3         ;

clrf    D_t             ; D_t = 0
clrf    D_t-1           ;
clrf    D_t-2           ;
clrf    D_t-3           ;

clrf    V_o             ; V_o = 0
clrf    V_o-1           ;
clrf    V_o-2           ;
clrf    V_o-3           ;

clrf    E_t             ; E_t = 0
```

```
      clrf    E_t-1           ;
      clrf    E_t-2           ;
      clrf    E_t-3           ;

      movlw   0x82            ; Eight = 8
      movwf   Eight           ;     used for scaling
      clrf    Eight-1         ;
      clrf    Eight-2         ;
      clrf    Eight-3         ;

      movlw   0x6E            ; T1_scale = 1x10^-6  *   8.0
      movwf   T1_scale        ;     to convert from microseconds to seconds
      movlw   0x06            ;       and correct for prescaler 1:8
      movwf   T1_scale-1      ;
      movlw   0x37            ;
      movwf   T1_scale-2      ;
      movlw   0xBD            ;
      movwf   T1_scale-3      ;

      movlw   0x76            ; C_00223 = 0.002236936
      movwf   C_00223         ;     constant for drive friction
      movlw   0x12            ;
      movwf   C_00223-1       ;
      movlw   0x99            ;
      movwf   C_00223-2       ;
      movlw   0x8F            ;
      movwf   C_00223-3       ;

      movlw   0x86            ; C_140 = 140
      movwf   C_140           ;     constant for drive friction
      movlw   0x0C            ;
      movwf   C_140-1         ;
      movlw   0x00            ;
      movwf   C_140-2         ;
      movlw   0x00            ;
      movwf   C_140-3         ;


                              ;
**************************************************
      banksel EEADR           ;2            Initialize bank 2 registers
                              ;
**************************************************

      movlw   0x20            ; Clock display registers
      movwf   Ten_h           ;
      movwf   One_h           ;
      movwf   Ten_m           ;
      movlw   0x30            ;
      movwf   One_m           ;
      movwf   Ten_s           ;
      movwf   One_s           ;
      movlw   0x64            ; Initialize seconds counter to 100d
      movwf   Sec_count       ;

      movlw   0x7B            ; Scale_TF  =  0.0879765
      movwf   Scale_TF        ;     scale from deg C in binary to deg F
```

```
movlw   0x34            ;
movwf   Scale_TF-1      ;
movlw   0x2D            ;
movwf   Scale_TF-2      ;
movlw   0x06            ;
movwf   Scale_TF-3      ;

movlw   0x84            ; Off_TF = 32
movwf   Off_TF          ;     Offset of 32 degrees C to F
movlw   0x30            ;     + 12 deg correction
movwf   Off_TF-1        ;
clrf    Off_TF-2        ;
clrf    Off_TF-3        ;

movlw   0x82            ; C_kcal1  =  10.8
movwf   C_kcal1         ;     kcal conversion formula constant
movlw   0x2C            ;
movwf   C_kcal1-1       ;
movlw   0xCC            ;
movwf   C_kcal1-2       ;
movlw   0xCD            ;
movwf   C_kcal1-3       ;

movlw   0x80            ; C_kcal2  =  3.5
movwf   C_kcal2         ;     kcal conversion formula constant
movlw   0x60            ;
movwf   C_kcal2-1
clrf    C_kcal2-2
clrf    C_kcal2-3

movlw   0x71            ; C_kcal3  =  1/12,000
movwf   C_kcal3         ;     kcal conversion formula constant
movlw   0x2E            ;
movwf   C_kcal3-1       ;
movlw   0xC3            ;
movwf   C_kcal3-2       ;
movlw   0x3E            ;
movwf   C_kcal3-3       ;

movlw   0x82            ; M_v  =  13.6 Kg  =  (30 pounds)
movwf   M_v             ;     mass of vehicle
movlw   0x59            ;
movwf   M_v-1           ;
movlw   0x99            ;
movwf   M_v-2           ;
movlw   0x9A            ;
movwf   M_v-3           ;

movlw   0x85            ; M_r  =  70 Kg  =  (155 pounds)
movwf   M_r             ;     mass of rider
movlw   0x0C            ;
movwf   M_r-1           ;
movlw   0x00            ;
movwf   M_r-2           ;
movlw   0x00            ;
movwf   M_r-3           ;
```

```
movlw   0x7A            ; C_m2mi  =  0.0006213711
movwf   C_m2mi          ;     meter to miles
movlw   0x7E            ;     conversion constant
movwf   C_m2mi-1        ;     changed to 0.06213711
movlw   0x83            ;     0.01 mile
movwf   C_m2mi-2        ;
movlw   0x7B            ;
movwf   C_m2mi-3        ;

movlw   0x80            ; C_ms2mph  =  2.236936
movwf   C_ms2mph        ;     meters per second to miles per hour
movlw   0x0F            ;     conversion constant
movwf   C_ms2mph-1      ;
movlw   0x29            ;
movwf   C_ms2mph-2      ;
movlw   0xF6            ;
movwf   C_ms2mph-3      ;

movlw   0x7B            ; F_ascale  =  0.071506
movwf   F_ascale        ;
movlw   0x12            ;
movwf   F_ascale-1      ;
movlw   0x71            ;
movwf   F_ascale-2      ;
movlw   0xBD            ;
movwf   F_ascale-3      ;

clrf    W_tableH        ; averaging variables
clrf    W_tableL        ;
movlw   d'9'            ;
movwf   Count           ;



banksel 0               ;0
bcf     PCLATH,3        ; select page 0

bsf     STATUS,IRP      ; select bank 2,3 for indirect addressing
movlw   M_v             ; get mass of vehicle
call    To_AARG         ;
movlw   M_r             ; get mass of rider
call    To_BARG         ;
bcf     STATUS,IRP      ; select bank 0,1 for indirect addressing

call    FPA32           ; calculate mass of rider and vehicle

bsf     STATUS,IRP      ; select bank 2,3 for indirect addressing
movlw   M_vr            ;
call    AARG_To         ; save M_vr
bcf     STATUS,IRP      ; select bank 0,1 for indirect addressing

bsf     PCLATH,3        ; select page 1

return
```

```
;*********************************************************************
;*********************************************************************
;*********************************************************************

      org   0x2100            ;
                              ; ************************************
                              ;            Initialize table
                              ; ************************************

      de    d'1'        ; airspeed look-up table
      de    d'2'        ;
      de    d'3'        ;
      de    d'4'        ;
      de    d'5'        ;
      de    d'6'        ;
      de    d'7'        ;
      de    d'8'        ;
      de    d'9'        ;
      de    d'10'       ;
      de    d'11'       ;
      de    d'12'       ;
      de    d'13'       ;
      de    d'14'       ;
      de    d'15'       ;
      de    d'16'       ;
      de    d'17'       ;
      de    d'18'       ;
      de    d'19'       ;
      de    d'20'       ;
      de    d'21'       ;
      de    d'22'       ;
      de    d'23'       ;
      de    d'24'       ;
      de    d'25'       ;
      de    d'26'       ;
      de    d'27'       ;
      de    d'28'       ;
      de    d'29'       ;
      de    d'30'       ;
      de    d'31'       ;
      de    d'32'       ;
      de    d'33'       ;
      de    d'34'       ;
      de    d'35'       ;
      de    d'36'       ;
      de    d'37'       ;
      de    d'38'       ;
      de    d'40'       ;
      de    d'43'       ;
      de    d'46'       ;
      de    d'50'       ;
      de    d'53'       ;
      de    d'56'       ;
      de    d'59'       ;
      de    d'62'       ;
      de    d'66'       ;
      de    d'69'       ;
```

```
      de    d'72'       ;
      de    d'76'       ;
      de    d'80'       ;
      de    d'85'       ;
      de    d'90'       ;
      de    d'94'       ;
      de    d'99'       ;
      de    d'103'      ;
      de    d'108'      ;
      de    d'112'      ;
      de    d'116'      ;
      de    d'121'      ;
      de    d'125'      ;
      de    d'130'      ;
      de    d'134'      ;
      de    d'139'      ;
      de    d'143'      ;
      de    d'148'      ;
      de    d'152'      ;
      de    d'68'       ;  not used at this time
      de    d'69'       ;
      de    d'70'       ;
      de    d'71'       ;
      de    d'72'       ;
      de    d'73'       ;
      de    d'74'       ;
      de    d'75'       ;
      de    d'76'       ;
      de    d'77'       ;
      de    d'78'       ;
      de    d'79'       ;
      de    d'80'       ;
      de    d'81'       ;
      de    d'82'       ;
      de    d'83'       ;
      de    d'84'       ;
      de    d'85'       ;
      de    d'86'       ;
      de    d'87'       ;
      de    d'88'       ;
      de    d'89'       ;
      de    d'90'       ;
      de    d'91'       ;

;*********************************************************************
;*********************************************************************
;*********************************************************************
      End
```