July 2006

Flight Data System

Russell A. Pead
Worcester Polytechnic Institute

Tri Lai
Worcester Polytechnic Institute

Project Number: FJL-AVON

Flight Data System

A Major Qualifying Project Report: submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
by

Tri Lai
trilai@wpi.edu   _____

Russell Pead
rustyp@wpi.edu   _____

Date: July 26, 2006

Fred Looft
fjlooft@ece.wpi.edu ———————————————

WPI Worcester Polytechnic Institute
100 Institute Road Worcester, MA 01609

Abstract

The goal of this capstone design project was to design a fully portable sailplane Lift/Drag (L/D) calculator. The system utilizes NMEA format GPS data strings for software data analysis executed by software. Calculated effective L/D results are stored on a removable data storage media (e.g. CompactFlash card) for later data analysis.

Table of Contents

Table of Figures

1   Introduction

1.1 Introduction

One of the least known sports is high performance gliding, commonly referred to as soaring. Sailplanes make use of thermals, rising columns of air, and other forms of lift to achieve high altitudes. A pilot can fly for hours depending on their skill level, granted conditions are ideal. Some of these conditions include weather, the number of thermals in the area, and the strength of the area. These will be explained in more detail later.

In order to successfully fly long distances and increase flight time, two critical characteristics of a sailplane should be optimized. The two most critical performance characteristics of a sailplane are the Lift/Drag (L/D) ratio and Minimum Sink Rate. Both performance characteristics rely heavily on the physical characteristics of the plane such as weight, wing shape, and the actual aerodynamic design of the plane. Because the pilot has no control over physical aspects of the plane, the pilot must control different flight variables, such as velocity and angle of attack.

Since the actual L/D ratio of a plane relies on the physical characteristics of the plane, it is not possible to design a portable L/D calculator without having to program the system with such information. However, using GPS data, it is possible to calculate an effective L/D, also known as best glide ratio.

1.2 Problem Statement

The purpose of the Flight Data System project was to design, implement, and test a fully independent portable instrument capable of calculating the effective L/D ratio of a sailplane using GPS data and storing data onto removable media for analysis.

1.3 Summary

This chapter introduced a problem, which sailplane pilots frequently find themselves confronted with. In order to help solve this problem, we have developed a problem statement, which in turn will lead to the development of a portable Flight Data System that can calculate the effective L/D of any given sailplane. Although there are currently instruments on the market capable of calculating the effective L/D ratio, we hope to develop a system that has the capability to store data onto removable media, is lightweight, and portable.

The proceeding is a list of chapters following this section and a quick overview of each chapter's contents:

- Background – Contains an explanation regarding some basic knowledge of soaring. This chapter also provides an explanation of how the Global Positioning System works.
- Problem Statement – Defines the purpose of this project and the requirements of the Flight Data System.
- System Design – Detailed explanation of the individual modules required to implement the Flight Data System. It also contains the trade studies conducted to validate the component selections used in the individual modules.
- Results – Contains an explanation of how the hardware and software were implemented once each module was tested individually. This chapter also contains the data obtained from our tests and also an analysis of the data.
- Summary and Conclusions – Compares our final results to the objectives stated in the Problem Statement chapter. It also lists any future works that should be considered and recognizes parts of the project in which we would have done differently.

2   Background

In order to understand the approximation method used in this project, the fundamental theory behind how the Lift/Drag ratio is calculated must be understood. In order to understand why the L/D ratio is important during flight, a basic knowledge of soaring must be acquired. A working knowledge of the Global Positioning System (GPS) is also necessary to understand the approximation method implemented in our design.

2.1 Soaring

Throughout time, mankind has always dreamed of the ability to fly in the sky amongst the birds. In Greek mythology, Daedalus and his son, Icarus, were imprisoned by King Minos. In order to escape, Daedalus fabricated wings made of wax and feathers so that they may fly off the island. Prior to the escape, the father had warned Icarus not to fly too close to the sun or the wax would melt the wings. Although forewarned, Icarus became ecstatic with the ability to fly and kept flying higher closer to the sun. The wings eventually melted and Icarus fell to his death.

During the fifteenth century, Leonard da Vinci made sketches of machines capable of flying, one of which almost resembles the modern day helicopters. Figure 1 is a copy of da Vinci's sketch of an aerial screw, "classified as the helicopters ancestor" (AIAA 2006).
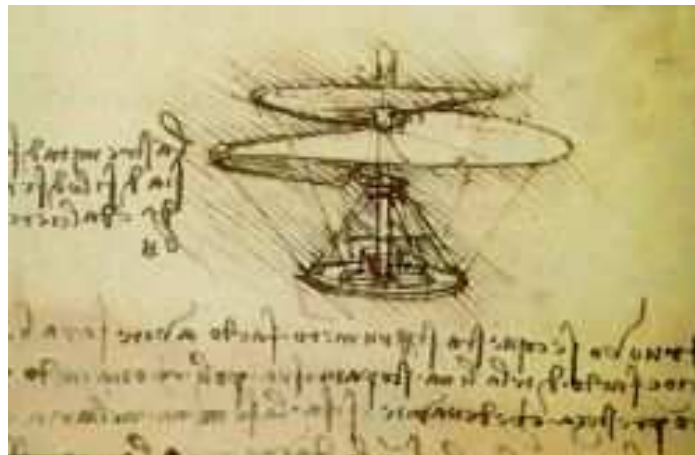
Figure 1 - Leonardo da Vinci's Aerial Screw[1]

During the late 1700's and early 1800's, a man by the name George Cayley designed a series of gliders which were controlled by body movements. In the late 1800's, Otto Lilienthal, a German engineer, designed a glider and was the first person able to fly long distances (UEET NASA). Most known in history for flight are the Wright Brothers. They were the first to make powered flight possible, which occurred in 1903.

---

[1] http://www.aiaa.org/content.cfm?pageid=425

During the 1920's soaring became a sport of its own caused in part, by the World War I treaty of Versailles because it banned Germany from any type of powered flight (USST 2004). Soaring flight is defined as the ability to "maintain or gain altitude rather than slowly gliding downward" (FAA 2003). In the early 1920's, pilots found lift in the updraft caused by wind deflected off a hillside. Soon after in the late 1920's, Robert Kronfeld discovered the ability to use rising warm air, known as thermals, as a form of lift for his glider. Then in the early 1933, Wolf Hirth discovered wave lift or air blowing over mountain ranges forming waves (USST 2004), which lead to the first high altitude flights (FAA 2003).

2.1.1 Thermal Soaring

From the different types of soaring, thermal soaring is the most common. The thermalling process consists of four main steps: locating a thermal, entering the thermal, centering, and finally exiting. A thermal is defined by the Merriam-Webster Dictionary as "a rising body of warm air" and according to the FAA it is "a buoyant plume or bubble of rising air". Figure 2 is an illustration of thermals in the plume form (left) and the bubble form (right). Figure 3 illustrates a general path of glider while thermal soaring, in which the upward pointing arrows depict thermals and the downward pointing arrows depict cooler sinking air.
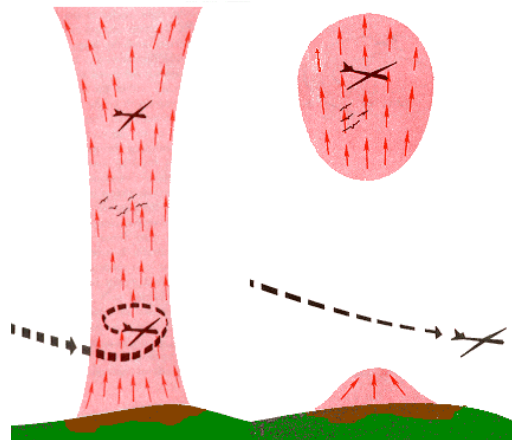
Figure 2 - Plume Thermal vs. Bubble Thermal[2]
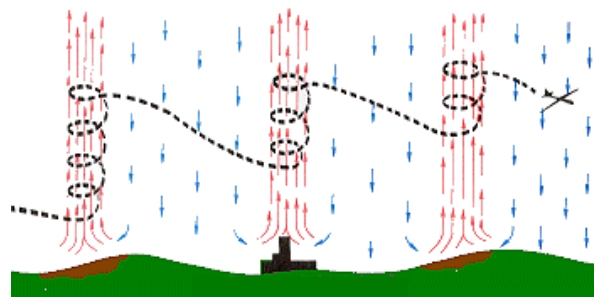
Figure 3 - General Illustration of Thermal Soaring[3]

---

[2] https://ntc.cap.af.mil/ops/DOT/school/NCPSC/GliderNCPSC/CAPF_5_glider/soaringtechniques.htm

[3] https://ntc.cap.af.mil/ops/DOT/school/NCPSC/GliderNCPSC/CAPF_5_glider/soaringtechniques.htm

Since air is invisible, locating thermals can be difficult. Cumulus clouds, also known as Cu (pronounced like the letter 'q'), are common indicators of thermals because when there is enough moisture in the air and the air is warm enough to carry it high enough, the moisture begins to condense and form clouds. Figure 4 illustrates the formation of a cumulus cloud. Deeper clouds, indicated by darker bases, typically have stronger thermals. Clouds that have a fuzzy or stringy appearance typically mean that the clouds are at the end of their lifetimes and there is little lift or possibly even sink.
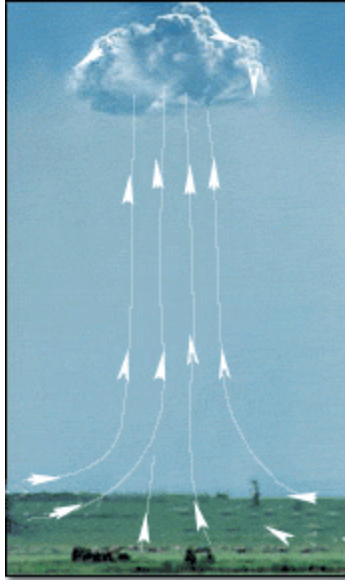
Figure 4 - Formation of a Cumulus Cloud[4]

A cloudless sky does not necessarily mean that there are not any thermals around. Blue thermals, or dry thermals, will form when the air is cool enough and the surface temperatures warm up sufficiently and are just as strong as typical thermals marked by clouds (FAA 2003). The lack of moisture in the air is the reason why cumulus clouds are not formed and why they are called dry thermals. These thermals are typically found by accident but to locate dry thermals, there are several indicators. Some common indicators are:

- Another glider circling
- Circling birds
- Haze domes created by aerosols transported and deposited by thermals

Typically the first indicator of a nearby thermal is increased sink, in which flight speed should be increased in order to penetrate the sink. After the sink, a positive G-Force is felt and is this force increases, speed should be reduced to between the best L/D speed and minimum sink speed. Then, at the right moment, begin turning into the thermal and if all goes well, "the glider will roll into a coordinated turn, at just the right bank angle, at just the right speed, and be perfectly centered" (FAA 2003).

---

[4] http://www.flyaboveall.com/mountainpilot/thermalclinic.htm

Centering is important because gliders have a natural tendency to drift away from thermals. In the center of a thermal is where the stronger lift occurs, and as a glider flies into a thermal not centered, the stronger lift will cause the glider to bank in the opposing direction. Figure 5 shows a glider entering a thermal, not centered, to its left. The stronger lift in the center causes the glider to bank right into another thermal, but again is not centered and the new thermal causes the glider to bank left away from either thermal. In other words, the thermal rejects the glider if it is not centered. Inside the thermal, use the smallest possible bank angle at minimum sink speed. The reason for this is the higher the bank angle, the greater the sink and flying below minimum sink speed increases the chance of a stall.
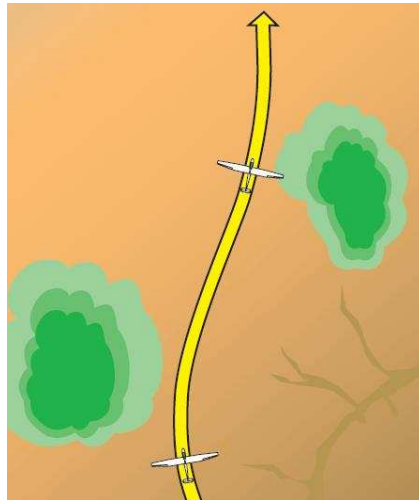
Figure 5 - Thermalling: Result of Not Centering[5]

Nearing completion of thermalling, the pilots must continually check the sky so that they may see if there is any nearby air traffic and also determine where to go for the next thermal. This is important to avoid any collisions with other gliders and because it may result in an unexpected landing. It is also important to increase speed prior to exiting to penetrate back through the strong sink found on the edge of a thermal.

2.1.2 Ridge and Slope Soaring

When wind is deflected over hillsides or ridges, an updraft is created which can be used as a form of lift. This is known as ridge soaring, or slope soaring. In "*Gliding – A Handbook on Soaring*", Derek Piggott states the best lift in slope soaring is "found over or in front of the steepest slope". Thermals tend to be found nearby so a combination of thermalling and slope soaring can be utilized. Figure 6 is an example of using a combination of slope soaring and thermalling. If the thermal is at the end of its cycle, the slope can be used to continue soaring. It is important to fly above minimum sink speed while slope soaring because minimum sink speed is close to stall speed and also it may be harder to control the glider around the terrain. (FAA 2003)

---

[5] Figure 10-7 from FAA's "Glider Flying Handbook"
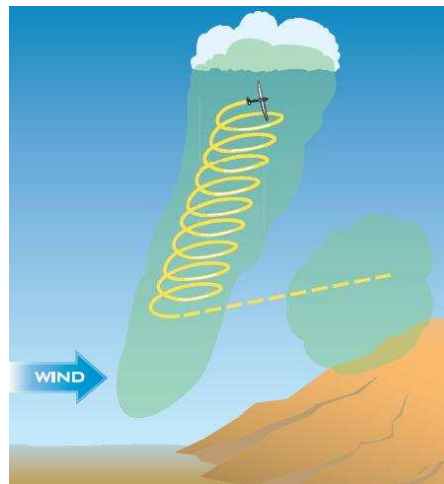
Figure 6 - Combination of Slope Soaring and Thermalling[6]

2.1.3 Wave Soaring

Similar to slope soaring, wave soaring utilizes air waves coming from mountains, known as lee waves. The big difference between slope soaring and wave soaring is that the updraft from a hill occurs on the upwind side, where as the lee waves are formed on the downwind side. As wind flows along the upwind side of a mountain, air parcels are displaced upward. The parcels have a natural tendency to stay at an equilibrium level, so when the wind stops pushing the parcel upwards, the parcel begins to fall to try to regain the equilibrium level. Upon reaching its equilibrium level, it had accelerated to a speed too fast for it to stop and overshoots the equilibrium. Closer to the ground the parcel warms up and begins to float upwards. This process repeats, diminishing in amplitude with each cycle, forming the lee waves. Figure 7 illustrates a lee wave system. Under the crest, there is a turbulent area, called rotors, which can be mild to severe.
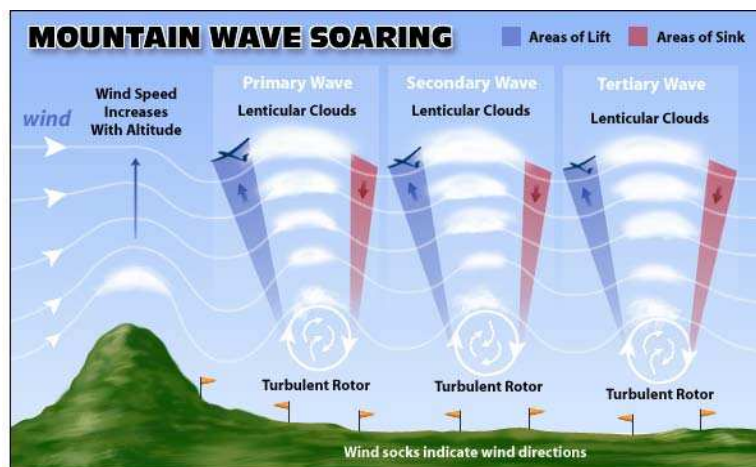
Figure 7 - Lee Wave System[7]

---

[6] Figure 10-18 from FAA's "Glider Flying Handbook"

[7] http://www.ssa.org/sport/whatissoaring3.asp

There are two main ways to enter the wave system. The first way is to soar into it, which provides three options:

- Thermalling
- Being towed to the upside of the rotor and climbing up to the wave
- Transitioning into the wave from slope soaring

The other option is to be towed directly through the rotor and releasing into the wave. It is possible to ask the tow pilot to fly around the rotor, but this usually takes more time and is less ideal for the tow pilot.

Once inside the wave, the best lift of the wave will be found on the upwind side of the rotor and the most efficient speed to fly will be close to the gliders minimum sink speed. If the winds are stronger on a particular day, it will be necessary to fly faster than the minimum sink speed so that the glider does not drift backwards into a sink region. Once inside the sink region, it will be hard to penetrate the wind to get back to the area of best lift. Wave soaring has been known for providing the longest and highest flights because the waves can go well above the mountain top.

If the pilot desires to descend, the pilot may utilize the sink region downwind. The sink region itself "can easily be twice as strong as lift encountered upwind". It is very likely and almost unavoidable that a decent into the turbulent rotor zone will occur. Going through the rotor is always unpleasant and in the worse case scenario be extremely dangerous if done too fast. (FAA 2003)

2.2 Speed-to-Fly

According to the FAA's "Glider Flying Handbook", Speed-to-Fly is explained as follows:

"Speed-to-fly refers to the optimum airspeed for proceeding from one source of lift to another. Speed-to-fly depends on the following:

1. *The rate-of-climb the pilot expects to achieve in the next thermal or updraft*
2. *The rate of ascent or descent of the air mass through which the glider is flying*
3. *The glider's inherent sink rate at all airspeeds between minimum sink airspeed and never exceed airspeed*
4. *Headwind or tailwind*

The object of speed-to-fly is to minimize the time and/or altitude required to fly from the current position to the next thermal. Speed-to-fly information is presented to the pilot in one or more of the following ways:

- *By placing a speed-to-fly ring (MacCready ring) around the variometer dial*
- *By using a table or chart*
- *By using an electronic flight computer that displays the current optimum speed-to-fly*

The pilot determines the speed-to-fly during initial planning and than constantly updates this information in flight. The pilot must be aware of changes in the flying conditions in order to be successful in conducting cross-country flights or during competitions." (FAA 2003)

2.2.1 Lift

On any aircraft, there are three main forces acting upon the system. These three forces are lift, drag, and weight. Figure 8 illustrates how these forces acted on a glider. If the aircraft is powered, then there is an additional force called thrust, which will not be discussed for the purposes of this paper. Lift is the most important of the forces and is required for flight. It is a force produced by "the dynamic effects of the airstream acting on the wing, opposing the downward force of weight" (FAA 2003).
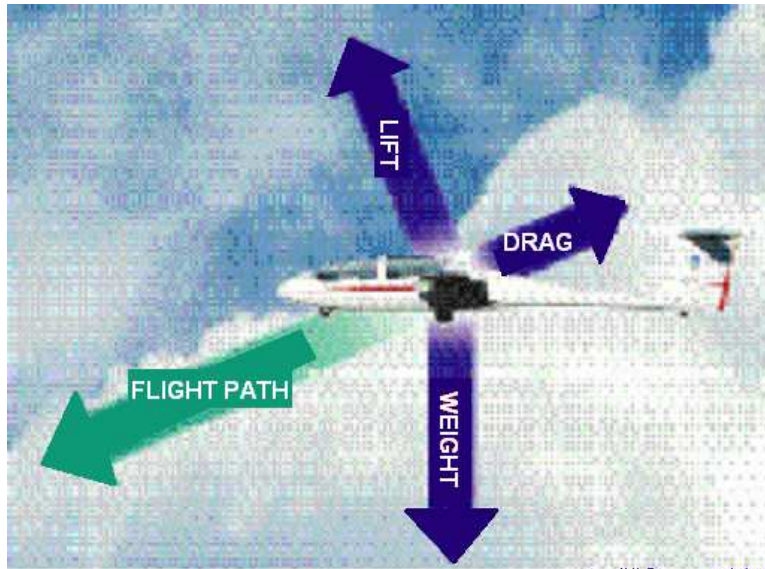
Figure 8 - Forces on a Glider[8]

Lift is defined as $L = C_L V^2 \frac{\rho}{2} S$ where L denotes lift, C_L is the coefficient of lift, V is velocity, ρ is air density, and S is the wing's surface area. The coefficient of lift itself is the ratio of lift pressure to dynamic pressure and area. Looking at the equation, it is seen that if any of these factors were to increase, so would lift.

As a wing cuts through the air, some of the air travels above the wing and some travels beneath the wing. The air flowing over the top of the wing accelerates, resulting in a decrease in pressure which is explained using Bernoulli's principle. The air flowing beneath the wing causes a build up of pressure. The air traveling underneath the wing is deflected at a slight angle, the wind is forced downward, and according to Newton's Third Law of Motion, "for every action there is an equal and opposite reaction", and the downward wind causes an upward reaction, resulting in lift.

2.2.2 Drag

Like everything else in the world, there is an opposing force that acts on the glider. Similar to friction opposing movement of objects across solid surfaces, the force that opposes movement of a glider through air is known as drag. Total drag is the sum of two types of drag; parasite drag and induced drag. Parasite

---

[8] Adapted from Figure 3-2 from FAA's "Glider Flying Handbook"

drag is defined as "drag caused by any aircraft surface, which deflects or interferes with the smooth airflow around the glider" (FAA 2003). Parasite drag is then broken up into three forms:

- Form drag
- Interference drag
- Skin friction drag.

Form drag is caused when the airstream makes contact with plane and is separated to flow either above or below the plane, resulting in some turbulence. Figure 9 shows how the streamline design of the wing reduces form drag. When different currents of air interact over the body of the plane and cause turbulence, this is known as interference drag. The last form of parasite drag known as skin friction drag is caused when thin layers of air adheres to the microscopically rough surface of the plane, creating small currents and turbulence (FAA 2003).
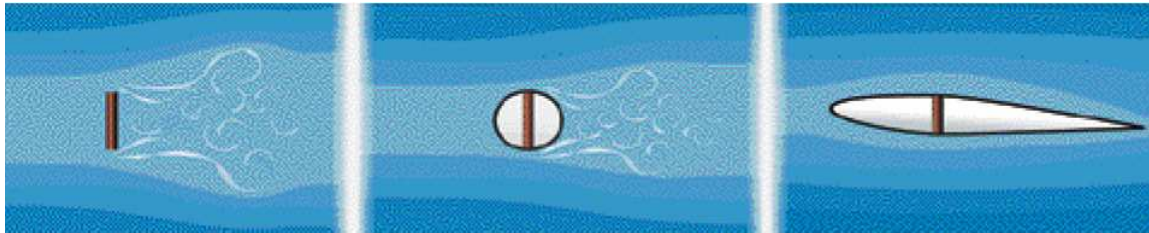
Figure 9 - Reduction of Form Drag Using Streamlined Wing[9]

Induced drag is a byproduct of the lift created by the air moving around the wings. As the airstreams move across the wings, and the low pressure air meets the high pressure air at the trailing edge, a vortex is formed. The vortex itself deflects the airstream downward, in this case an average relative wind in which the wing is flying through. Since lift acts perpendicular to the relative wind, a portion of the lift is directed toward a rearward direction. This rearward lift is known is induced drag (FAA 2003). Figure 10 illustrates the summation of parasite drag and induced drag to form total drag with respect to velocity. The best speed-to-fly to achieve maximum L/D is found where total drag is at a minimum.
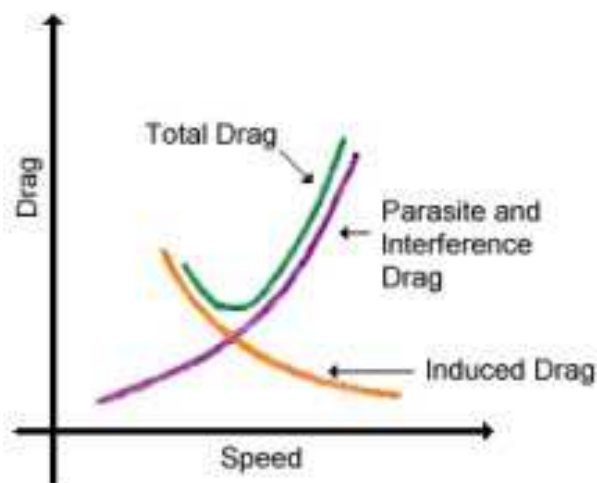
Figure 10 - Total Drag[10]

---

[9] Figure 3-9 from FAA's "Glider Flying Handbook"
[10] http://www.pilotsweb.com/principle/liftdrag.htm

2.2.3 Effective L/D

Effective L/D, previously mentioned as max glide ratio, is the unit of distance traveled divided by the unit loss in altitude. Since effective L/D is a ratio, any distance unit can be used without the necessity for using a particular measuring system. The effective L/D is as significant, if not more, than the manufacturer specified L/D since the glider will not always achieve the specified L/D. The effective L/D approximation used for this project represents the effect of all factors from the pilot's flying style to various temperatures experienced at different altitudes, and the movement of the air through which the glider is flying, and the movement of the air through which the glider is flying. Table 1 is comprised of some typical sailplane data, such as minimum sink rates, best glide rations, and speeds at which they are achieved. As seen in the table, values vary with change in weight and wingspan, but these are not the deciding factors. Other factors such as airfoil shape, wing area, and any additional weight added by having an additional pilot or filling the water ballasts will have an effect on these values.

Sailplane Type	Empty Weight	Wingspan	Min. Sink and Speed Achieved	Best L/D and Speed Achieved	Va (Max Rough Air Speed)
304C	235 kg	15.0 m	112.2 ft/min @ 41.6 kt	42.5 @ 62.6 kt	108.0 kt
DG-300	245 kg	15.0 m	116.1 ft/min @ 42.0 kt	41.0 @ 54.0 kt	108.0 kt
LS-4	245 kg	15.0 m	120.1 ft/min @ 43.2 kt	40.2 @ 54.0 kt	112.0 kt
1-26 E	172 kg	12.0 m	155.5 ft/min @ 33.0 kt	23.0 @ 39.0 kt	unavailable
PW-5	190 kg	13.4 m	128.0 ft/min @ 40.0 kt	32.0 @ 44.0 kt	79.0 kt
2-33	272 kg	15.5 m	185.0 ft/min @ 36.5 kt	23.0 @ 43.4 kt	unavailable
L23 2-Seater	310 kg	16.2 m	159.4 ft/min @ 37.0 kt	28.0 @ 49 kt (2 pilots), 43 kt (1 pilot)	81.0 kt
Duo Discus 2-Seater	410 kg	20.0 m	114.2 ft/min @ 45.9 kt	45.0 (speed unavailable)	97.2 kt

Table 1 – Typical Sailplane Data

2.3 Soaring Weather

All glider pilots, regardless of experience, have to decide if the weather is safe enough to fly. Even if the weather does not pose a hazard to the safety of the pilots, a pilot may decide not to fly on certain days since the weather may not be favorable to producing thermals worth the expense of a tow. In order for the

pilot to make a well-educated decision concerning the safety risk of the weather, a few factors of weather must be considered.

The fundamental factors of weather are temperature, density, and pressure yet humidity should be considered as well since water vapor can easily make for a rough flight with cold temperatures. The definitions for the three fundamental factors plus humidity are as follows.

1) temperature - the average kinetic energy of molecules
2) density – mass of molecules per unit volume
3) pressure – "force per unit area (FAA 2003)"
4) humidity – concentration of water vapor in the atmosphere

Ignoring humidity for a moment, dry air behaves close enough to an "ideal" gas that the extra effect of water is considered negligible yielding the equation P/DT=R.

- P is Pressure
- D is Density
- T is Temperature
- R is a constant

Density is usually calculated from the temperature and pressure readings. The pilot can then determine the performance of the aircraft and make flight adjustments as necessary. The most important part of density is the affect it has on the performance of the aircraft when ascending and descending to different altitudes. The air temperature drops 2ºC for every 1000 feet rise in altitude. Figure 11 shows the temperature based on a given altitude for the standard atmosphere of the Earth. An important note: altitudes above 10000 feet have such a low density of oxygen that breathing may become difficult or even cease function. Commercial jets have pressurized cabins to avoid this problem but glider pilots must be prepared with oxygen if intending to fly at such high altitudes.

In order to get a nice day of soaring weather, the atmosphere needs to be unstable but not as in a tornado, hurricane, or thunderstorm. The air needs to be dry enough to allow the sun to heat the surface and have cool air masses above; otherwise the thermals will be weak at best.
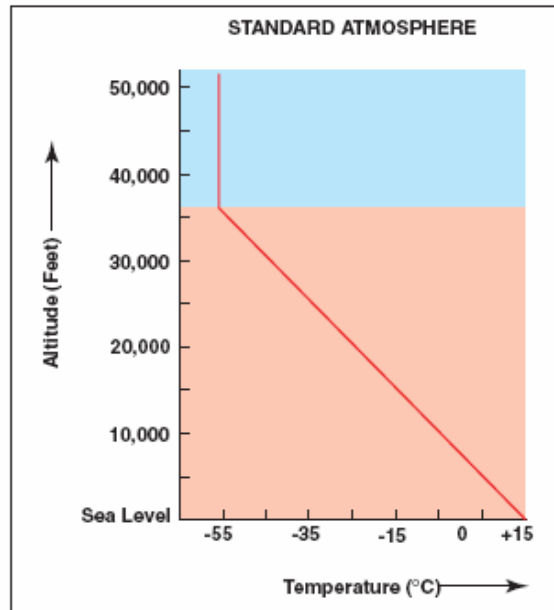
Figure 11 - Standard Atmosphere[11]

Slope soaring requires two simple conditions for a good flight: elevated terrain and wind. When wind encounters topography, the wind is deflected in some direction(s). Many pieces of terrain do not provide enough wind in one direction to allow for slope soaring since the wind can deflect around the terrain instead of over it. Figure 12 shows the prime location to find the best lift when slope soaring.
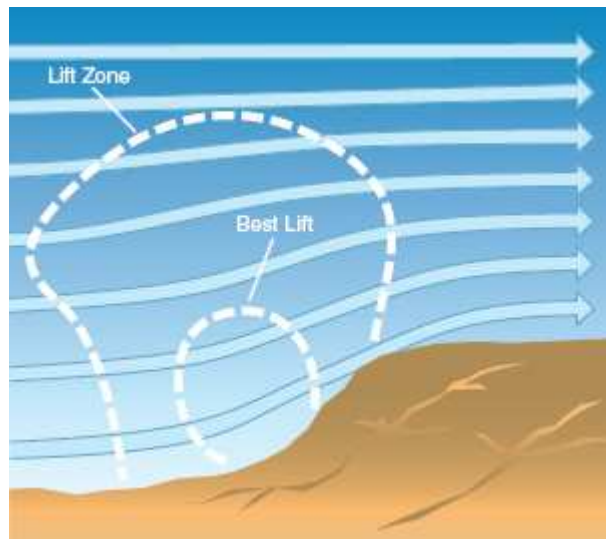
Figure 12 - Slope Soaring[12]

An intriguing phenomenon with wind is lift due to convergence. As seen in Figure 13, convergence occurs when two winds meet and produce an upward stream of air. Convergence can happen for several miles (e.g. two storms collide). Convergence also happens when wind slows down in front of more wind with the excess forced in the upward direction.

---

[11] Figure 9-3 from FAA's "Glider Flying Handbook"
[12] Figure 9-22 from FAA's "Glider Flying Handbook"
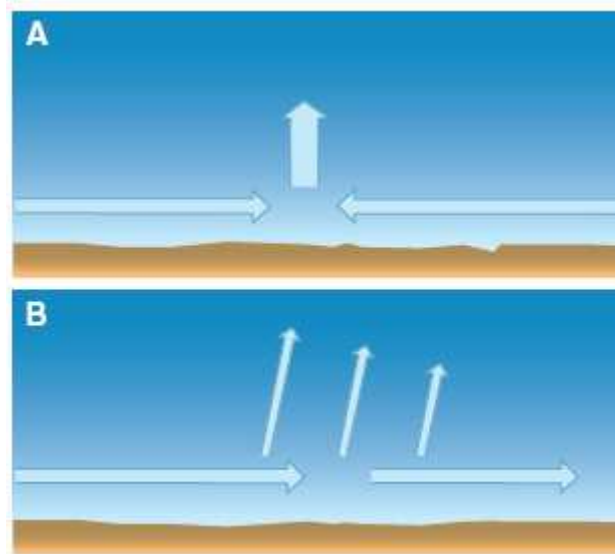
Figure 13 - "Convergence examples. (A) Wind from different directions. (B) Wind slows and 'piles up.'"[13]

2.4 Global Positioning System (GPS)

Through the course of time, there has been a strong desire for the ability to accurately navigate the world. At the beginning of time, travelers were forced to either memorize their way around or mark trails for themselves so that they find their way back. Eventually, people were able to follow maps, but these maps were hand drawn and inaccurate. One of the earliest forms of traveling great distance was by sea, in which sailors used the stars to chart their paths but this was a problem because the stars were only visible at night. From there, different devices such as the compass and the sextant were developed to help with navigation. With each device, there was an associated map which would tell the navigator information relative to the devices reading. For example, as we all know the compass always points to Magnetic North, which tells the user their heading or the direction they are traveling in. These users carried along maps which contained data in which heading they should follow to arrive at a desired location. Eventually radio based navigation centers were developed but this allowed for the following two possibilities:

1. Highly accurate navigation that did not cover a wide area (GPS Primer 3).
2. Navigation that covered a wide area but was not accurate. Radio based navigation centers in turn lead to the development of the Global Positioning System (GPS Primer 3).

2.4.1 History of GPS

The development of the Global Positioning System (GPS) began in the 1960's and the first set of GPS satellites were launched between 1978 and 1985. This first set of satellites was known as Block I and consisted of 11 satellites, one of which was lost due to launch failure. Starting in 1989, launching of the second generation of satellites began. This second generation of satellites is known as Block II and consists of 24 satellites. The constellation of satellites was completed in 1997, and beginning in 1997 NASA began launching more satellites into space to replace the older satellites (GPS Primer 2). Figure 14 illustrates the constellation of the 24 satellites.

---

[13] Figure 9-30 from FAA's "Glider Flying Handbook"

Figure 14 - GPS Satellite Constellation[14]

The satellites are located approximately 12,000 miles above the Earth's surface and make two complete obits every 24 hours. The constellation of satellites is setup such that at any given point, at least four satellites are "visible" to a GPS receiver (HowStuffWorks). The higher the number of satellites the receiver is able to lock onto, the more accurate it is at determining its position. On the GPS receiver, almanac data containing the approximate locations of the satellites is stored and constantly updated with information received from the satellites.

2.4.2 Wide Area Augmentation System (WAAS)

Wide Area Augmentation System (WAAS) is being developed by the Federal Aviation Administration (FAA) and the Department of Transportation (DOT) for approved "use in precision flight approaches." WAAS consists of an estimated 25 US ground stations used to monitor GPS satellite data. Currently, WAAS ground stations only exist in the US but several WAAS capable GPS receivers exist in other countries.

To this effect, other governments are developing their own analogous system. The Japanese Multi-Functional Satellite Augmentation System (MSAS) is used in Asia whereas the Euro Geostationary Navigation Overlay Service (EGNOS) is used in Europe. Figure 15 shows a map of the areas covered by each system while also showing the areas not yet covered by a precision GPS system.

---

[14] http://www.garmin.com/aboutGPS/
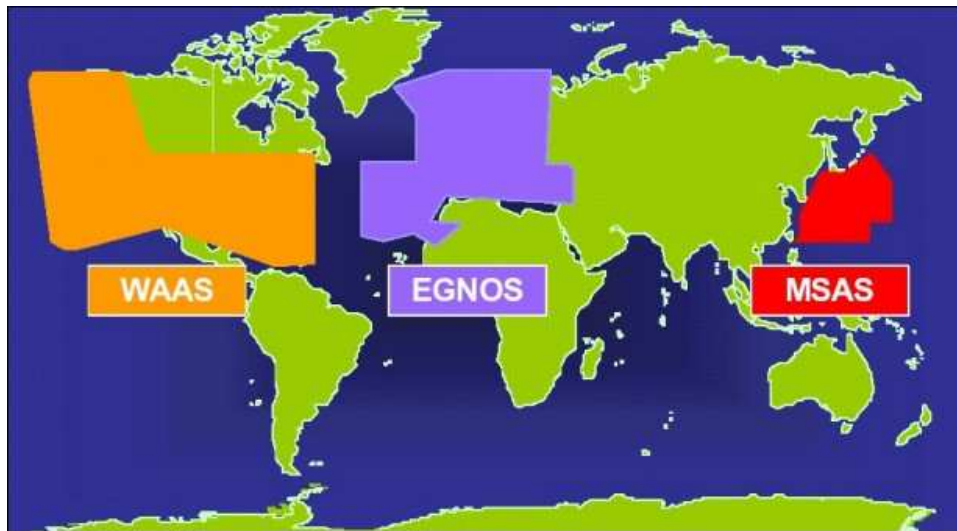
Figure 15 - Precision GPS System Coverage Map[15]

The main purpose of WAAS is to increase GPS accuracy drastically. Figure 16 shows the different position accuracies of the various Global Positioning Systems.

- Original GPS with Selective Availability (SA) enabled position accuracy: 100 meters
- Typical GPS with SA disabled position accuracy: 15 meters
- Typical differential GPS (DGPS) position accuracy: 3-5 meters
- Typical WAAS position accuracy: <3 meters
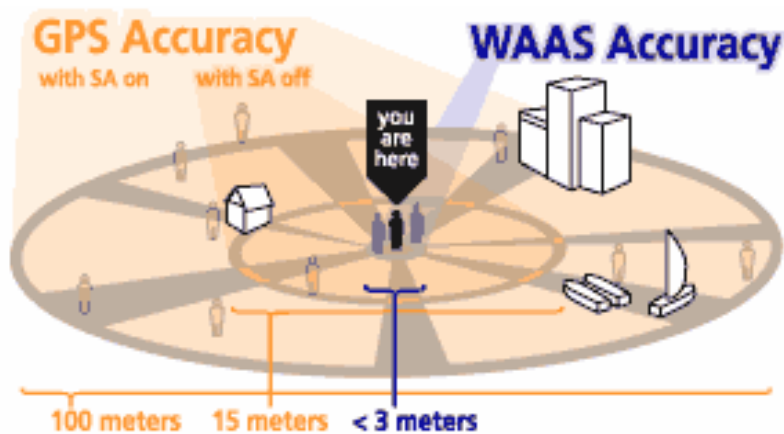
Figure 16 - GPS Accuracy Comparison[16]

In order to achieve an accuracy of less than 3 meters, WAAS uses the 25 ground stations to send correction messages from two master stations with one located on the east coast of the US and the other on the west coast of the US. The two master stations send a correction message which "accounts for GPS

---

[15] http://www.easydevices.co.uk/sitepage/WAAS.html

[16] http://www.garmin.com/aboutGPS/waas.html

satellite orbit and clock drift plus signal delays caused by the atmosphere and ionosphere." Figure 17 shows how the system works while including the following steps.

1) Ground stations acquire satellite velocity and position
2) Ground stations transmit data to master station
3) Master stations transmit GPS correction message to geostationary satellites or satellites with a fixed position over the equator
4) Geostationary satellites send GPS data to vehicles
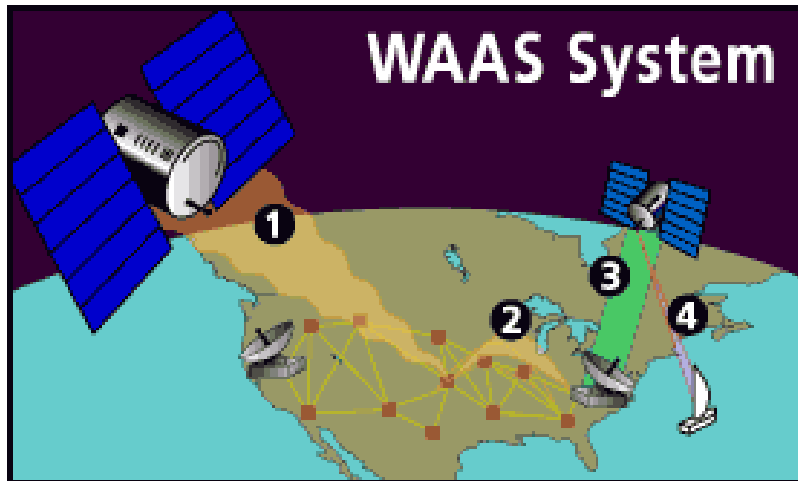
Figure 17 - The Four Steps of the WAAS[17]

2.4.3 World Geodetic System 1984 (WGS84)

World Geodetic System 1984 (WGS84) is the current system used today for land, air, and sea navigation. WGS84 allows us to use GPS since everyone around the world uses the same geodetic, three-dimension coordinate, system. Web sites including http://en.wikipedia.org/wiki/WGS84 explain the details of WGS84.

Knowing GPS uses a three-dimensional coordinate system with the only main difference between our current system and WGS84 is accuracy while used around the world is sufficient.

2.4.4 How it works

The GPS determines the location of a user through a process called trilateration. In order for trilateration calculations to be accurate, at least four satellites must be visible to the receiver. By locking on to a satellite, a GPS receiver is able to determine its distance away from the actual satellite. At midnight (HowStuffWorks), a satellite and the GPS receiver will simultaneously begin transmitting the same signal called a pseudo-random code. When the GPS receiver receives the signal from the satellite, the time offset of the signal is used to determine the distance the receiver is away from the satellite. Using this distance, an imaginary sphere with a radius equal to the distance of the receiver is created around the receiver. According to this sphere, a user can be located anywhere along the edge. By overlapping another similar sphere from another satellite, the user's location is now somewhere along the edge of a perfect circle. By

---

[17] http://www.garmin.com/aboutGPS/waas.html

adding another satellite connection, all possible locations have been eliminated except for two. Then a fourth satellite is used to eliminate the remaining possibility. In essence, the Earth can be used as a fourth sphere to eliminate the remaining possibility lying in space, but with information from an actual fourth satellite, the results will be more accurate. Figure 18 depicts the results of trilateration when data from only three satellites are used.
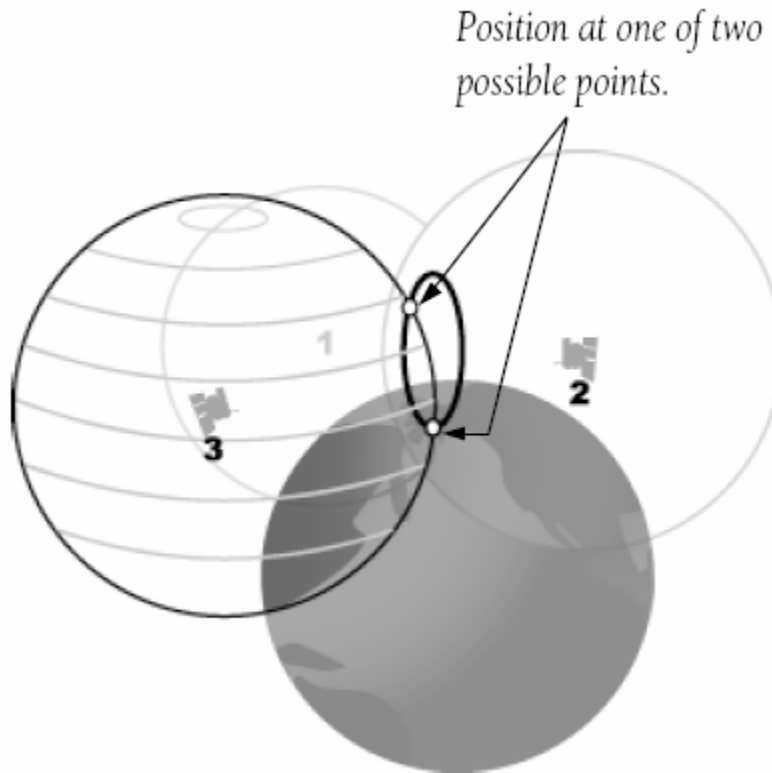
Figure 18 - Trilateration with only 3 Satellites[18]

In GPS applications, there are two signal carriers and two types of pseudo-random code, the first of which is called C/A code (course acquisition). The first carrier is called L1 and has a frequency of 1575.42 MHz which is modulated by the C/A code. Each of the satellites has its on C/A code so receivers can distinguish which satellite is which. Civilian receivers use the C/A code, whereas the military uses the second type of code, called P (Precise) code. The P code modulates both the L1 carrier and the L2 carrier, which as a frequency of 1227.60 MHz, at 10 MHz. For extremely accurate applications, such as missile guidance, the P code can be encrypted, which is called Y code (GPS Signals). As a matter of national security, only receivers with authorization keys and special decoders can receive Y code signals.

In order for trilateration to work, timing must be highly accurate. This poses a problem on the receivers behalf, because all the satellites have atomic clocks installed but would be too expensive and impractical for a receiver to also have an atomic clock. Instead, GPS receivers contains a quartz clock which is updated constantly using the information from received from the GPS satellites (HowStuffWorks).

---

[18] Garmin Part Num. 190-00224-00 Rev. A – GPS Guide For Beginners

There are many factors that can cause GPS information to be inaccurate, such as timing. As the signals travel through the atmosphere, they are slowed down which causes an inaccurate calculation of the distance between the receiver and the satellite. Another timing issue is caused by obstacles in the path of the GPS signal. If there are buildings and trees in the way, the signal is bounced around and takes longer to arrive at the receiver. Probably the most common error in GPS data is caused when the true orbit of the satellites do not match up to the almanac data stored in the GPS receiver. This will result in errors that offset the receivers' longitude and latitude coordinates relative to the satellites actual location (GPS Guide 8-9).

By using a Differential GPS (DGPS) equipped receiver, accuracy is greatly increased. Normal GPS receivers are accurate to within fifteen meters, whereas DGPS equipped receivers are accurate to within three to five meters (Garmin – What is GPS?). There are stations with known locations which monitor the locations of the satellites and compare it to the data specified in the almanac. The stations then determine the errors in the satellites' signals and then send out corrective information to DGPS equipped receivers. Accuracy of DGPS systems improves horizontal accuracy to within 1-6 meters as opposed to the 3-5 meter accuracy of WAAS (GPS Guide 11). GPS Control states as a general rule of thumb "to expect your vertical accuracy to be somewhere near half the horizontal accuracy". For example, if the horizontal accuracy is 1 meter, expect the vertical accuracy to be 2 meters.

2.5 Summary

This chapter presents the most pertinent background information necessary to understand the remainder of the Flight Data System project. Without a basic understand of the Global Position System and the meaning of the L/D ratio, this project would seem very confusing. Contrarily, this project is simple in the concept.

3   Problem Statement

3.1 Introduction

The goal of our project was to design, implement and test a portable sailplane flight performance instrument. The purpose of this chapter is to specifically state the project goal, identify the objectives necessary to achieve the goal, and list the tasks necessary to accomplish the objectives and the overall goal.

3.2 Problem Statement

The goal of the Flight Data System project was to design, implement, and test a fully independent portable instrument capable of calculating the effective L/D ratio of a sailplane using only GPS data and storing pertinent data onto removable media for analysis.

3.3 Objective Statements

After considering our overall goal, we came up with the following objectives:

- Develop a detailed working knowledge of the Global Positioning System (GPS).

- Develop a detailed working knowledge of sailplane aviation and flight performance factors.

- Understand the incoming GPS string data format, content, and configuration options.

- Design, implement, and test the system hardware.

- Design, program, and test software used to interconnect all necessary system components.

3.4 Tasks

In order to complete the objectives listed above, we divided each objective into a list of tasks needed to reach the object:

- Develop a detailed working knowledge of the Global Positioning System (GPS)

  o   Define Global Positioning System and all necessary subsystems

- o Research how the Global Positioning System works

- o Research the means of communication within the system including standards and methods

- o Research commercially available receivers

- Develop a detailed working knowledge of sailplane aviation and performance factors

  - o Research performance glider (sailplane) aviation

  - o Research the most crucial characteristics concerning sailplanes

  - o Research required flight instruments

  - o Observe modern racing sailplane concerning cockpit layout and proper placement of Flight Data System device

- Understand the incoming data format and content and configuration options

  - o Research GPS data transmission syntax and content along with configuration options

  - o Research the National Marine Electronics Association (NMEA) 0183 Standard

- Design, implement, and test the system hardware

  - o Research and contrast commercially available microprocessor and field-programmable gate array (FPGA) boards based on communication standard requirements

  - o Research and contrast liquid crystal display (LCD) implementation options (e.g. serial versus parallel)

  - o Research and contrast commercially available receivers

  - o Research various types of removable media formats

  - o Research and contrast commercially available removable media

  - o Research common portable device independent power sources

  - o Research and contrast power distribution topographies based on the various common portable device independent power sources for simplicity of design and constrained space requirements.

- o Prove all chosen system components can physically fit onto a printed circuit board (PCB) which falls within the space constraints of the enclosure.

- Design, program, and test software used to interconnect all crucial system components

  - o Research necessary programming language for microprocessor or FPGA

  - o Create a software flow chart for clarity of organization before programming

  - o Simulate test flight and verify the L/D ratio results

3.5 Measurement Objectives

After researching the characteristics and capabilities of racing sailplanes, we created a list of measurable objectives needed for our instrument to work over a broad range of performance racing sailplanes:

- Air Speed: 30 – 150 knots

- Altitude: sea level to 9,999'

- L/D ratio: 2:1 – 100:1

- Total system runtime of no less than 1 hour but 3 or more hours is preferred.

3.6 Summary

The goal of the Flight Data System project was to design, implement, and test a fully independent portable instrument capable of calculating the L/D ratio of a sailplane. In order to successfully complete this goal, we specified a set of objectives our project would have to satisfy in order to be considered complete. The set of measurable requirements is given in the following chapter –Methods– since we had to research several areas before determining which measurable requirements are necessary for the successful completion of the Flight Data System project.

4   System Design

4.1 Introduction

The overall system design of the Flight Data System is presented in this chapter. All inputs, outputs, and processing needs are defined and explained within the context of the system design. Where appropriate, the rationale for the design chosen is also presented.

4.2 Overall System Design

Before we could move onto implementation and choosing various system components, we had to define the critical components necessary to meet our design goals and objectives. As seen in Figure 19, several key modules were apparent when we referred to the Problem Statement and Measurement Objectives chapters including the following:

- GPS Module
- Liquid Crystal Display (LCD)
- Removable data storage media (prefer onboard)
- User Interface (push-button switches)
- System Controller (a microprocessor or FPGA)
- Portable Power Source
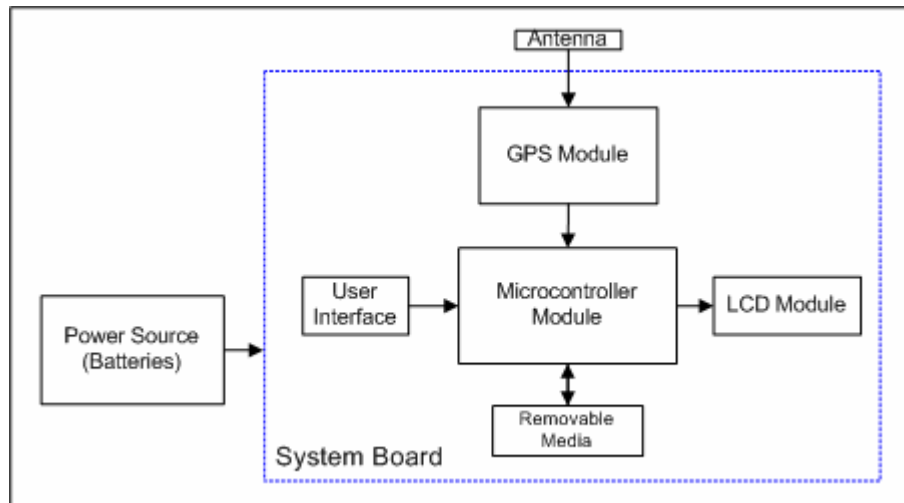
Figure 19 - System Block Diagram

4.2.1 Enclosure

One of the most critical goals we needed to satisfy, to be able to use the Flight Data System during a flight, is to meet the necessary size requirement of the enclosure as to not impair the pilot's view. Figure 20 illustrates the need for the small size requirements since the only place to attach the device to the sailplane is in place of the clock due to the limited space available. Because the clock is being replaced, a clock will be implemented as one of the displays in our system. At the start of our project, Professor Looft provided us with an enclosure he felt adequate to implement the system, yet small enough to fit the space constraint in the cockpit. The enclosure measured 3.125 inches by 6.5 inches.

Figure 20 - Sailplane Cockpit

4.2.2 GPS Module

The GPS module consists of the GPS antenna, GPS receiver, regulated power, and a true RS232 data line from the GPS receiver to the microcontroller board as seen in Figure 21. The GPS receiver receives the NMEA strings though the GPS antenna and transmits the strings to the serial port of the microcontroller via a true RS232 serial data line. The GPS data includes altitude, speed, time, etc. and provides the only vital data input to the system controller.
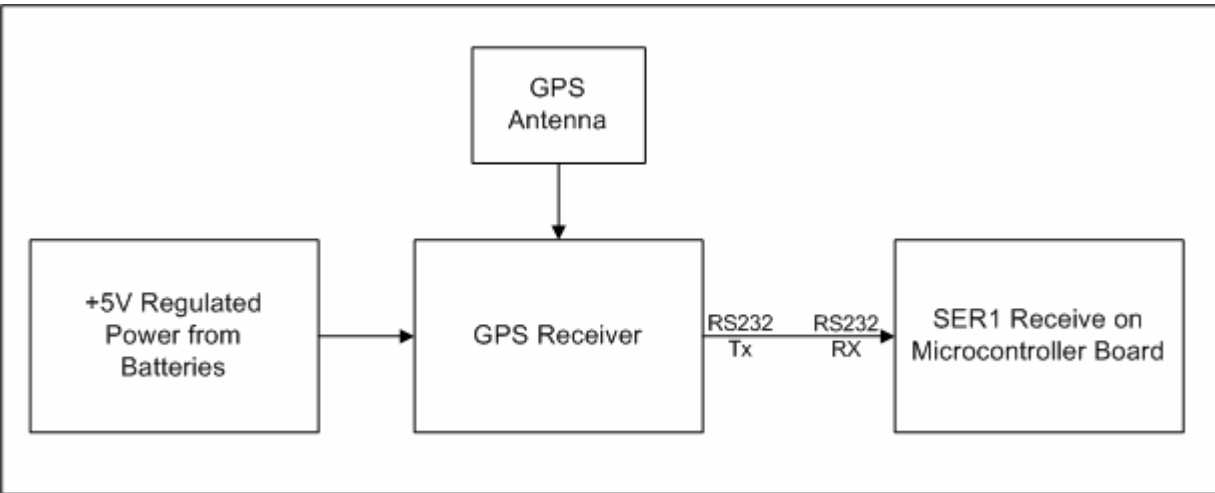
Figure 21 - GPS Module Block Diagram

4.2.3 LCD Module

The LCD can be interfaced to the microcontroller either via a serial or parallel connection. We decided to use the parallel interface since serial only transfers one bit at a time as opposed to the 10 bits simultaneously with the parallel interface. As seen in the LCD module block diagram in Figure 22 , the LCD module consists of a parallel LCD which connects to the microcontroller via 8 data lines, chip enable (E), read/write (R/D), and register select (RS). The LCD module provides the only available user feedback concerning the current state of the system.
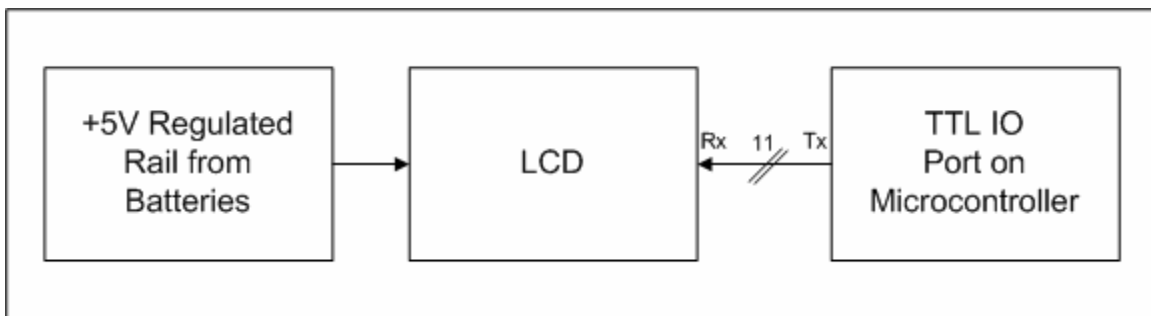
Figure 22 - LCD Module Block Diagram

4.2.4 Removable Data Storage Media

The purpose of the removable data storage media is to transfer the calculated data to another system for analysis (e.g. MS Excel on a computer). Therefore, the only two requirements are a device with a format any computer can read, with the appropriate hardware, and ample storage capability to hold the calculated data for the duration of at least one flight.

The size requirement of the removable data storage media relies heavily on the implementation of the software. In our design, we only have the data recorded after the user presses a switch to activate the

effective L/D calculation. After five seconds to prepare and ten seconds to take measurements, the final effective L/D is calculated and stored onto the removable data storage media along with the date, time, average velocity, initial altitude, and final altitude. We chose not to have the calculation run continuously in order to optimize available space on the removable data storage media. Having a calculation based on the input of the user allows readings only when the pilot has readied the plane for a calculation and has noted the weather conditions and other sources of error.

Based on our initial research, one microcontroller included an onboard CompactFlash card device. There may be other microcontroller boards available with a similar onboard feature, but we were not able to find any. At that point in our research, the FlashCore-B(FB) was highly favorable.

4.2.5 User Interface

Based on our overall design, we would need an ON/OFF switch for power and momentary pushbutton switches. The momentary pushbutton switches contain the inherent problem of what's known as bounce. Bounce happens when you close a mechanical switch. After the initial close, vibrations in the switch cause it to open and close several times. Bounce can be minimized either by a software delay or a hardware delay. We chose to use the hardware delay since hardware is more straightforward to debug than software and would be simpler to design. Figure 23 shows how the user interface is connected through a debouncing circuit to the microcontroller.
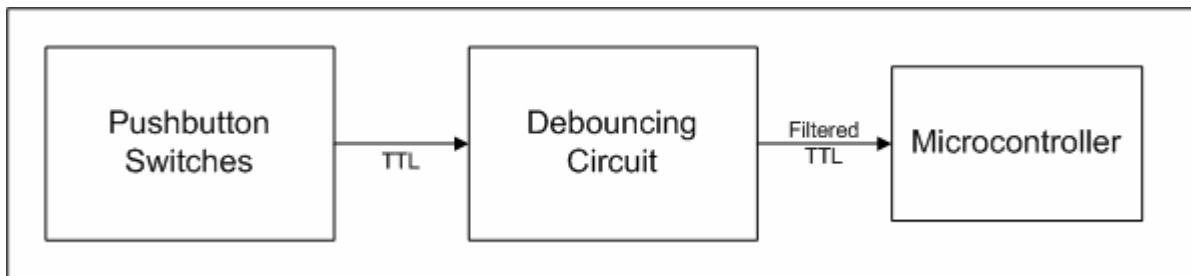
Figure 23 - User Interface Block Diagram

4.2.6 System Controller

The system controller provides the processing power of the system, performs all of the calculations, outputs calculated and received data to the LCD, and stores calculated data onto the removable data storage media. Figure 24 illustrates the connections between the microcontroller board and the other modules in the system. Note that the GPS receiver, Regulated power, and User Interface provide inputs only to them system and the removable media storage device and LCD provide outputs only. Based on the modules of the other necessary components in the system, we need 10 TTL inputs for LCD, 3 TTL inputs for the user interface, and one true RS232 serial port for the GPS receiver.
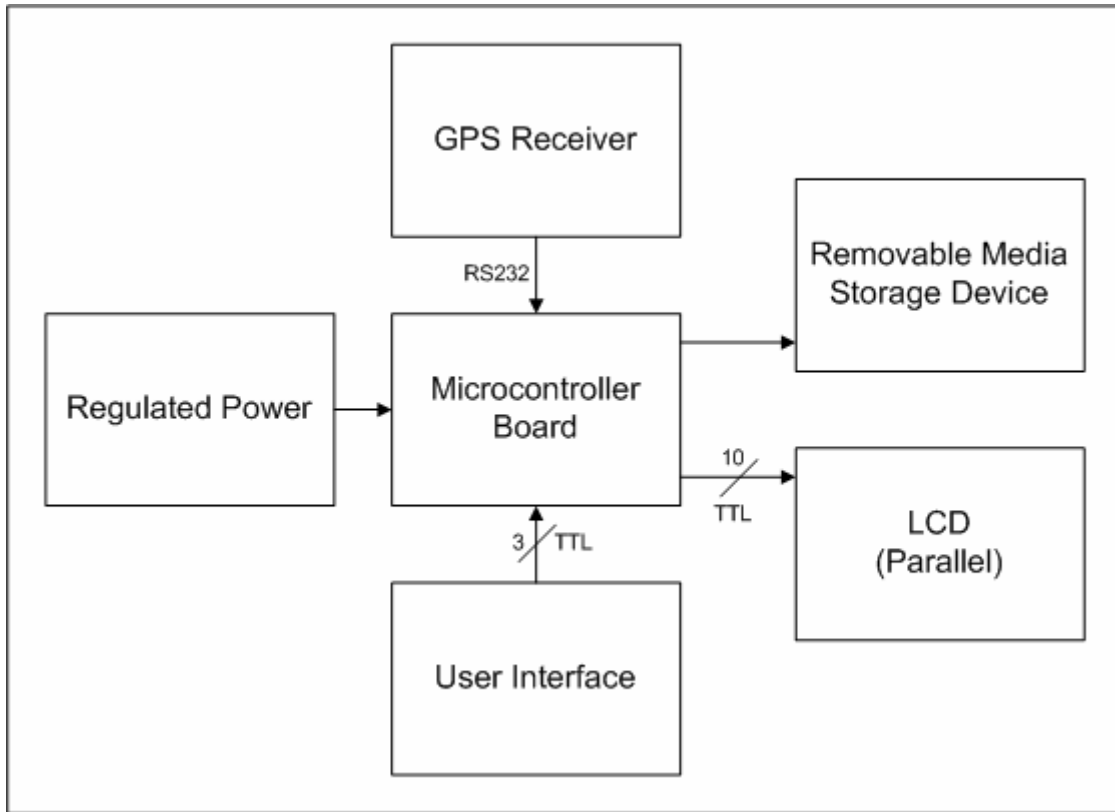
Figure 24 - Microcontroller Block Diagram

4.3 Component Selection

Now that the functionality requirement of each module has been defined, we can present our component selection. For any system, the most important system component is the microcontroller module. The microcontroller module determines what programming language the design will require, if any extra modules are required (e.g. external true RS232 transceiver), and the interface to the removable storage media as well as its type.

4.3.1 GPS Receiver

The first component that we had to find was the GPS receiver because it provided all of the vital data necessary for our system. From our searches, we found that the only manufacturer that provided GPS receivers intended for development and integration is Garmin, a world leader in providing GPS equipment including GPS receivers. Other world leaders in providing GPS equipment including Magellan and TomTom only provide consumer products with the GPS receivers already developed into some sort of mapping system, rather than a standalone GPS engine required by our system.

Garmin currently has three models available; the GPS 15, GPS 15H, and GPS 15L. We then gathered information regarding size, accuracy, power requirements, and some other additional information such as

number of channels (maximum number of satellites the unit can track at once) and acquisition times (time it takes to find satellites). Table 2 was generated using the information found on each of the units.

The most important characteristic of a GPS receiver for our project is dimensions. Seen in Figure 25 is the Garmin GPS 15L receiver, which measures a mere 1.400" x 1.805" x 0.327", just slightly larger than the GPS 15. Although larger, it can achieve an accuracy of 3-5 meters when WAAS enabled and the DGPS information is available, otherwise it is just as accurate as the GPS 15. The receiver runs off of a 5V regulated input voltage whereas the 15H runs off of an 8-40V unregulated input voltage, which is out of our voltage range. For our system, we decided to go with the GPS 15L over the GPS 15 because of the better accuracy, even though there are slightly larger space power requirements.

Model	Size	Position Accuracy	Velocity Accuracy	Channels	Min Acquisition Time	Max Acquisition Time	Voltage	Current	Price[19]
GPS 15[20]	0.94" x 1.69" x 0.30"	Standard < 15m DGPS N/A	0.1 kt	12	2 sec	5 min	3.3V	75mA	$47.99
GPS 15H/L[21]	1.40" x 1.80" x 0.33"	Standard <15m DGPS < 3 (WAAS)	0.1 kt	12	2 sec	5 min	15H 8V-40V 15L 3.3V – 5.4V	15H 60mA @ 8V 15L 100mA	$51.99

Table 2 - GPS Receiver Comparison
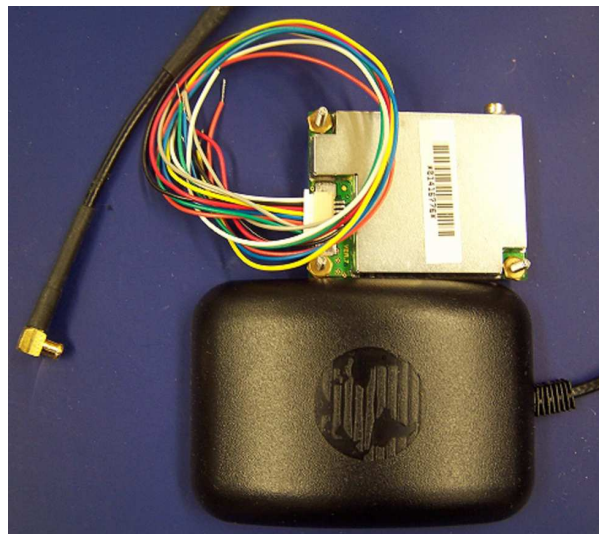
Figure 25 - Garmin 15L GPS Receiver with Antenna

---

[19] http://www.megagps.com/index.asp?PageAction=VIEWCATS&Category=19
[20] http://www.garmin.com/manuals/GPS15_TechnicalSpecification.pdf
[21] http://www.garmin.com/manuals/237_TechnicalSpecifications.pdf

4.3.2 LCD

When choosing an LCD module, there were a few factors that have to be considered. The LCD should be chosen based on the balance between cost and display capability including size, power requirements, and data interface (parallel versus serial). As mentioned earlier, we had decided to go with a parallel LCD since the parallel connection would be faster. Upon searching for LCD modules, we found that LCD's capable of displaying 8-characters by 2-lines and 16-characters by 2-lines were most common. From there, we drafted an LCD screen-flow diagram to illustrate the different displays that we felt were necessary. We based our drafts based on the 8x2 display capability and found that it was sufficient for our purposes.

Once we decided that we were going to use an 8x2 character LCD, we had to compare the different options based on the other factors previously mentioned. Table 3 was generated from the data we used to compare various LCD's available to us. As seen from the table, the LCD's manufactured by Maxim Orbital and Orient Display are pretty much the exact same except for price. Although the model from Crystalfontz requires slightly more current when using a backlight and is relatively much more expensive, we decided to go with the CFAH0802A-GYH-JP based on past experiences with Crystalfontz products. Seen in Figure 26 is model CFAH0802A-GYH-JP made by Crystalfontz.

LCD Manufacturer	Model	Dimensions	Character Size	Voltage	Current	Price
Crystalfontz[22]	CFAH0802A-GYH-JP	58mm x 32mm x 13.5mm	2.96mm x 5.56mm	4.75V - 5.25V	1.2mA + 70mA for backlight	$17.99
Maxim Orbital[23]	MOP-AL082B-BYFY	58mm x 32mm x 14mm	2.96mm x 5.56mm	4.5V - 5.5V	1.5mA + 60mA for backlight	$9.95
Orient Display[24]	AMC0802B-B-Y6WFDY	58mm x 32mm x 14mm	2.96mm x 5.56mm	4.5V - 5.5V	1.5mA + 60mA for backlight	$8.00

Table 3 - LCD Comparison

[22] http://www.crystalfontz.com/products/0802a/CFAH0802AGYHJP.pdf
[23] http://www.matrixorbital.ca/manuals/MOP_series/MOP-AL082B/MOP-AL082B.pdf
[24] http://character-lcd-lcds.shopeio.com/inventory/pdf/AMC0802B.pdf

Figure 26 - Crystalfontz CFAH0802A-GYH-JP[25]

4.3.3 Microcontroller Module

Figure 27 shows the system controller block diagram adapted from the TERN manual. Due to the complexity of the calculations utilizing the GPS data, a high-level programming language such as C or C++ is preferred when compared to a machine code or basic-level language such as assembly.
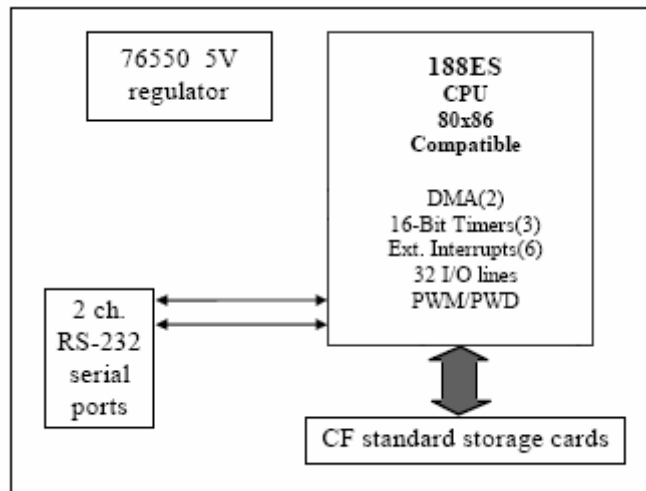
Figure 27 - System Controller Block Diagram[26]

The microcontroller module was the most important component selection since the embedded features, or lack thereof, determined the complexity of the design of the system. For our microcontroller module, we had three primary options. First of all, the FlashCore-B(FB) (seen in Figure 28) was being used concurrently by another team in the same lab and included several important features necessary for our system already on the module. Second of all, the ICOP6015 was over $300 cheaper, as seen in the trade study in Appendix B, but did not include an embedded CompactFlash card reader. Third of all, a

---

[25] http://www.crystalfontz.com/products/0802a/index.html#CFAH0802AGYHJP
[26] http://www.tern.com/docs/fb.pdf

Microchip PIC processor provided the necessary I/O but the whole design would have been programmed in PIC assembly. However, programming our design entirely in PIC assembly would have been extremely difficult when compared to the Paradigm C/C++ software included with the FlashCore-B(FB) since a library of functions for the CompactFlash such as read/write and other necessary functions had already been written. Yet, the ICOP1605 used DOS software to program, which isn't as high-level as the Paradigm C/C++, but would still be easier to program than PIC assembly. Also, the latter two options also would require an additional module for the removable media storage device.

Processor speed does not account for the most important aspects of the CPU selection but more so the peripherals the CPU can interface to including a RS232 serial port and over 20 TTL programmable logic inputs/outputs (IO). The RS232 serial port is a necessity for interfacing to the GPS receiver, and the TTL logic outputs make interfacing to the LCD possible due to a parallel connection to the LCD. Parallel LCD displays provide better solutions to embedded systems, such as this one, due to faster display speeds and ease of use due to the lack of a need for a clock to run the LCD serial line.
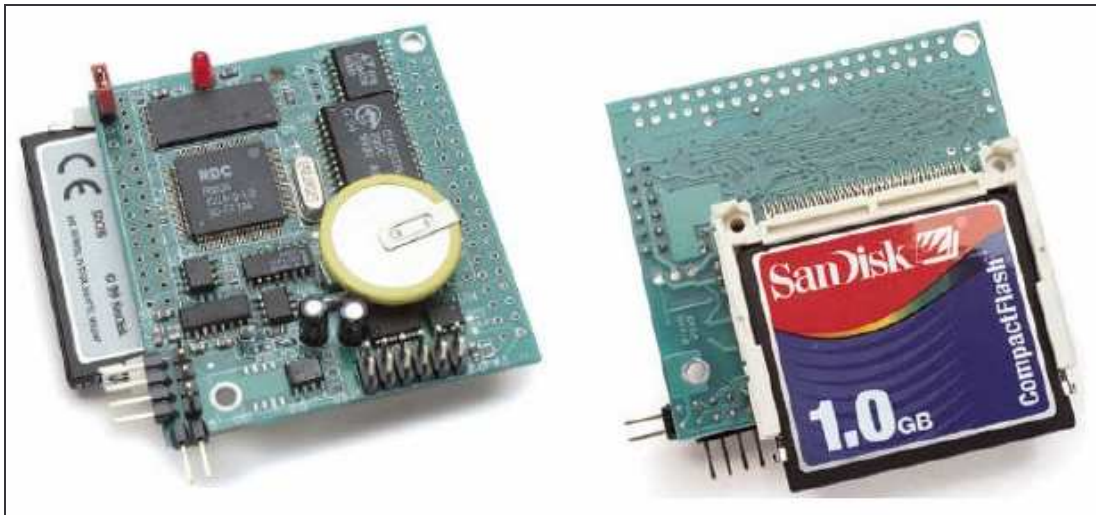
Figure 28 - TERN FlashCore-B(FB) System Controller Board[27]

We chose the TERN FlashCore-B(FB) due to its small dimensions when compared to the dimensions of our enclosure (and its competitors as well), the low power requirements, and the embedded CompactFlash card reader which the included software library contained the necessary read/write functions. Please refer to the trade study in Appendix B for further details.

4.3.4 DC/DC Converter

Our current topology boosts the voltage from the batteries (approximately 4.5V-6V for four AA batteries) to a +12V rail that is distributed to point-of-load (POL) regulators on the system board. This topology provides an added feature of allowing DC power source voltages from 2.3V to 12V for input to the

---

[27] Adapted from the TERN FlashCore-B(FB) Manual

system. Even though the system is designed not to require any external power sources or data, a 9V or 12V battery would be able to be used with only a few modifications to the input connector.

When choosing a DC/DC converter, Linear Technology provides the most reliable and efficient solutions (with efficiencies on most DC/DC converters over 85%). After deciding on the boost then point-of-load buck topology, we researched +12V output boost converters. The Linear Technology site led us to the LT1935 which has a 2.3V to 16V input range converted up to a 38V maximum output voltage with an efficiency over 85% using our input voltage (approximately 5V) which is illustrated in Figure 29.
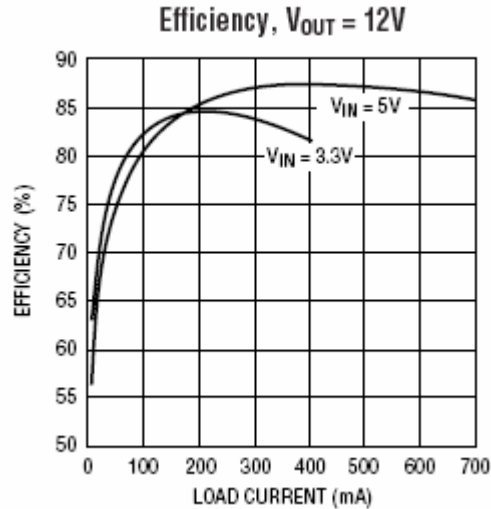
Figure 29 - 5V Input Efficiency Curve[28]

4.4 Software

Figure 30 shows the software flow diagram utilized in our design. One vital element necessary to achieve optimum accuracy and performance is a minimum of four satellites. Whenever less than four satellites are detected, the L/D ratio calculations are no longer valid, and then the system must wait until a sufficient number of satellites have been detected again.

In order to ensure the system has not crashed, an incrementing timer has been included to show continued system activity, which is displayed in the lower right corner.

Incoming GPS data is transmitted via strings, which contain multiple pieces of information. The software looks for the strings containing the pertinent data needed for the calculations. All pertinent data is displayed on the corresponding menus while the L/D calculation results are stored on the CompactFlash card.

---

[28] http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1042,C1031,C1061,P2507,D2339

The user interface consists of two parts: three switches and the LCD. Button 1 is used for a confirmation or enter button (i.e. "Press When Ready"). Button 2 is used to scroll through the various menus. Button 3 is used to jump to the L/D calculator menu.

On power up, the user must acknowledge he/she is ready by pressing button 1. Once acknowledged, the software will search for the number of satellites while simultaneously displaying the current number of satellites and the incrementing timer on the menu. This portion of software does not permit any user input because data is not valid yet. Once connections with at least four satellites have been established, all data is then valid. At any point in time, if the number of connected satellites drops below the minimum number required (four), all data is deemed invalid and the system waits until the minimum number of satellite connections is achieved.

The user is able to scroll through different menus, by using button 2, including velocity, altitude, L/D, current number of satellites, etc.
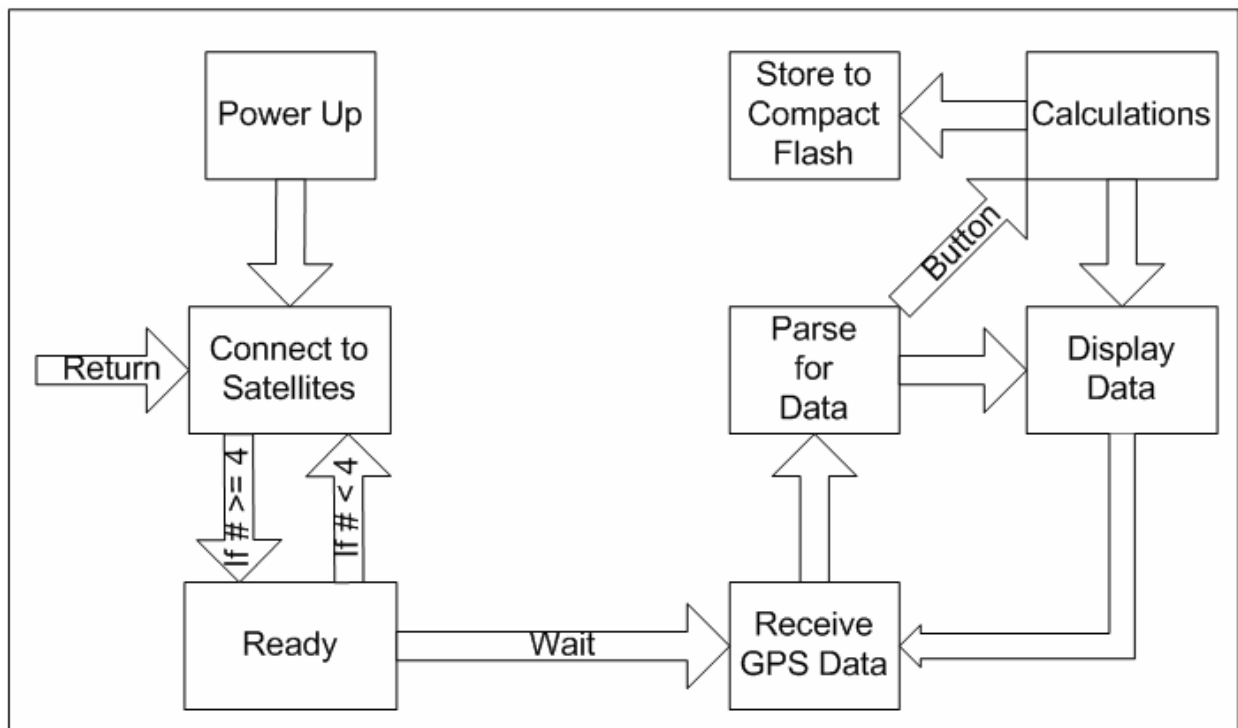
Figure 30 - Software Flow

4.5 Summary

This chapter presented the overall system design of the Flight Data System project by including a functional diagram. First of all, the functional block diagram defined the inputs and outputs of the entire system. Second, the top-level system design diagram outlined the hardware necessary to harness the previously mentioned inputs to obtain the desired outputs. Lastly, the software flow outlined the interaction of software with the various inputs to obtain the desired outputs.

5   Results

5.1 Introduction

This chapter presents the design options available and the components we incorporated into our design. The hardware and software implementations and final test results of the system are included in this chapter as well.

5.2 Hardware Implementation

In order to design a complete system that worked as intended, each module had to be tested individually to ensure hardware functionality alone before interfacing with another component to deter a chain of problems occurring at the same time, thereby avoiding a tedious task of debugging. The module testing and compatibility schedule proceeded as follows:

- FlashCore-B(FB) Hardware Verified
- CompactFlash Write Capability on FlashCore-B(FB) Verified
- LCD Hardware Verified (using hardware method)
- Garmin GPS Receiver Hardware Verified

5.2.1 FlashCore-B(FB) and CompactFlash

In order to test the FlashCore-B(FB) board, we powered it on and waited for the red light-emitting diode (LED) to indicate power is on. We then programmed the board via COM1 on a PC after successfully compiling a Paradigm C/C++ template with no functional code. The next step to verify the FlashCore-B(FB) board was functional was to test the programmable input/output (PIO) pins[29].

In order to test the functionality of the PIO pins, the TTL test board seen in Figure 32 and Figure 33 were built to indicate the state (logic "high" or "low") of all the pins on the jumper used in our design (J2)[30]. Figure 31 shows the schematic of the TTL test board used to test the PIO functionality which corresponds to the pictures in Figure 32 and Figure 33. The software test set all pins to logic "high," defined as 5V, to toggle the active high LEDs on and off. All LEDs were toggled several times with a time delay in between changing states to be able to observe the change.

A 3kOhm resistor was used to limit the current to under the 2mA maximum source limit of the FlashCore-B(FB) PIO pins. The LED drop was approximately 2V with a supply voltage of 5V. A simple

---

[29] Page 16 in http://www.tern.com/docs/fb.pdf
[30] Page 50 in http://www.tern.com/docs/fb.pdf

Ohm's Law (V=IR) calculation yielded a current of 1mA for a 3kOhm resistor, which was exactly 50% of our limit. The tests proved the PIO worked as expected.
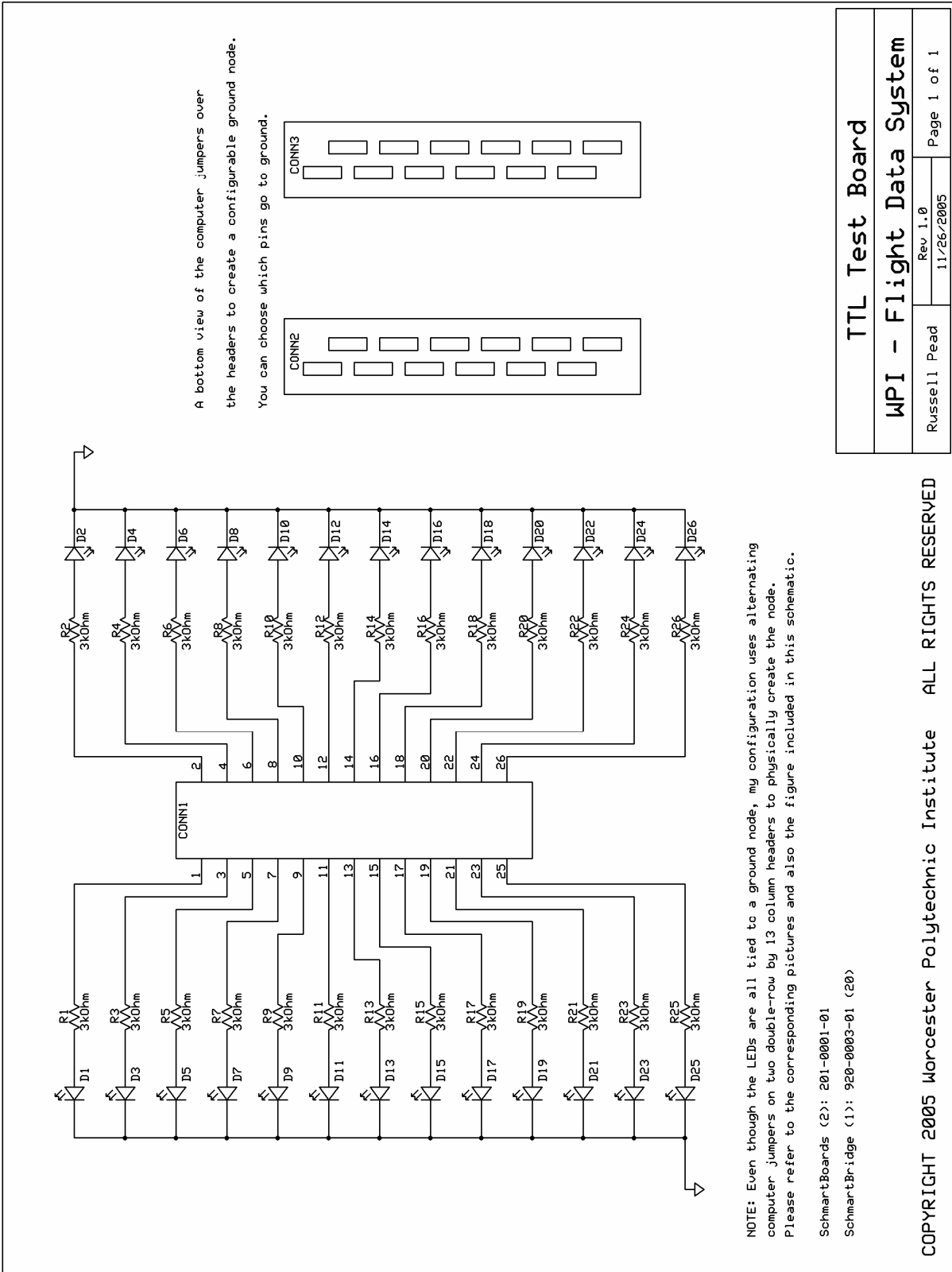
Figure 31 - TTL Test Board
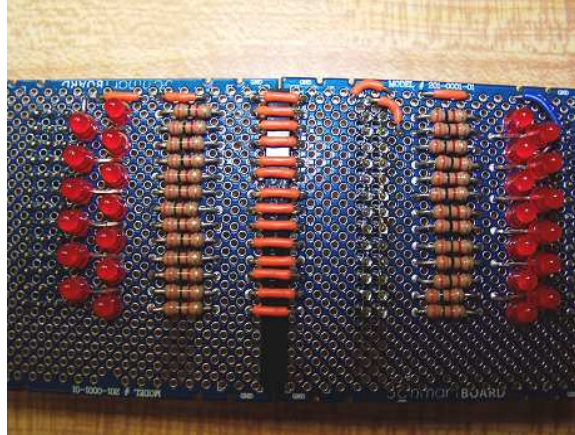
Figure 32 - LED Test Board (Top View)
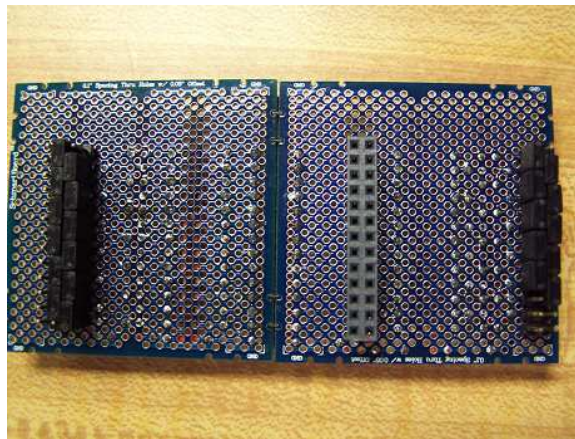
Figure 33 - LED Test Board (Bottom View)

TERN had a library of functions available for the Paradigm C/C++ software including functions that read from or wrote to different specified PIO pins to simplify programming of the available PIO pins.

The CompactFlash card write capability was tested next, since writing to the CompactFlash was a requirement of the project. This task was relatively simple due to the fact that functions for working with the onboard CompactFlash unit were included in the FlashCore-B(FB) package and also the fact that another group in our lab, the Pot Hole Detection project team, was using the same unit. Jose Brache, the key programmer in their group, sat down with us and explained what functions were required to initialize the compact flash and the key functions to reading and writing. From there, we went on to implement these functions in a simple program that created a text file onto the CompactFlash card and then write simple strings into the file.

5.2.2 Crystalfontz LCD 8 x 2

To test our LCD unit, we built a circuit from the schematic found in Appendix C Using dip-switches to control the LCD's inputs, we ran the initialization sequence found in the unit's datasheet. After running the initialization sequence, we then tried displaying characters onto the LCD. Located inside the datasheet, is also a chart containing the characters and what input value to the eight data lines

corresponded to the respective character. It was eventually noticed that a character's input value was the character's ASCII value converted to binary. To control what value, high or low, was being sent to each line, we used dip switches. In the on state, the data line was sent a high and in the off state it was sent a low. Once the dip switches were set, another switch controlled the LCD's "enable" line. The data set on the eight data lines latches on the falling edge of the "enable". Since there was no maximum setup time for the data, timing was not an issue since minimum setup times were in the milliseconds, or smaller.

5.2.3 Garmin 15L GPS Receiver

To ensure that our GPS receiver was working properly, Professor Looft provided us with a serial data logger. After making the necessary connections, GPS receiver's transfer line to the receive line of the data logger and ground to ground, and powered both units before taking them outside. The receiver's manual states that if the receiver has no initial data of its whereabouts, approximately five minutes is required to acquire the necessary satellite connections. Knowing this, we waited for about ten minutes before powering them down and going back inside. To see if there was any data recorded, we then connected the serial data logger to a computer, using a cable created based on the schematic provided to us by a supplier of the logger[31], and used HyperTerminal to see the recorded data. The schematic of the cable and the HyperTerminal instructions can be found in Appendix D. Using the correct HyperTerminal instruction, all data recorded onto the logger was downloaded to the HyperTerminal window and it was seen that the GPS receiver was indeed receiving GPS NMEA sentences.

5.2.4 Power Distribution Requirements and Implementation

Power distribution plays a key role in any design with portability as a requirement due to the need for batteries and a limited amount of power available to the system. The primary component of the power distribution is the power source. Table 4 shows the power analysis for the portable power source between one 9V NiMH rechargeable battery and four 1.2V AA NiMH rechargeable batteries. As seen in the power analysis, even though the 9V battery has a higher voltage than the nominal voltage of 4.8V for the four AA batteries in series, the maximum power of 1.35W available from the 9V battery is extremely low compared to the 9.6W available from the four AA batteries. The four AA batteries only cost $2 more than the one 9V battery yielding a significant cost savings while gaining more total power available to the system.

Battery Size	9V NiMH Rechargeable	Four AA NiMH Rechargeable
Nominal Voltage	9 Volts	4.8 Volts (1.2V nominal each)
mAh Rating	150 milliamp-hours	2000 milliamp-hours
Total Power	1.35 Watts	9.60 Watts
Battery Price	$12.99 for one 9-volt	$14.99 for four AA
Price Per Watt	$9.62	$1.56

Table 4 - 9V Battery Power Versus the power of four AA Batteries

---

[31] http://homepages.tig.com.au/~robk/datalogger.html

Before the method of power distribution can be chosen, an analysis of how much power the system can consume during maximum power consumption must be calculated to ensure the system lasts long enough to endure the duration of a flight to record the necessary data calculated during the flight. Table 3 shows the system power analysis and identifies all devices in the system that consumes power.

The total system power consumption calculated in Table 5 assumes no losses in any of the conversion circuitry. Efficiency of all converters in the system must be accounted for in order to ensure an accurate runtime calculation.

Device	Maximum Current	Operating Voltage	Max. Power Dissipation
LCD Display	1.2mA	5V	6mW
LCD Backlight	140mA	4.2V	588mW
FlashCore-B(FB)	140mA (with CF)	5V	700mW
GPS Receiver	100mA	5V	500mW
Total Power Consumption	-----------------------	-----------------------	1.794W

Table 5 - Maximum System Power Usage

Now that the power source has been chosen and the total system power has been calculated, the topology of the power distribution can now be chosen. The power source is one of the key factors when determining which method of distribution to choose since a power supply can have multiple output rails available whereas a battery would need DC/DC converters to boost, raise the output voltage when compared to the input voltage, or buck, lower the output voltage when compared to the input voltage, the power source voltage. The total amount of power distributed in a system was another key factor considered when we determined the topology of power distribution since the total power emphasizes the need for a higher efficiency power distribution design.

A couple of different methods of power distribution exist. The point-of-load topology utilizes one higher voltage rail (usually 12 volts and requires the placement of DC/DC regulators near components with larger loads in order to prevent pulling down a rail below a critical voltage due to the sudden current draw. Our distribution scheme uses the concept of the point-of-load topology but uses it more for simplicity than necessity. The FlashCore-B(FB) board comes with an AC power supply with a 12V output. At first, we decided to keep the current regulator on the FlashCore-B(FB) board and just boost to 12V to power the CPU board to make testing easier. However, after researching various boost converters to fit our design and comparing them to the buck-boost converters available, the boost converters provided application notes specifically for our application. Even with designing in the boost circuit before the 5V regulator, we still achieved at least 70% efficiency and were able to run the system for a minimum of 4 hours. We currently use the LT1935 1.2MHz Boost DC/DC Converter to boost from between 2.5V and 12V to 12V and then regulate the 12V back down to 5V for the FlashCore-B(FB) and another for the LCD display, GPS receiver, and debouncing circuitry.

Instead of using the previous boost then point-of-load buck topology for power distribution, a more efficient method combining both buck and boost together into the buck-boost topology would increase

efficiency by at least 10% since each stage is 90% efficient at maximum efficiency. However, designing a new buck-boost topology would take a great deal of research and work to implement due to the buck-boost converters not being made for this particular application currently. We would have had to completely design and implement the circuit to handle the full voltage range possible for 4 AA batteries (2.5V-6.5V).

Another important part of choosing a power source is the type of battery used. Table 4 compares the size of the battery such as AA and 9V which chooses the nominal output voltage, power capacity, and physical size and weight of the battery. The type classifies the chemical mixture used to create the voltage including non-rechargeable Alkaline and rechargeable Nickel Cadmium (NiCad), Nickel Metal Hydride (NiMH), or Lithium Ion (Li-Ion).

NiCad batteries have been the mainstay of rechargeable batteries due to being the original rechargeable batteries. Even though NiCads are rechargeable for possibly up to 1000 cycles, NiMH and others are replacing them due to many faults including being toxic (cadmium) and having far less energy density per unit volume than NiMH. For example, NiCad size AA cells have a nominal voltage of 1.2V and a capacity of 700mAh. However, AA size NiMH batteries have a 1.2V nominal voltage and over 2000mAh capacity. The 2000mAh capacity means when run at 2000mA, the battery will last an hour. When buying new rechargeable batteries, NiMH will increase the runtime of the system, provide nontoxic rechargeable power, and save money from buying batteries over and over again. The main reason for using AA size batteries as opposed to the 9V, besides the total power, is the ability to use the AA batteries in many other devices including a digital camera, palm pilot, etc. to make the purchase more worthwhile.

5.2.5 Module Interfacing

Now that each module had been tested on its own, we then had to test if the modules could work with each other. The first test conducted was to see if we could control the LCD with the FlashCore-B(FB). We decided to do this first because we felt it was harder to test if we could interface the FlashCore-B(FB) with the GPS receiver. We felt that it was harder because it entailed using the CompactFlash unit at the same time, to actually see if the two units were communicating properly as opposed to displaying immediate results on the LCD. In Appendix E, the connection between the LCD and FlashCore-B(FB) is shown. Once connected properly, a test program had to be written which initialized the LCD and wrote characters to the screen. This task was extremely tedious because we had to set each pin on the FlashCore-B(FB) manually in the program, which meant a maximum eight lines of code for each of the data lines, two additional lines for the "enable" line, and an additional line for the "register select" line to switch between instructions and data. It was not until later on where we figured out a function which could take in a string, break it up, and send the correct TTL value to the correct pin.

In order to test the Garmin GPS receiver, we needed to read each incoming character, assemble the GPS string one character at a time, and print the strings to the CompactFlash card to prove the GPS receiver could communicate to the FlashCore-B(FB). Figure 34 illustrates the steps needed to prove the Garmin GPS receiver and FlashCore-B(FB) could communicate together.
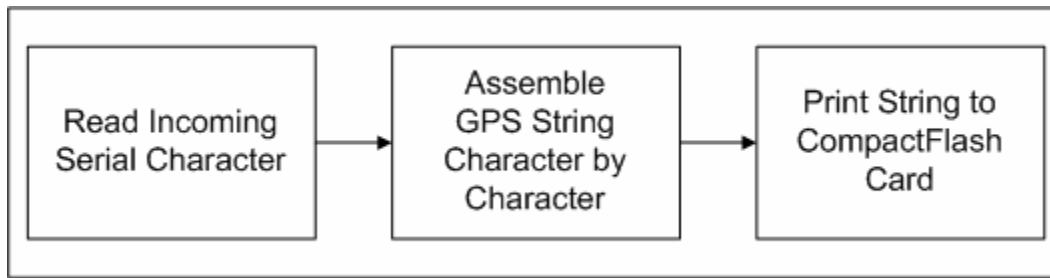
Figure 34 – Garmin GPS Receiver Test Block Diagram

5.2.6 System Board

After we selected all of the components and verified that each module could be interfaced to the other modules, we drafted some drawings seen in Appendix F. The drawings gave us detailed measurements concerning modifications to the enclosure and the orientation of all the components to ensure all components fit on one board and all signals had enough room to get to other modules since we decided to use a two-layer board. After the drawings were completed, we then moved onto designing our PCB. The first thing we did in our design was lay out where all the modules were supposed to be placed and from there we made the necessary connections. Seen in Figure 35 is the fully implemented system module. Appendix G contains our actual PCB layout, including the silkscreen, top layer, and bottom layer as separate illustrations.
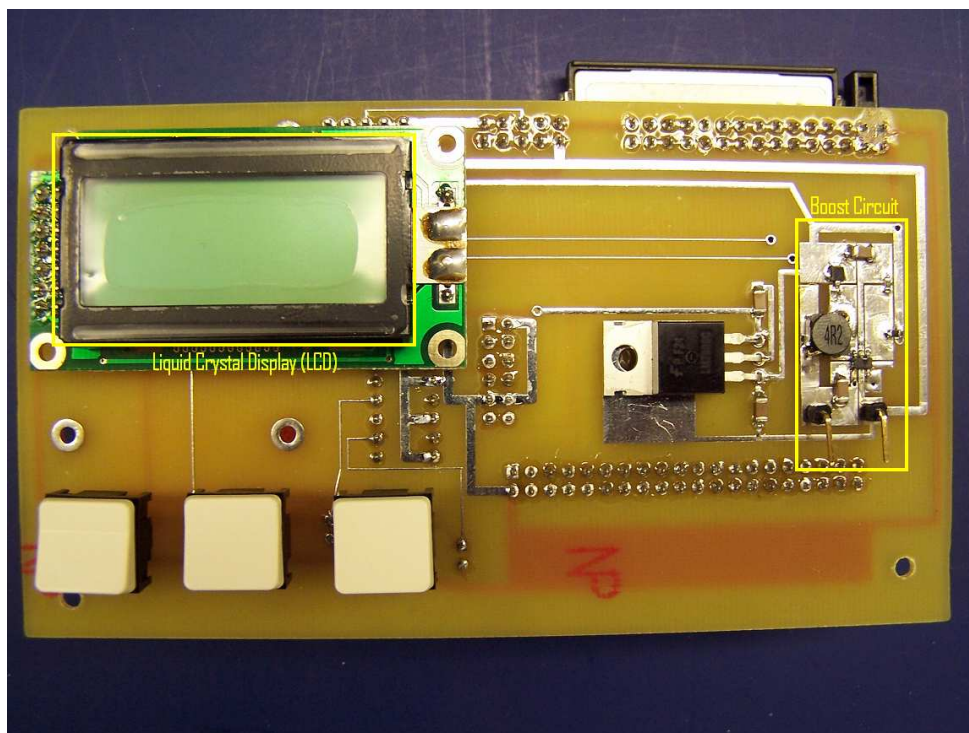
Figure 35 - System Module

5.3 Software Implementation

The implementation of the software was a crucial part in the completion of this project. The FlashCoreB(FB) includes the Paradigm C/C++ software development kit which we used to program the microprocessor and to debug the code. Inside the code, after the initialization of the microprocessor board and LCD is the main loop. Inside the main loop is where all the GPS data is processed and the control of the LCD occurs. Also within the main loop is another subsection, in the form of a loop, which performs the L/D calculations, described later. Figure 36 is the software flow chart which gives the details on how our program operates.

$$\Delta ALT = (ALT_0 - ALT_N) \qquad \text{(Eqn. 1)}$$

$$V_a = \frac{1}{10} \sum_0^t V_t, t = 10 \qquad \text{(Eqn. 2)}$$

$$V_N = V_0 \times 30.86667 \qquad \text{(Eqn. 3)}$$

$$L\!\!\!/_D = \frac{V_N}{2\Delta ALT} \qquad \text{(Eqn. 4)}$$
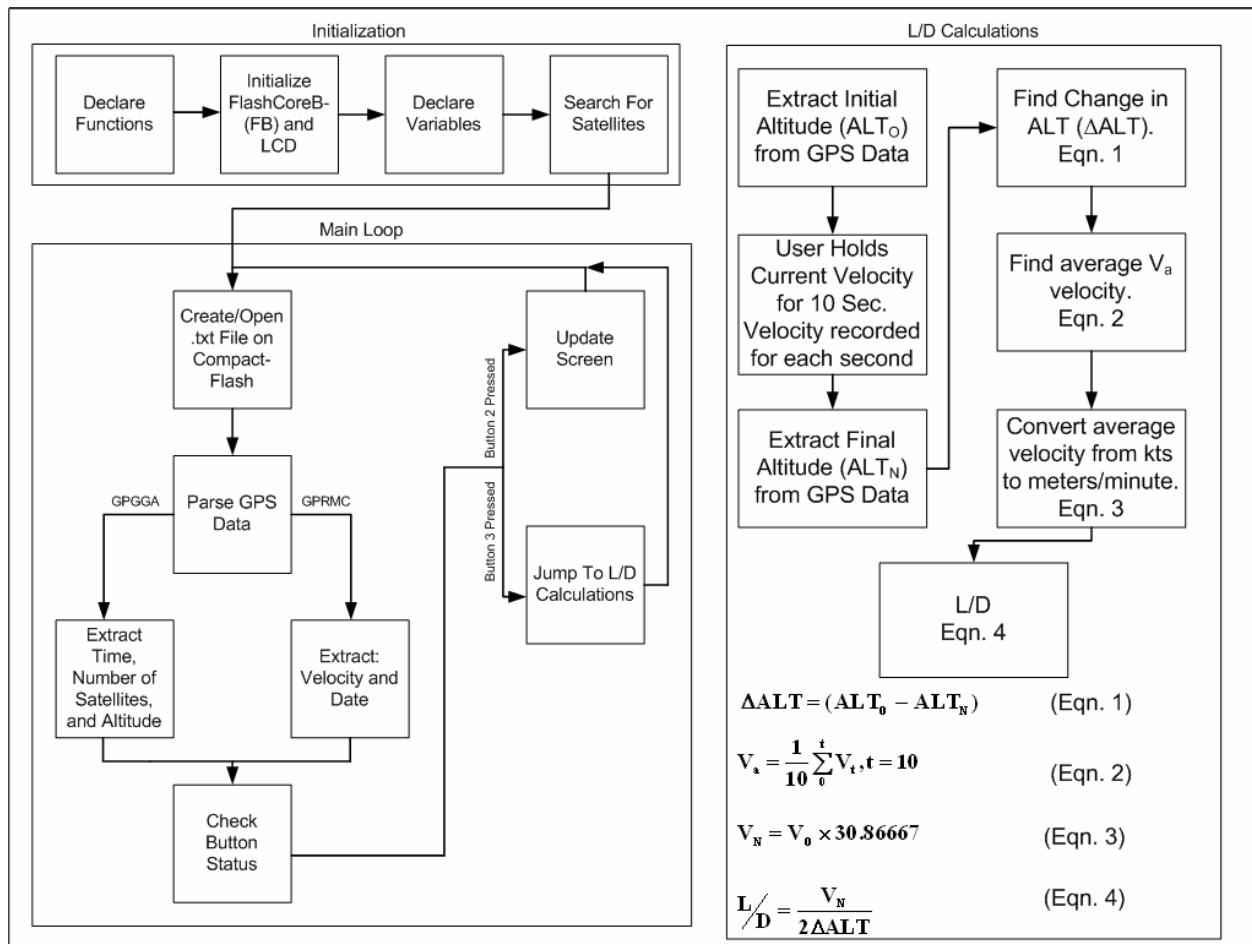
Figure 36 - Detailed Software Flow

5.3.1 GPS serial in

The GPS data is transmitted one character at a time in sentences through the serial port located on the FlashCoreB(FB) microprocessor board. From there, the software makes a call to a function, "GetString()", which reads in the GPS strings and stores them into an array, "gpsStr". Once the sentence is read, it is important to find out which GPS sentence it is, so that we can parse for any necessary data, relevant to that string. Because the GPS sentence structure is standardized, it is easy to parse for the

sentence type first. Each GPS sentence begins with a "$" and is then followed by five letters which represent its type. Since the GPS string is stored in an array buffer, the software directly copies the characters in elements gpsStr[1] through gpsStr[5] into a smaller temporary array. This temporary array is then compared to other strings using a series of "if" statements to find out which string has just been read in. Once the software knows which string it is, calls to the proper "Find" functions are made to extract the needed data. There are a total of three "Find" functions which find the number of satellites, velocity, and altitude. The number of satellites is always a two digit number, velocity is a four digit number in the form of "###.#", both with leading zeros transmitted, and altitude can be anywhere in the range of -9999.9m to 9999.9m, without leading zeros transmitted. The "Find" functions all work in a similar fashion as to extracting the string type from the sentence. Because the GPS strings are all transmitted in the NMEA standard, the locations of the information is always the same. The following is an example of the GPS string type "GPGGA", where the number of satellites and altitude are bolded:

*$GPGGA,214844,4214.3599,N,07145.2118,W,1,**06**,1.7,**127.5**,M,-33.3,M,,*78*

If this string were to be stored in an array, each character would be a separate index, where the first index is 0. From this string, we see that the number of satellites starts at index 41 and ends at index 42. It is also seen that the altitude begins at index 48, but where it ends is tricky because leading zero's are not transmitted. To solve this problem, after each number was copied into the new string array, the program would compare the character in the next location to a comma. If it was not a comma, the function would continue to copy the entire value, with the decimal included.

The difference between the "find" functions and the "GetString" function is that the find functions return numerical data types integer and double, as opposed to a string. To do this, the functions first extract the data as strings and store them into a temporary array and then use the C++ function "atof" which converts a string to a floating point number. The function that finds the number of satellites works in a slightly different manner. The software subtracts 30 from the characters ASCII value which converts the character into a numerical value. The first number is then multiplied by 10, so that it is placed in the ten's digit location, and added to the second number.

For this project, there are two GPS sentences that contain the necessary data for the approximation of L/D. The first sentence is "$GPGGA", Global Positioning System Fix Data(NMEA Sentence), which contains the number of satellites, time, and altitude. The second sentence we needed is "GPRMC", Recommended Minimum Specific GPS/TRANSIT Data(NMEA Sentence), which contains the velocity and the date. Unlike the other GPS data, the time and the date are parsed for in the exact same method as the GPS sentence type, except the date is handled differently. The date is transmitted in the format "DDMMYY", so when extracting, it is stored in the "MM/DD/YY" format. One key portion of the program is a located at a label designated "NES" which stands for "Not Enough Satellites". After all the necessary data from the "GPGGA" sentence has been parsed, the software checks to see if there are at least four satellites. If there are not enough satellites, the software will jump to the "NES" label, which forces the user to wait until there is enough.

5.3.2 LCD control

Controlling the LCD required the most functions. Because the LCD we are using for the project is a parallel device, the most important function in controlling the LCD would have to be the "set_pio()" function, which takes in an integer as an input. The input is then copied over to a temporary local variable and all the bits are masked except the least significant bit. This is then divided by a 01H to extract its actual value and sent to the correct pin on the FlashCore-B(FB), corresponding pins between the LCD and FlashCoreB(FB) can be found in the schematic. The input value is then recopied into the temporary local value and then all bits are masked except the second least significant bit. This is then divided by a 02H to extract its actual value and also sent to the corresponding pin on the FlashCore-B(FB). This process is repeated until all eight pins have been set, dividing by 01H, 02H, 04H, 08H, 10H, 20H, 40H, and 80H respectively. This function simplifies the code because the decimal or hexadecimal value of the internal LCD 8-bit instructions can be passed into the set_pio() function instead of having the programmer manually set each pin every time. This is very important for the initialization of the LCD because there are seven instructions that need to be sent to the LCD and also eases the creation of the other functions created for the control of the LCD. The other functions that really on this function are the clr_scr(), char_pos(), and str_wr() which clear the screen, set the cursor position, and writes out the inputted string respectively. For the clr_scr() and the char_pos() functions, the set_pio() function is passed the appropriate instruction code, but the char_pos() is a little more difficult. The instruction for setting the cursor position is "1XXXXXXX", where the X's in the last seven bits correspond to the DDRAM address of the desired location. For the top row of the LCD the addresses are 00H-07H and for the bottom row the addresses are 40H-47H. The char_pos() function takes in either a 1 or 2, for the row, and a number between 0 through 7 for the desired column. If row 1 is chosen, the column value is passed directly to the set_pio(), but if row 2 was chosen, then 40 is added to the column value before being sent to set_pio(). After the pins have been set by set_pio(), the function manually overrides the most significant bit to account for the "1" in the instruction code. The str_wr() function is a while loop which repeats the set_pio() function eight times or until the end of the string. The string passed into the function is an array and within the function a local variable "r" is initialized to 0 for the beginning of the string. The function then passes string[r] into set_pio() for processing and then increments "r". The function then checks the next character to make sure the end of the string has not been reached. If it is the end, the function sets "r" to a value which will cause the conditional of the while loop to return false, exiting out of the function.

Another important aspect of the LCD control, was the ability to scroll through different screens. The different screens that a user is able to scroll through are number of satellites, velocity, altitude, the last L/D calculated, the average of all the L/D's calculated, and a date/time screen. To make this all possible, two global integer variables initialized to 0. One variable controls which screen to display, "disp_type", and the other variable controls whether or not the screen needs to be cleared, "clear". The software goes through a series of "if" statements which checks which screen to be displayed. Once the screen type is found, there is another "if" statement. If it is the first time displaying the chosen screen, the "if" statement will be true, and the old screen will be cleared to remove unnecessary data. The "clear variable is then incremented so that the screen will not clear until the next screen is displayed. To change screens, the software checks to see if the user has pressed button 2. If the button is pressed, the "disp_type" variable will be incremented, causing the current true "if" statement to become false and the next statement to be

true. Once on the last screen, if the button is pressed again, both variables are reset back to 0 which allows the user to loop around through the screens. The only screen that requires a little more additional coding is the last screen which displays the date on the top line and the time on the bottom line in standard format (HH:MM:SS). The date is displayed as is, but the time needs to be altered. Upon extraction of the time, it is stored as a string in the format "HHMMSS" and it is the official coordinated universal time (UTC), which is four hours ahead of the local time. Although it is four hours ahead, we simply cannot merely subtract four hours, because of the cases when the time is between midnight (00:00:00) and 04:59:59. If four hours was subtracted, the result would be either negative or zero hours. The program first checks to see if it is 5:00 AM or later and if it is four hours are subtracted. If it is earlier than 5:00 AM then the program will add eight hours which converts UTC time to Eastern Time. After these steps, the program then checks to see if it is 1:00 PM or later, and if it is the program will subtract 12 hours to convert it from military time to standard time. This gives us the appropriate time conversion for the local time zone, but in the even the device is taken elsewhere, the program will need to be altered. Since leading zeros are omitted from numerical data types, the program will add an additional 10 hours as a place holder, because if it is earlier than 10 o'clock the value stored will be in the format "HMMSS". The time is then converted back to a string, the extra 10 hours is subtracted by subtracting 1 from the first position of the array, and the colons are added in between. Figure 37 is a copy of the actual screen flow.
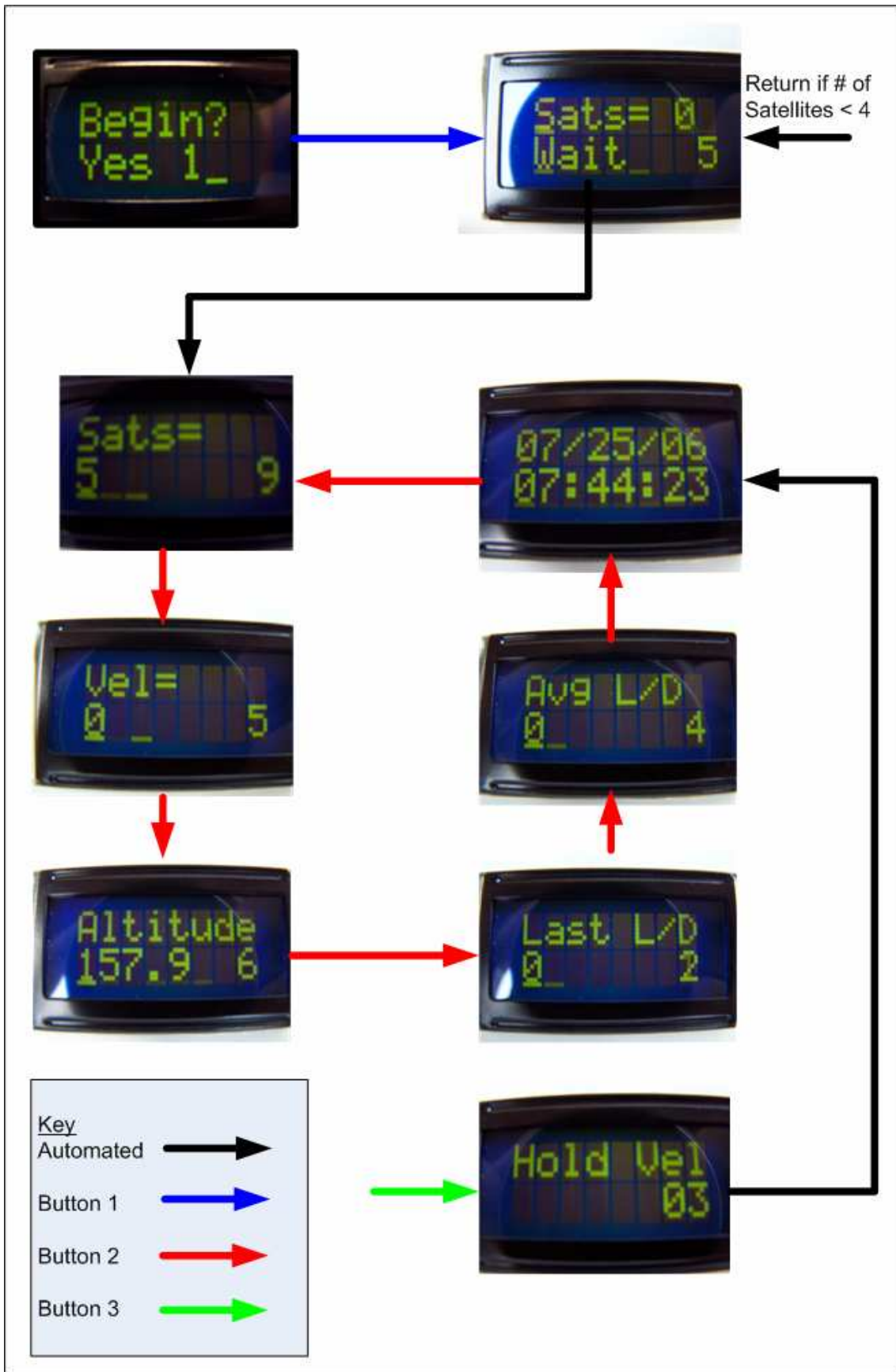
Figure 37 - LCD Screen Flow

5.3.3 CompactFlash Transfer

One main goal of this project was the ability to store data to removable memory. The FlashCore-B(FB) microprocessor that we chose for this project comes with a CompactFlash drive and all the necessary functions to use it. At the beginning of the main loop, the program checks to see if there is a file, of the given name, and makes sure the file is not corrupt. If the file does not exist, then the file is created and program continues, checking to make sure the file is not corrupt upon reiteration of the loop and if the file is corrupt, the program loses the ability to write to the file until the file is deleted so that a new file may be created. The only time in the program that the file is written to, is after the L/D calculations have been completed. The data that is stored are the date, L/D, the average velocity, initial altitude, and final altitude using the function fs_fprintf() which sends the data to a buffer and the function fs_fflush which sends the data from the buffer to the file. The data is now ready to be exported to another source for analysis.

5.3.4 Calculations (L/D)

The calculation of L/D starts when button 3 is pressed. When button 3 is pressed, the program will enter a "while" loop, located in the main loop. The conditional for this "while" loop is a Boolean variable which is set to false up until the moment the button was pressed. This means when the button is pressed, the variable is set to true, allowing the program to enter the L/D calculation loop. Seen in Figure 38 is the flow chart followed for the calculations. The user is asked to hold their velocity for 10 seconds, during which the velocity for each second is stored into an array. At the beginning and end of the 10 seconds, an altitude reading is stored into "alt_init" and "alt_final" respectively. At the end of the data extraction, the Boolean variable is then reset to false so that the L/D "while" loop is executed only once. The final altitude is subtracted from the initial altitude and the change in altitude is then multiplied by 6 to account for a minute duration of flying. The program then takes an average of the velocities stored from the 10 seconds and is then converted to meters/minute by multiplying the average velocity by 30.8667. The average velocity is then divided by the change in altitude, which yields the L/D result. The L/D is then added to another variable which stores the total of all the L/D's calculated. This allows for an average L/D of the day to be taken. The key factor in any calculation is to be consistent with units. In our case, we converted anything regarding time to minutes and anything regarding length to meters.
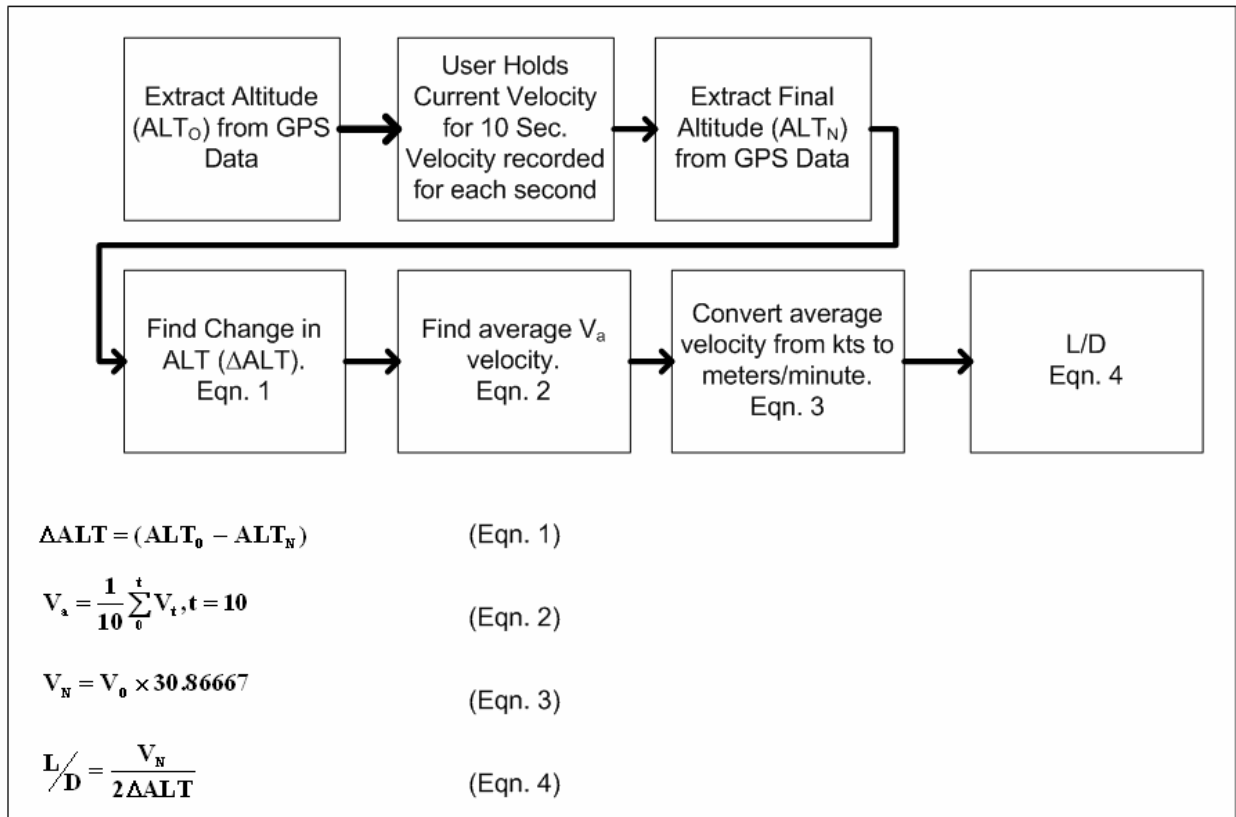
Figure 38 - L/D Calculation

The equations shown in the figure:

$$\Delta ALT = (ALT_0 - ALT_N) \qquad \text{(Eqn. 1)}$$

$$V_a = \frac{1}{10}\sum_0^t V_t, t = 10 \qquad \text{(Eqn. 2)}$$

$$V_N = V_0 \times 30.86667 \qquad \text{(Eqn. 3)}$$

$$L/D = \frac{V_N}{2\Delta ALT} \qquad \text{(Eqn. 4)}$$

5.4 System Test

One important step necessary in the development of our system was testing the accuracy of our GPS receiver, Garmin's 15H. According to the receiver's manual, it has an accuracy of up 3-5 meters with the Differential GPS activated. In our system, we have this feature set to automatic to receive DGPS information whenever it is available. In order to test its accuracy, we first had to find some known locations and their altitudes. To do this, we went to the Worcester Public Library and asked for topographic maps. As seen in Figure 39 and Table 6, the locations we found off of the topographic maps were Worcester Polytechnic Institute, Grafton Plaza, and Heywood Street. Since topographic maps do not give exact values, the values listed in the chart are rough readings based on the maps scales and lines of contour. The librarian also directed us to the library's homepage which listed a few additional locations. The ones we decided to use were City Hall and Union Station. Figure 39 is a graphical representation of the data found in Table 6. Once this information was found, we then had to take the GPS receiver and our system to test these values. The values read from our test were then compared to the information we found at the library. The actual values achieved from our system were relatively close to the indicated accuracy of 3-5 meters as advertised by Garmin. Any discrepancies found in the values may have been the result of various reasons. The first reason that comes to mind is the fact that the values read from the topographic maps were estimates as opposed to accurate values. Other reasons could be errors in the GPS timing as described earlier. When the skies are not clear or the area is surrounded by high buildings and

other structures, such as the case of City Hall, there is an added delay in receiving the GPS information from the satellites.

Location	Topography Elevation (Approximate)	GPS Elevation	Difference
WPI	167 m	169.9 m	2.9 m
Grafton Plaza	150 m	154 m	4 m
Heywood St	182.9 m	191 m	8.1 m
City Hall	146.6 m	140 m	6.6 m
Union Station	144 m	138 m	6 m

Table 6 - Elevation Comparisons
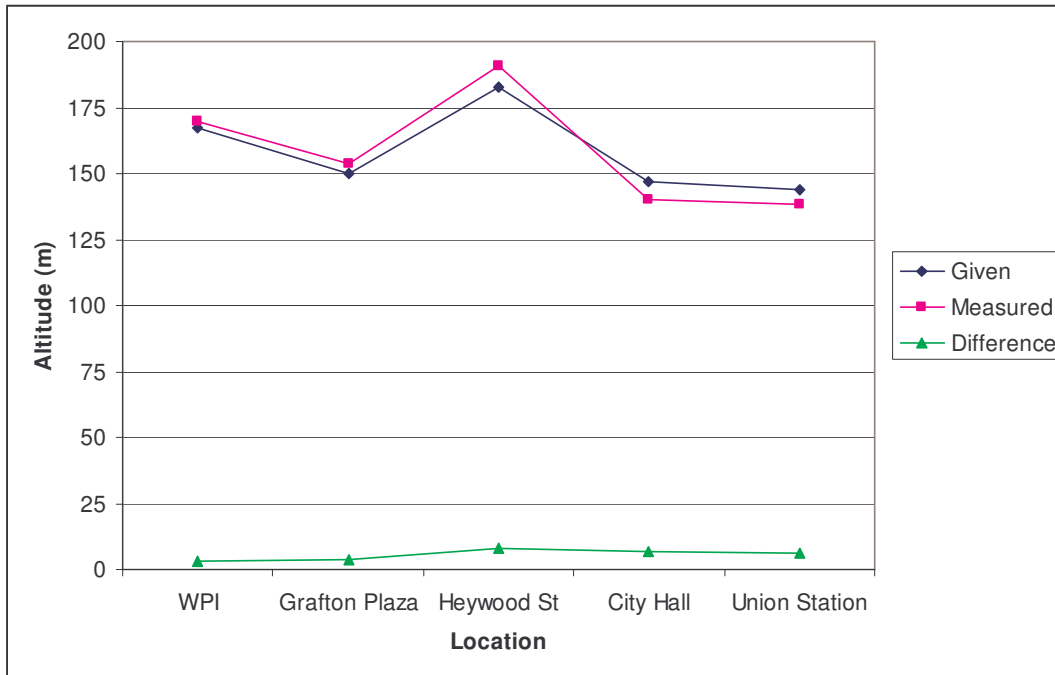
Figure 39 - Elevation Comparisons

After testing the altitude accuracy of the GPS receiver, we then moved onto testing the accuracy of the velocity. In order to test the velocity, we drove around with the unit in our cars. We drove along long roads with minimal stops at constant speeds, such as 30 mph and 45 mph, depending on the local speed limit. Knowing that we were traveling at the speed limit, we took note of the velocity readings given by our system. We then converted our speeds from miles per hour to knots and compared them. Table 7 is a sample of our comparison.
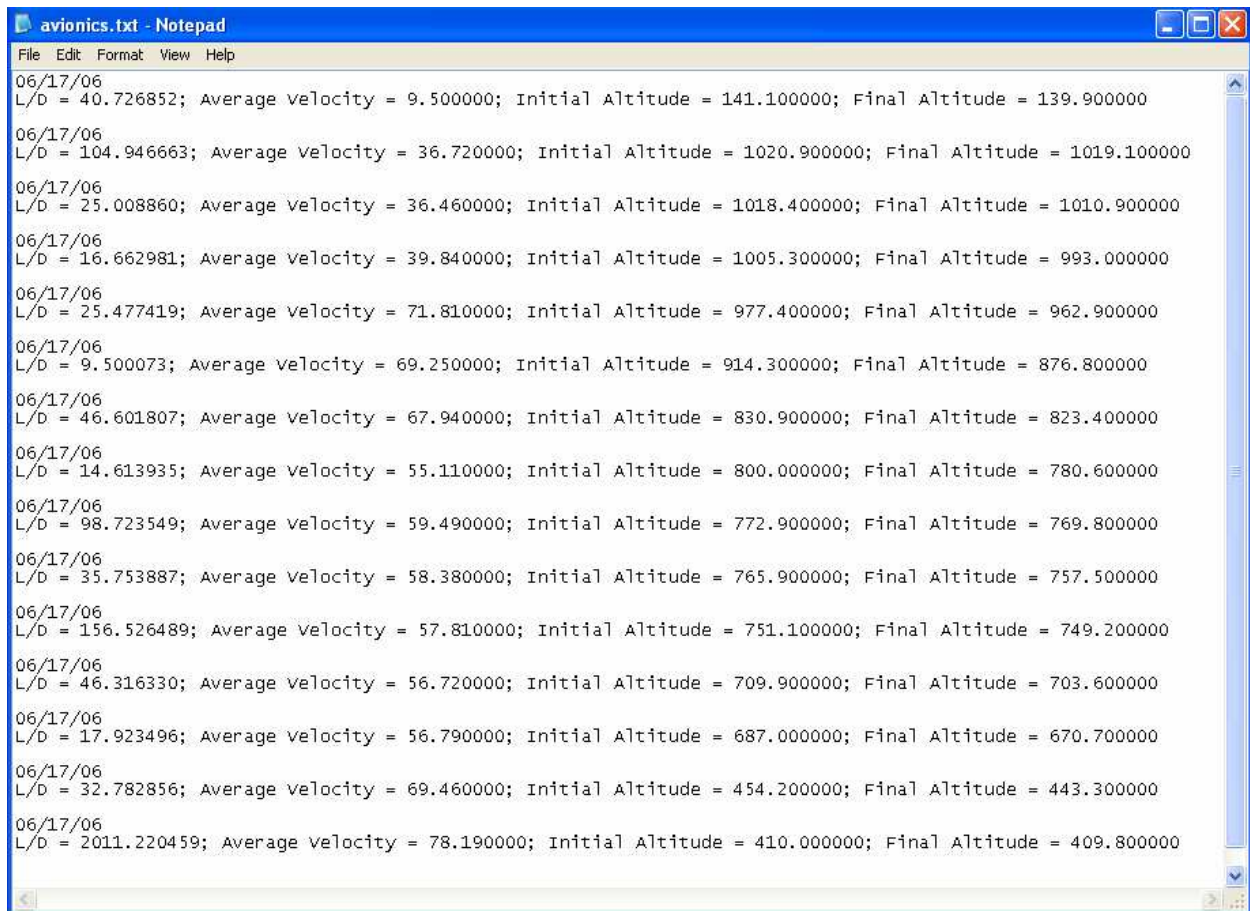
Actual Velocity (car - mph)	Expected Velocity (kts)	GPS Velocity (kts)	Percent Error (%)
30	24.5	26.1	6.1
45	37.2	39.1	4.9
65	54.9	56.5	2.8

Table 7 - Velocity Comparison

5.4.1 Routine Flight Data Analysis

Our routine flight took place in an L23 Super Blanik sailplane with a maximum specified L/D of 28:1 at 49 knots with 2 pilots[32]. Figure 40 is a screenshot of the actual file recorded during our flight, which we then adapted into Table 8 for analysis seen below. Comparing the calculated effective L/D data measurements to the maximum specified L/D of 28, we find a large discrepancy since our effective L/D measurements were recorded as much higher than the maximum specified L/D. Table 8 and Table 9, as well as Figure 41 and Figure 42, only differ with three outliers omitted from the first set of data. When the three points farthest from the consistent grouping are omitted, an average flight effective L/D of 20.82 is attained as compared to 48.53 initially.

The main source of error during this flight was the weather conditions. Our device is design to record accurate effective L/D during calm air. Since the flight took place after 10am on a hot, sunny day, the air had plenty of time to become unstable due to the heating effects on the ground. When flying, 300 ft/min ascending columns of air were not uncommon. Furthermore, descending columns of air of around the same rate sometimes immediately followed the ascending columns of air. Regardless, when the three least consistent points are omitted, the average effective L/D seems reasonable for the velocities measured.

Figure 40 - Recorded Flight Data

---

[32] http://www.nwi.net/~blanikam/ba/prod01.htm

L/D	Average Velocity	Initial Altitude	Final Altitude
104.946663	**36.72**	**1020.9**	**1019.1**
25.008860	36.46	1018.4	1010.9
16.662981	39.84	1005.3	993.0
25.477419	71.81	977.4	962.9
9.500073	69.25	914.3	876.8
46.601807	67.94	830.9	823.4
14.613935	55.11	800.0	780.6
98.723549	**59.49**	**772.9**	**769.8**
35.753887	58.38	765.9	757.5
156.526489	**57.81**	**751.1**	**749.2**
46.316330	56.72	709.9	703.6
17.923496	56.79	687.0	670.7
32.782856	69.46	454.2	443.3

Average L/D   48.526027

Table 8 - Original Flight Data
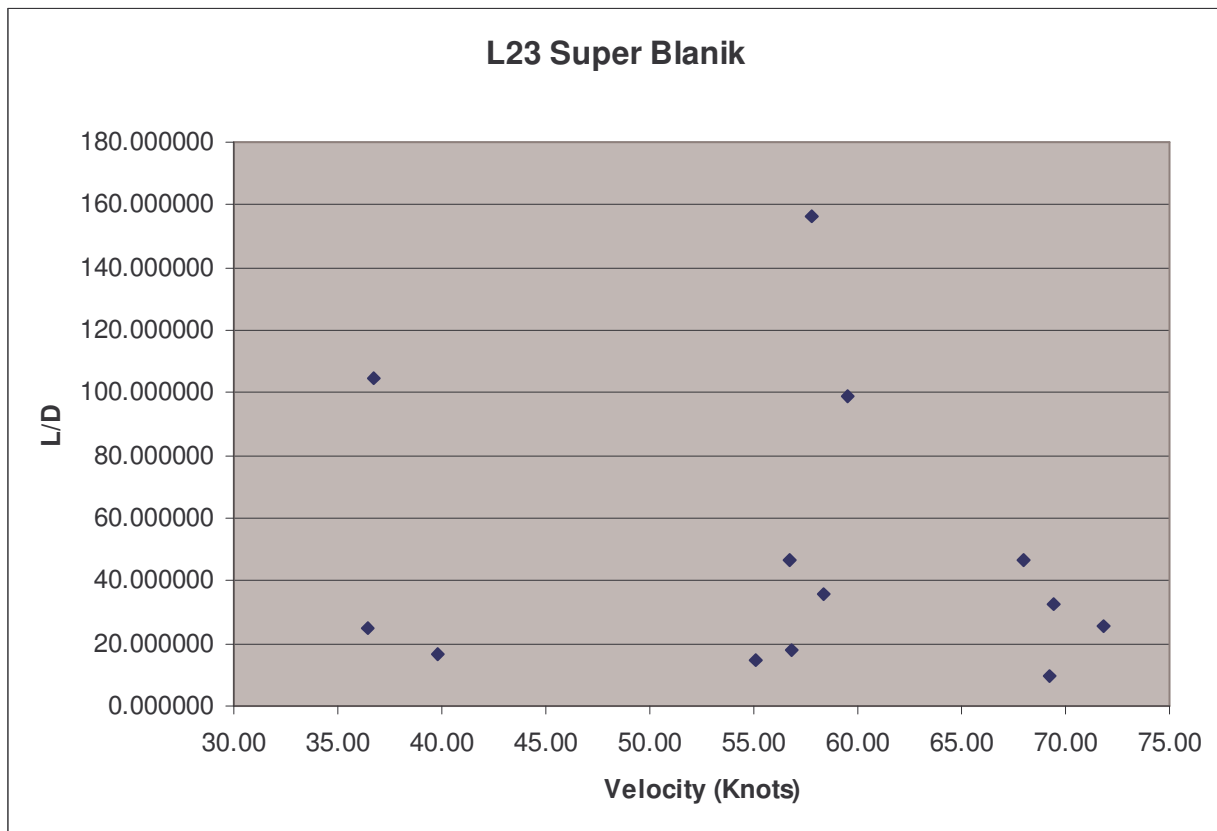
Figure 41 - Original Flight Data

L/D	Average Velocity	Initial Altitude	Final Altitude
25.008860	36.46	1018.4	1010.9
16.662981	39.84	1005.3	993.0
25.477419	71.81	977.4	962.9
9.500073	69.25	914.3	876.8
46.601807	67.94	830.9	823.4
14.613935	55.11	800.0	780.6
35.753887	58.38	765.9	757.5
46.316330	56.72	709.9	703.6
17.923496	56.79	687.0	670.7
32.782856	69.46	454.2	443.3

Average L/D   20.818588

Table 9 - Flight Data with Three Outliers Omitted
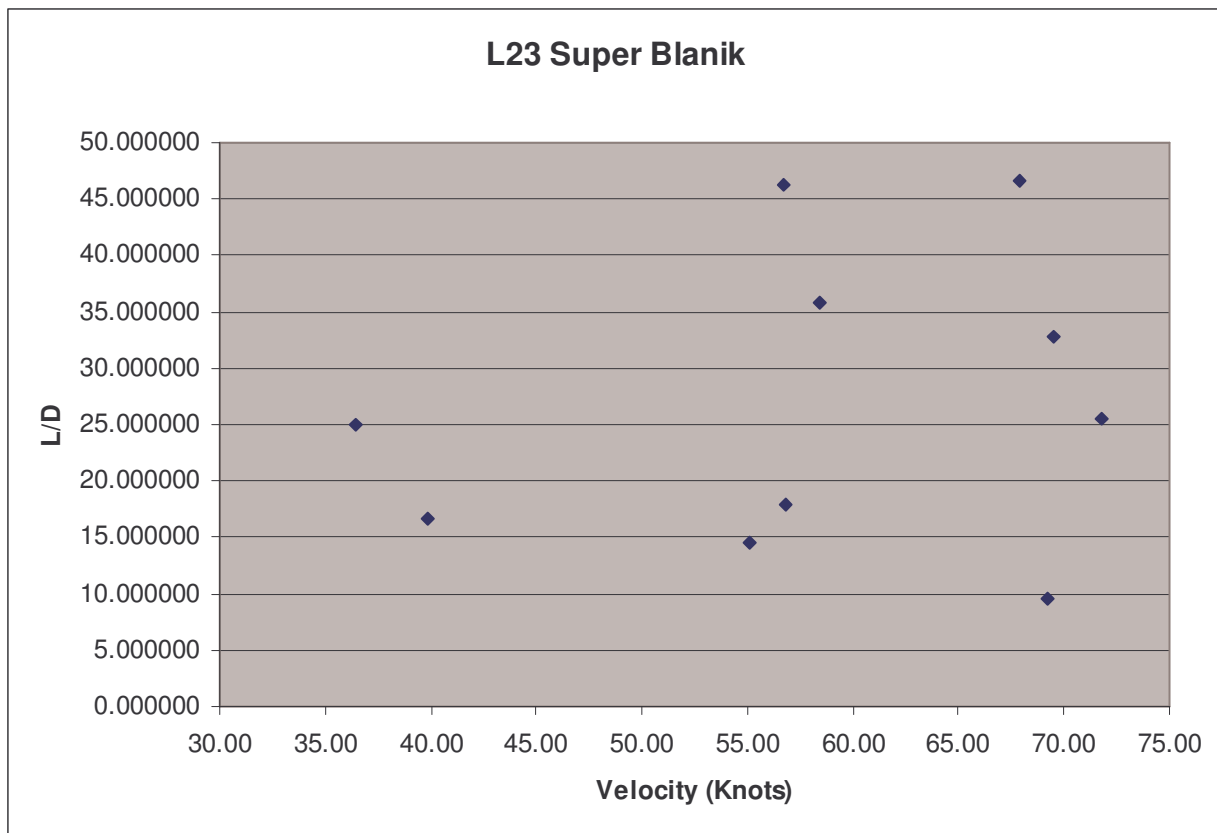
Figure 42 - Flight Data with Three Outliers Omitted

5.5 Summary

This chapter covered the most important parts of the Flight Data System project including hardware implementation, software implementation, and the results from our road test simulations. Most of the hardware implementation was simplified due to having a few parts lying around that included all of the necessary hardware features needed to include in the project.

6   Summary and Conclusions

6.1 Introduction

The project works and provides data reliable enough to give accurate readings when in calm air only. Flying after the sun has been able to heat the surface for more than a couple of minutes, the unstable air creates turbulence when flying which yields excessive variations in the data measurements due to the unstable atmosphere.

The project works and reports readings within the specified accuracy range of the Garmin GPS receiver when compared to the one-foot accuracy topographic map data. The enclosure has physical dimensions small enough to fit in the sailplane space available for this instrument. The CompactFlash Card allows for easy portability of pertinent data to the computer for data analysis.

Some expected future work includes designing a true buck-boost power distribution topology in place of the current boost to 12V then a point-of-load buck converter placed near high power loads. A more accurate L/D calculation should be researched and verified.

6.2 Completed Work

The project works and provides data reliable enough to give accurate readings when in calm air. Flying after the sun has been able to heat the surface for more than a couple of minutes, the unstable air creates turbulence when flying that makes readings jump due to the inconsistency.

6.3 Future Work

Future teams should research the possibility of including any physical factors of a sailplane to achieve a greater accuracy in the calculation of the L/D ratio. Currently, the GPS receiver reports data every second. The one Hertz data rate provides a major point of error since conditions change quite frequently in conditions where the air is even slightly unstable.

Future teams should implement a buck-boost power distribution topology to have an efficient version of the Flight Data System. The new topology would increase the efficiency from approximately 71% currently to 80-90% overall system efficiency.

The use of field-programmable gate arrays (FPGAs) should be researched to find out the feasibility of a successful design as opposed to using the FlashCore-B(FB) microprocessor board. We chose not to pursue a FPGA solution since the FlashCore-B(FB) was already in design for a concurrent project.

6.4 Summary

This chapter presented the work completed for this project and mentioned suggested work for future teams to help create a better project. Further trials are necessary for a proper conclusion since testing the system anywhere but in a sailplane defeats its intended purpose and does not account for the same weather factors experienced in the air as opposed to land.

7   Works Cited

What is GPS?. 1996-2006. Garmin. 8 Nov 2005 <http://www.garmin.com/aboutGPS/>

GPS Guide for Beginners. December 2000. Garmin. 8 Nov 2005
        <http://www.garmin.com/manuals/GPSGuideforBeginners_Manual.pdf>

Brain, Marshall and Tom Harris. How GPS Receivers Work. 1998 – 2006.
        How Stuff Works. 8 Nov 2005 < http://electronics.howstuffworks.com/gps.htm >

Ganssle, Jack G. A Guide To Debouncing. August 2004. 8 Nov 2005
        < http://www.ganssle.com/debouncing.pdf>

Burch, Jim D. Performance Airspeeds for the Soaring Challenged. 2000. 5 April 2006.
        <http://home.att.net/~jdburch/polar.htm>

Brandon, John. Knowing the Aircraft. 2004. 10 April 2006.
        <http://www.auf.asn.au/emergencies/aircraft.html>

Piggott, Derek. Gliding: A Handbook on Soaring Flight. Adam & Charles Black: London, 1977.

United States. Dept. of Transportation. Federal Aviation Administration. Glider Flying
Handbook. Washington: GPO, 2003.

Worcester Public Library. 25 July 2006.
http://www.worcpublib.org/resources/faqworcester.html#WORCESTER%20ALTITUDE/ELEV
ATION

United States Air Force, CAP National Technology Center: "CAPF 5G Online-Course". July 26,
2006
https://ntc.cap.af.mil/ops/DOT/school/NCPSC/GliderNCPSC/CAPF_5_glider/soaringtechniques.
htm

Soaring Society of America: "What is Soaring?". July 26, 2006
http://www.ssa.org/sport/whatissoaring3.asp

"Thermal Soaring". July 26, 2006
http://myweb.tiscali.co.uk/miskin/gliding/gliding/x_thermalling.htm

UEET NASA. "History of Flight". July 26, 2006
http://www.ueet.nasa.gov/StudentSite/historyofflight.html

Soaring Society of America. USA Soaring Team. "History of Gliding and Soaring". July 26, 2006
http://www.ssa.org/UsTeam/adobe%20pdf/pr%20pdf/BR%20Soaring%20History%20V5%2004.pdf

American Institute of Aeronautics and Astronautics. "Leonardo Di Vinci". 2006. July 26, 2006.
http://www.aiaa.org/content.cfm?pageid=425

Wings and Wheels. Fred J Looft. "New Glider Deliveries – 304C 'FL'". July 26, 2006
http://www.wingsandwheels.com/Glasflugel304C.htm

Glasflugel 304C Handbook. July 26, 2006
http://alvsbyflygklubb.se/dokument/flyghandbok304c.pdf

Wikipedia. "Glaser-Dirks DG-300". July 26, 2006
http://en.wikipedia.org/wiki/Glaser-Dirks_DG-300

Wikipedia. "Rolladen-Schneider LS4". July 26, 2006
http://en.wikipedia.org/wiki/Rolladen-Schneider_LS4

Wikipedia. "Politechniki Warszawskiej PW-5". July 26, 2006
http://en.wikipedia.org/wiki/Politechniki_Warszawskiej_PW-5

Wikipedia. "Schempp-Hirth Duo Discus". July 26, 2006
http://en.wikipedia.org/wiki/Schempp-Hirth_Duo_Discus

Geelong Gliding Club. "Rolladen-Schneider LS4a". July 26, 2006
http://www.gliding-in-melbourne.org/ls4a.htm

Greater Boston Soaring Club. July 26, 2006
http://soargbsc.com/fleet.php

Associated Glider Clubs of Southern California. "SGS 2-33". July 26, 2006
http://www.agcsc.org/sgs_2_33_info.html

Blanik America. "LET L23 Super Blanik". July 26, 2006
http://www.nwi.net/~blanikam/ba/prod01.htm

Appendix A: Executive Summary

Introduction

This appendix presents the executive summary of our report to serve as a brief overview and summary of the Flight Data System project.

Project Number: FJL-AVON

Flight Data System

A Major Qualifying Project Report: submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

Tri Lai
trilai@wpi.edu   _____

Russell Pead
rustyp@wpi.edu   _____

Date: July 26, 2006

Fred Looft
fjlooft@ece.wpi.edu ————————————————

Abstract

The goal of this capstone design project was to design a fully portable sailplane Lift/Drag (L/D) calculator. The system utilizes NMEA format GPS data strings for software data analysis executed by software. Calculated effective L/D results are stored on a removable data storage media (e.g. CompactFlash card) for later data analysis.

Worcester Polytechnic Institute
100 Institute Road Worcester, MA 01609

Introduction

One of the least known sports is high performance gliding, commonly referred to as soaring. Sailplanes make use of thermals, rising columns of air, and other forms of lift to achieve high altitudes. A pilot can fly for hours depending on their skill level, granted conditions are ideal. Some of these conditions include weather, the number of thermals in the area, and the strength of the area. These will be explained in more detail later.

In order to successfully fly long distances and increase flight time, two critical characteristics of a sailplane should be optimized. The two most critical performance characteristics of a sailplane are the Lift/Drag (L/D) ratio and Minimum Sink Rate. Both performance characteristics rely heavily on the physical characteristics of the plane such as weight, wing shape, and the actual aerodynamic design of the plane. Because the pilot has no control over physical aspects of the plane, the pilot must control different flight variables, such as velocity and angle of attack.

The purpose of the Flight Data System project was to design, implement, and test a fully independent portable instrument capable of calculating the effective L/D ratio of a sailplane using GPS data and storing data onto removable media for analysis.

This chapter introduced a problem, which sailplane pilots frequently find themselves confronted with. In order to help solve this problem, we have developed a problem statement, which in turn will lead to the development of a portable Flight Data System that can calculate the effective L/D of any given sailplane. Although there are currently instruments on the market capable of calculating the effective L/D ratio, we hope to develop a system that has the capability to store data onto removable media, is lightweight, and portable.

Background

Effective L/D, commonly referred to as max glide ratio, is the unit of distance traveled divided by the unit loss in altitude. Since effective L/D is a ratio, any distance unit can be used without the necessity for using a particular measuring system. The effective L/D is as significant, if not more, than the manufacturer specified L/D since the glider will not always achieve the specified L/D. The effective L/D approximation used for this project represents the effect of all factors from the pilot's flying style to various temperatures experienced at different altitudes, and the movement of the air through which the glider is flying, and the movement of the air through which the glider is flying.

Global Positioning System (GPS) data is transferred using the National Marine Electronic Association (NMEA) 0183 standard for GPS strings. The NMEA 0183 standard provides defined strings each containing a specific set of data. The Global Positioning System network consists of 24 active satellites constantly orbiting the Earth at an altitude of 11000 nautical miles. In order for GPS measurements to be accurate, at least 4 satellites must be detectable by the receiver, which is ensured by the orbit configuration of the 24 satellites.

The Global Positioning System uses a process called trilateration to determine the location of a receiver. With the bare minimum of 3 satellites, the location given by the data is accurate within 100 meters. With

4 or more satellites, accuracy is significantly improved to less than 10 meters, depending on the receiver. In essence, each satellite connection generates imaginary spheres with a radius equal to the distance between the satellite itself and the GPS receiver. The overlapping areas between the spheres are equivalent to the possibilities of where the GPS receiver could be located. With one satellite connection, the receiver can be anywhere on the outer edge of the sphere. Two satellites will yield a possibility of points along the radius of a circle. Further, with three satellite connections, there are only two possibilities of where the GPS receiver can be located. Of the two locations, one on Earth and the other is in space. By using the Earth as a fourth sphere, the possibility of being located in space is eliminated, but a fourth satellite connection is needed to attain accuracy within 10 meters.

Problem Statement

The goal of our project was to design, implement and test a portable sailplane flight performance instrument. The purpose of this chapter is to specifically state the project goal, identify the objectives necessary to achieve the goal, and list the tasks necessary to accomplish the objectives and the overall goal.

Measurable Objectives

After researching the characteristics and capabilities of racing sailplanes, we created a list of measurable objectives needed for our instrument to work over a broad range of performance racing sailplanes:

- Air Speed: 30 – 150 knots

- Altitude: sea level to 9,999'

- L/D ratio: 2:1 – 100:1

- Total system runtime of no less than 1 hour but 3 or more hours is preferred.

System Design

Before we could move onto implementation and choosing various system components, we had to define the critical components necessary to meet our design goals and objectives. As seen in Figure 19, several key modules were apparent when we referred to the Problem Statement and Measurement Objectives chapters including the following:

- GPS Module
- Liquid Crystal Display (LCD)
- Removable data storage media (prefer onboard)
- User Interface (push-button switches)
- System Controller (a microprocessor or FPGA)
- Portable Power Source
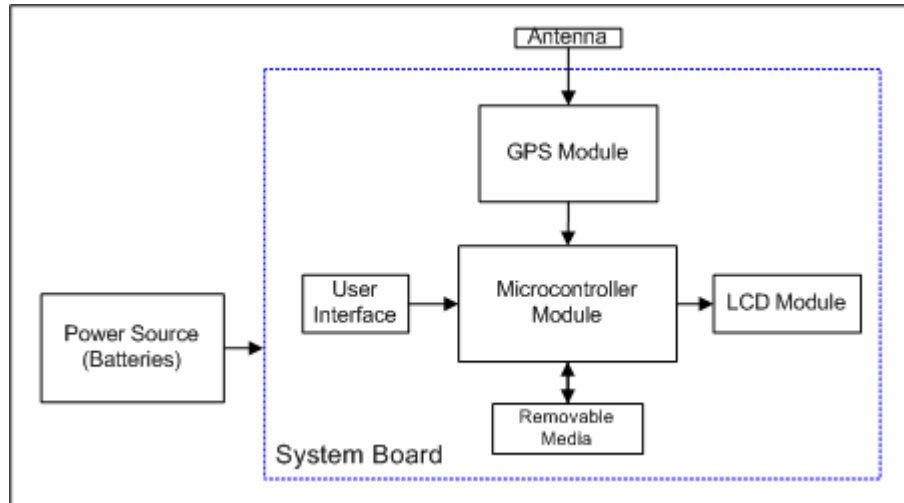
Figure 1: Top-level System Design

In order implement the system, we first began by conducting a trade study to select the different components to satisfy the requirements of each module. After the components were chosen, we had to test them individually to verify the hardware works and learn proper usage. Once that was completed, we then went on to test the interfacing of the modules. We began by interfacing the LCD with the FlashCoreB- (FB) microprocessor. We then interfaced the GPS receiver with the microprocessor and conducted a test separate of the LCD. When we found that it was possible to interface to the LCD and the GPS receiver separately, it was time to implement a full working system with both units working in unison.

Results

Figure 2 shows software flow diagram utilized in our design. One vital element necessary to achieve optimum accuracy and performance is a minimum of four satellites. Whenever less than four satellites are detected, the L/D calculations are no longer valid, and then the system must wait until a sufficient number of satellites have been detected again.

In order to ensure the system has not crashed, an incrementing timer has been included to show continued system activity, which is displayed in the lower right corner.

Incoming GPS data is transmitted via strings, which contain multiple pieces of information. The software looks for the strings containing the pertinent data needed for the calculations. All pertinent data is displayed on the corresponding screens while the L/D calculation results are stored on the CompactFlash card.

The user interface consists of two parts: three switches and the LCD. Button 2 is used to scroll through the various screens. Button 3 is used to jump to the L/D calculator screen. Button 1 is used for a confirmation or enter button (i.e. "Press When Ready").

On power up, the software will search for the number of satellites while simultaneously displaying the current number of satellites and the incrementing timer on the screen. This portion of software does not permit any user input. Once connections with at least four satellites have been established, the user must

acknowledge he/she is ready by pressing Button 1. The system waits until the acknowledgement button is pressed while still displaying the incrementing timer.

The user is able to scroll through different screens including velocity, altitude, L/D, current number of satellites, etc.
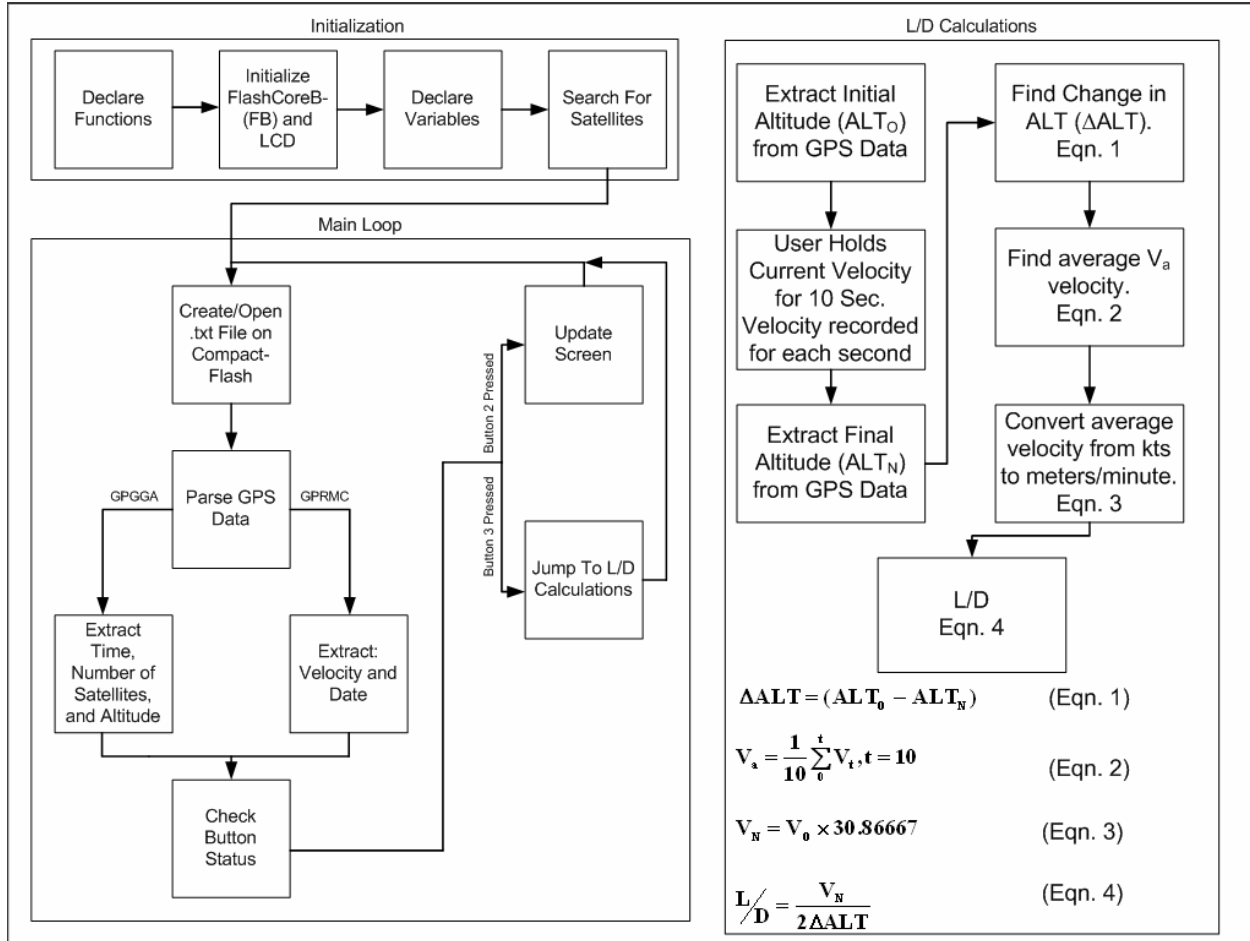
Figure 2: Software Flow

The calculation of L/D starts when button 3 is pressed. The calculation of L/D requires the extraction of velocity and altitude from the GPS strings. The user is asked to hold their velocity for 10 seconds, during which the velocity for each second is stored into an array. At the beginning and end of the 10 seconds, the initial altitude and final altitude are extracted to determine the change in altitude. The change in altitude is then multiplied by 6 to account for a minute duration of flying. The program then takes an average of the velocities stored from the 10 seconds and is then converted to meters/minute by multiplying the average velocity by 30.8667. The average velocity is then divided by the change in altitude, which yields the L/D result. The L/D is then added to another variable which stores the total of all the L/D's calculated. This allows for an average L/D of the day to be taken. The L/D calculation, velocity, and altitude are stored in the CompactFlash for future analysis.

One important step necessary in the development of our system was testing the accuracy of our GPS receiver, Garmin's 15H. According to the receiver's manual, it has an accuracy of up 3-5 meters with the

Differential GPS activated. In our system, we have this feature set to automatic to receive DGPS information whenever it is available. In order to test its accuracy, we first had to find some known locations and their altitudes. To do this, we went to the Worcester Public Library and asked for topographic maps. As seen in Table 1, the locations we found off of the topographic maps were Worcester Polytechnic Institute, Grafton Plaza, and Heywood Street. Since topographic maps do not give exact values, the values listed in the chart are rough readings based on the maps scales and lines of contour. The librarian also directed us to the library's homepage which listed a few additional locations. The ones we decided to use were City Hall and Union Station. Once this information was found, we then had to take the GPS receiver and our system to test these values. The values read from our test were then compared to the information we found at the library. The actual values achieved from our system were relatively close to the indicated accuracy of 3-5 meters as advertised by Garmin. Any discrepancies found in the values may have been the result of various reasons. The first reason that comes to mind is the fact that the values read from the topographic maps were estimates as opposed to accurate values. Other reasons could be errors in the GPS timing as described earlier. When the skies are not clear or the area is surrounded by high buildings and other structures, such as the case of City Hall, there is an added delay in receiving the GPS information from the satellites.

Location	Topography Elevation (Approximate)	GPS Elevation	Difference
WPI	167 m	169.9 m	2.9 m
Grafton Plaza	150 m	154 m	4 m
Heywood St	182.9 m	191 m	8.1 m
City Hall	146.6 m	140 m	6.6 m
Union Station	144 m	138 m	6 m

Table 1 - Elevation Comparisons

Summary and Conclusions

The project works and provides data reliable enough to give accurate readings when in calm air only. Flying after the sun has been able to heat the surface for more than a couple of minutes, the unstable air creates turbulence when flying which yields excessive variations in the data measurements due to the unstable atmosphere.

The project works and reports readings within the specified accuracy range of the Garmin GPS receiver when compared to the one-foot accuracy topographic map data. The enclosure has physical dimensions small enough to fit in the sailplane space available for this instrument. The CompactFlash Card allows for easy portability of pertinent data to the computer for data analysis.

Some expected future work includes designing a true buck-boost power distribution topology in place of the current boost to 12V then a point-of-load buck converter placed near high power loads. A more accurate L/D calculation should be researched and verified.

Future teams should research the possibility of including any physical factors of a sailplane to achieve a greater accuracy in the calculation of the L/D ratio. Currently, the GPS receiver reports data every second. The one Hertz data rate provides a major point of error since conditions change quite frequently in conditions where the air is even slightly unstable.

Future teams should implement a buck-boost power distribution topology to have an efficient version of the Flight Data System. The new topology would increase the efficiency from approximately 71% currently to 80-90% overall system efficiency.

Appendix B – Trade Study

One of the most important components in the design of our system was the microcontroller board. In order to choose a microcontroller board to use in our design, we researched commercially available microcontroller boards while keeping in mind how the microcontroller boards could interface to a computer. After conducting our research, the TERN FlashCore-B(FB) and IPOC1615 were the two best options after excluding those with excessive power consumption (more than three watts), dimension requirements (3.125" tall by 6.5" wide), or were too expensive (over $1000). In order to determine which board would be used in the design, we conducted a trade study to compare the important features of the different devices as listed below.

- Dimensions
- Input Voltage
- Power Consumption
- Operating Temperature Range
- Programmable I/O
- RS232
- Parallel/IDE
- Hardware/Software/Overall Cost

The comparison was organized into three sections including hardware, software, and cost as seen in Table 8. The bolded features were the key features used in our component selection.

Hardware	TERN[33]	ICOP[34]	PIC[35]
	FlashCore-B(FB)	6015	PIC24FJ128GA
Dimensions	**2.1" x 2.35" x 0.7"**	**3.94" x 2.60"**	**100-lead TQFP max**
Input Voltage	9-12V unregulated	5V regulated	2.0-3.6V regulated
Power Consumption	**700mW maximum**	**2.0W maximum**	**2.1W maximum**
Operating Temp. Range	-40 to +85	-20 to +60	-40 to +85
Programmable I/O	**20+ TTL lines**	**16-bit Digital I/O**	**84 TTL lines**
			2 UARTs support
RS232	2 ports	2 ports	RS232
Parallel/IDE	0/0		0/0
Software	Paradigm C/C++	DOS	MPLAB
		TASM Compiler	
Cost			
Hardware	$99 for board	$186 for board	$5.66/chip
Programmer	Included	Included	$86 PICStart Plus
	$249 Evaluation Kit		
Software	$699 Development Kit	Free X-DOS	Free MPLAB
Overall	**$351 or $798**	**$186**	**$92**

Table 8 - System Controller Trade Study

[33] http://tern.com/fc_b.htm
[34] http://www.icop.com.tw/products_detail.asp?ProductID=8
[35] http://ww1.microchip.com/downloads/en/DeviceDoc/39768a.pdf

Appendix C – LCD Test Board Schematic
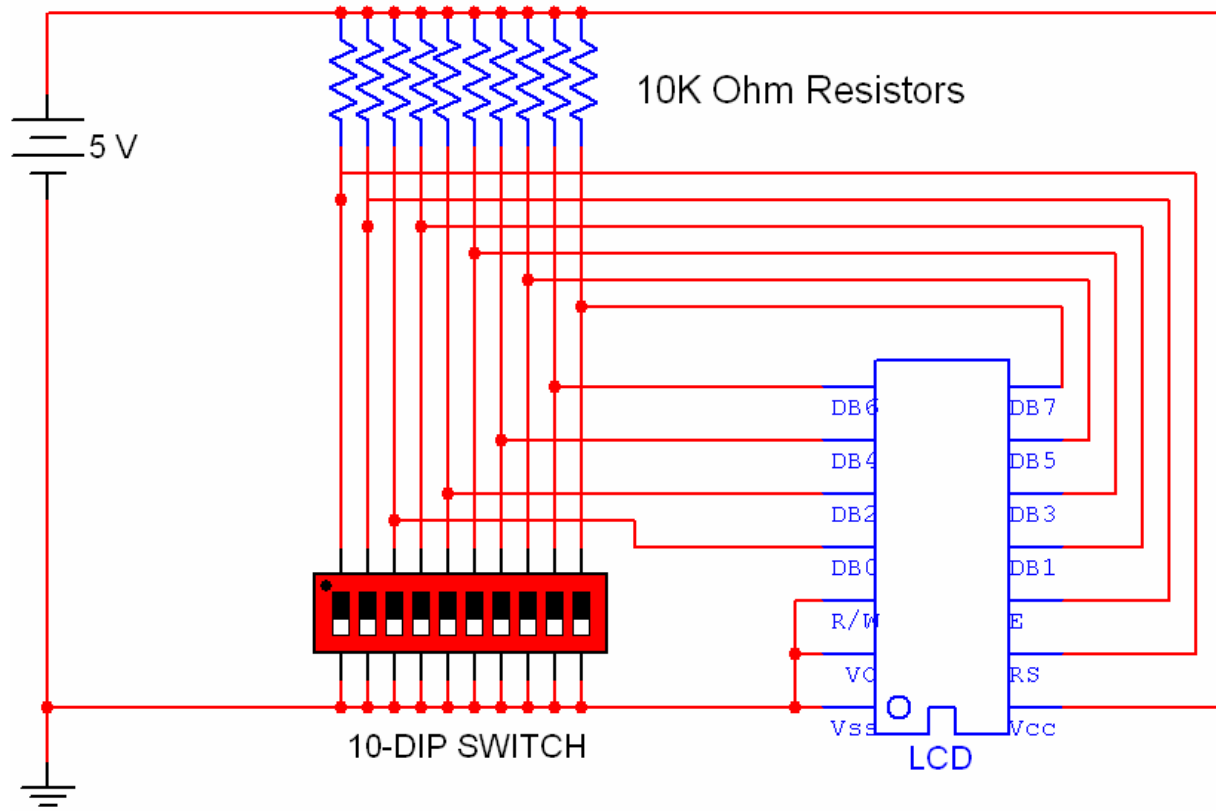
LCD Test Board Schematic

Appendix D – Data Logger Documents
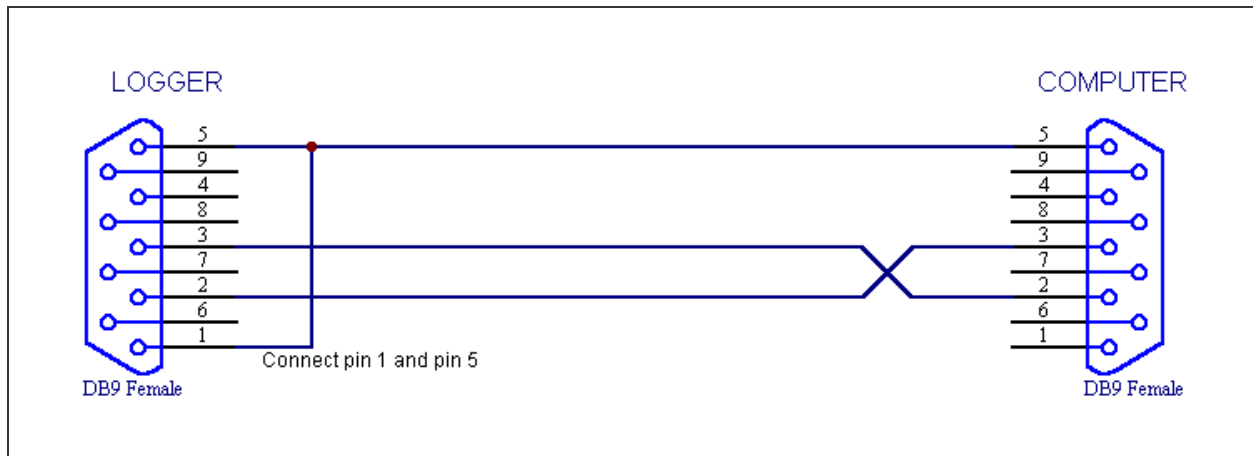
Data Logger Cable Pinout

Terminal Commands:

The following command set may be used with a terminal program, or as a guide for writing your own data logger
communications software.
Note:
Commands are accepted ONLY when pin 1 of the Data logger connector is connected to Ground.

Download:

Lowercase 'd'. The Data Logger will instantly download all stored data at the baudrate selected by DIP switch 1.
Datalogger will send 'ok' when the download is complete.
Download can be cancelled by pressing the Escape key.
Download can be cancelled by pressing pushbutton S2 (inside the battery compartment).

Info:

Lowercase 'i': The Data Logger will instantly return version and model number data.

Zero (Clear)

Uppercase 'Z' (Shift-Z). All data will be instantly cleared from memory.
Datalogger will reply with 'ok'

Appendix E – System Board Schematic

Appendix F – Enclosure Drawings
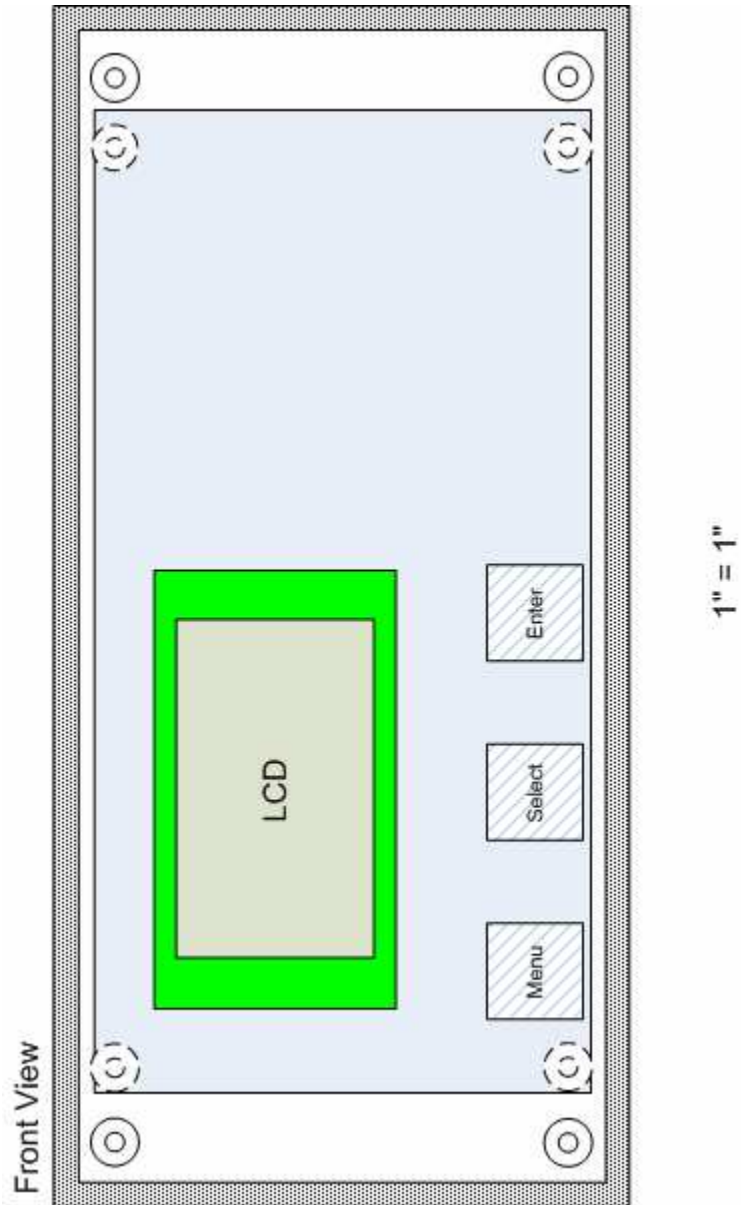
Enclosure – Front View
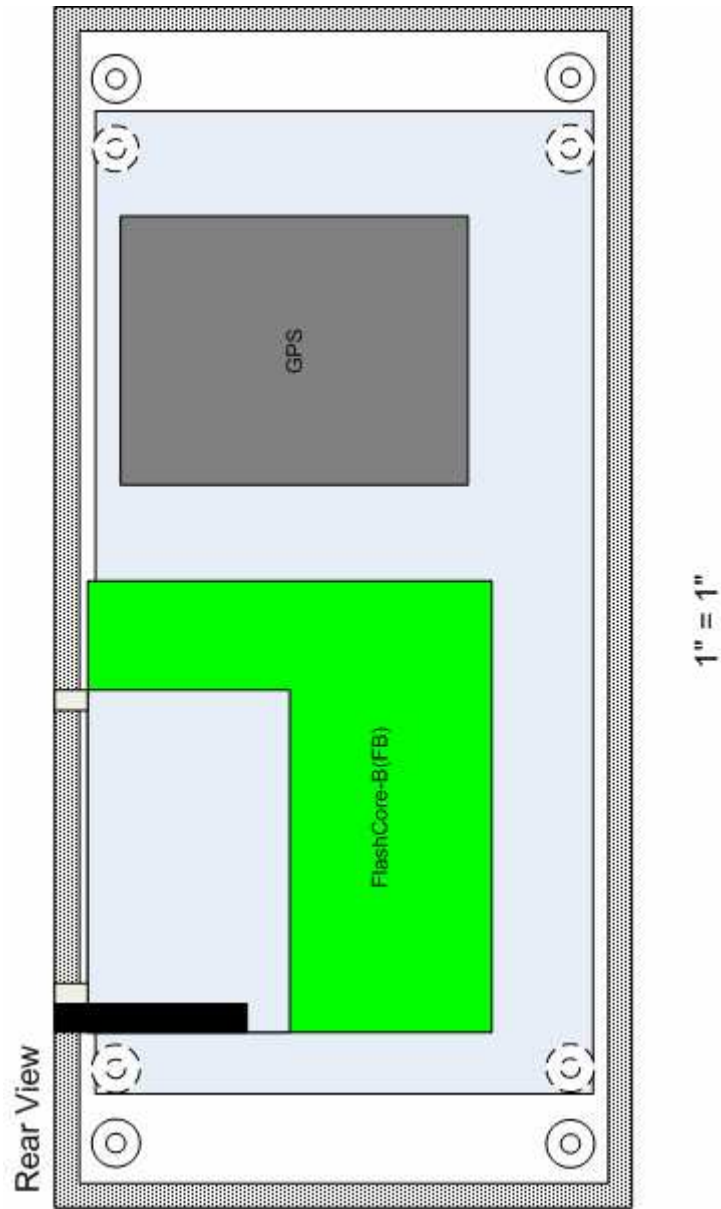
Enclosure – Rear View
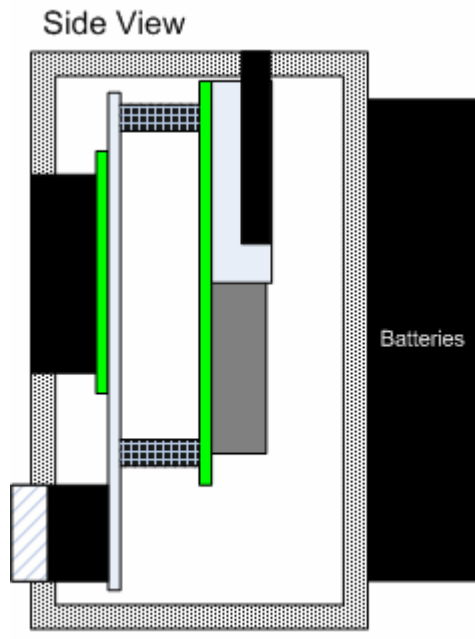
Enclosure – Side View

Appendix G – PCB Layouts
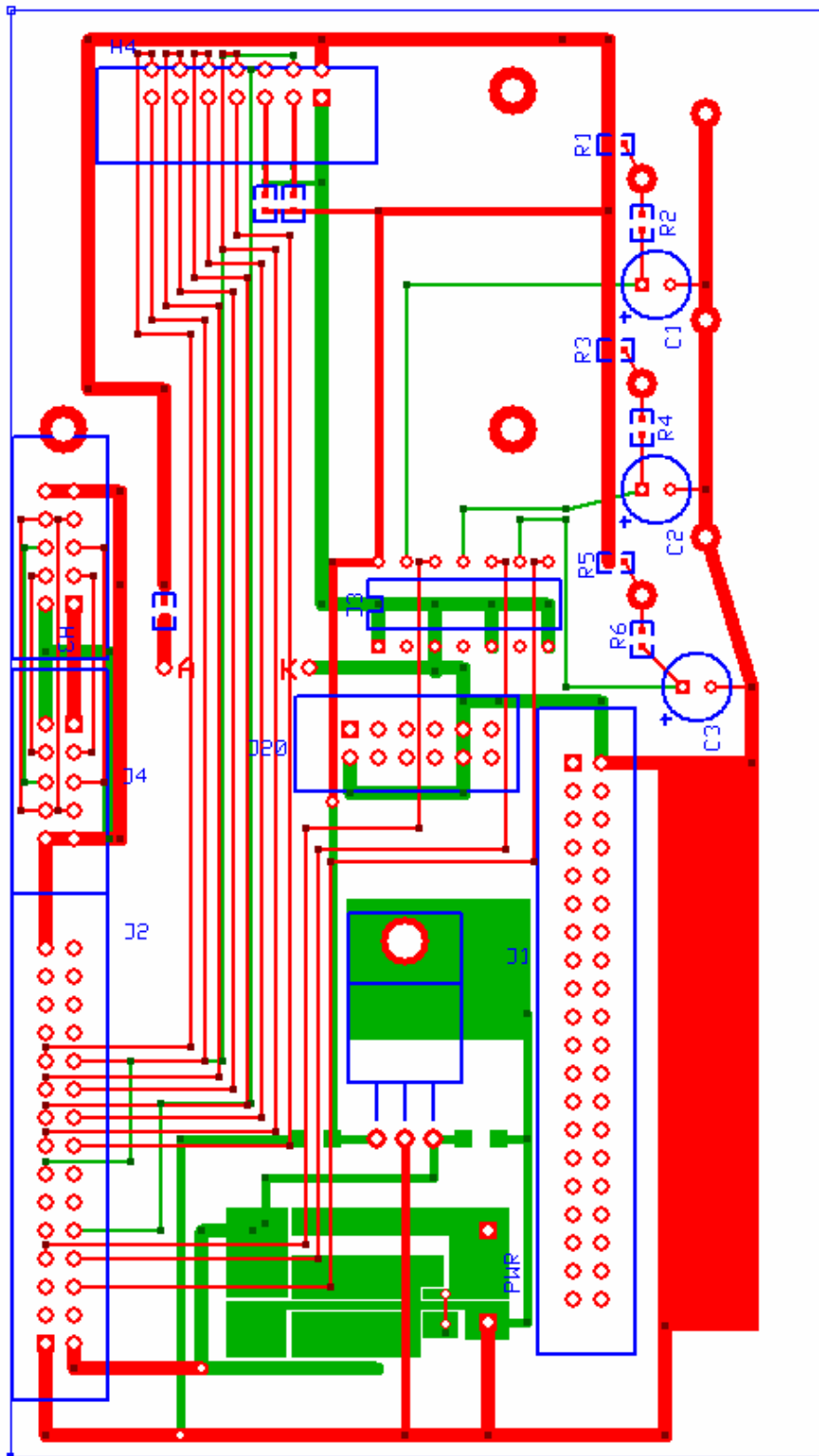
Complete PCB Layout
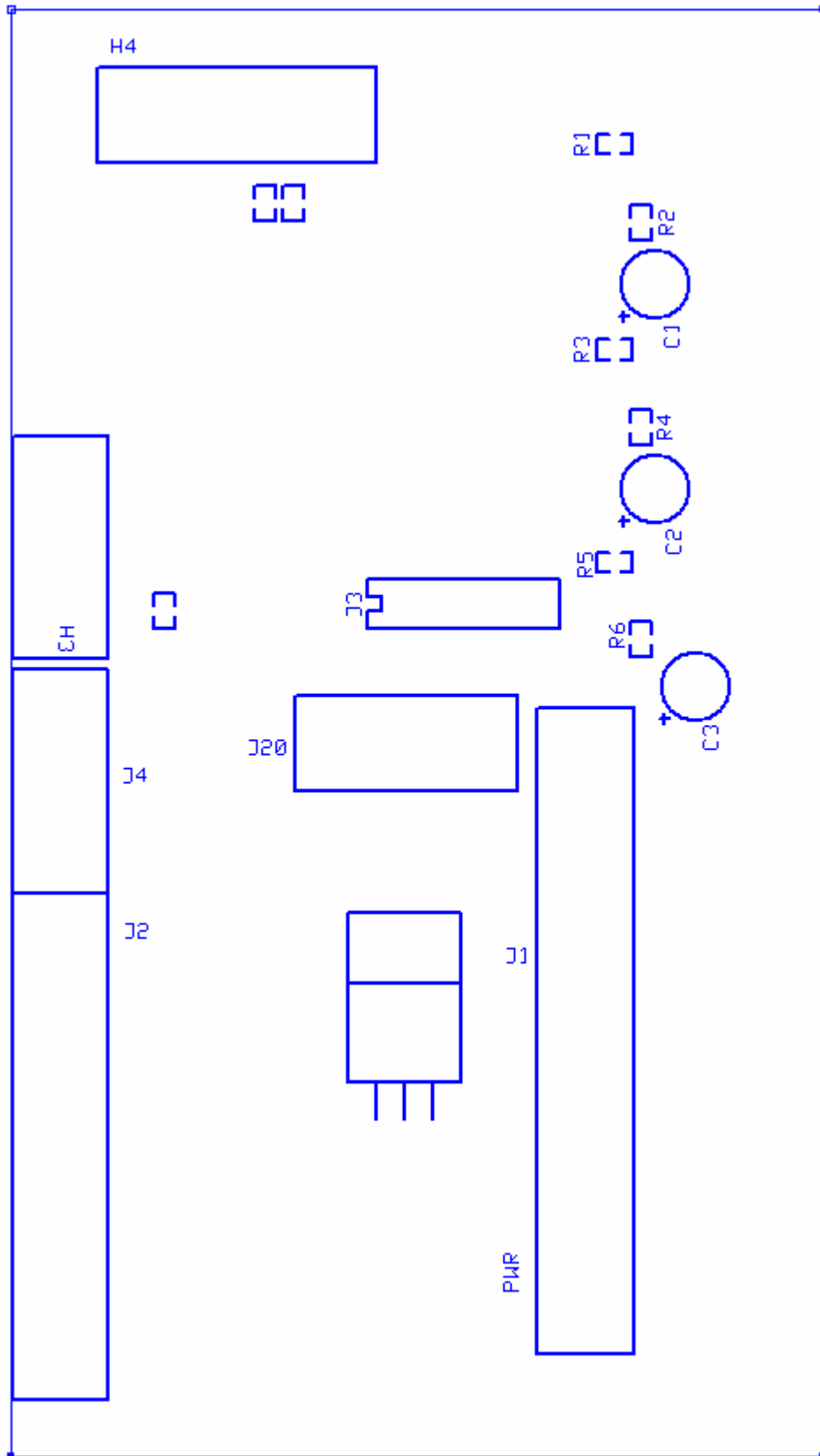
Silkscreen
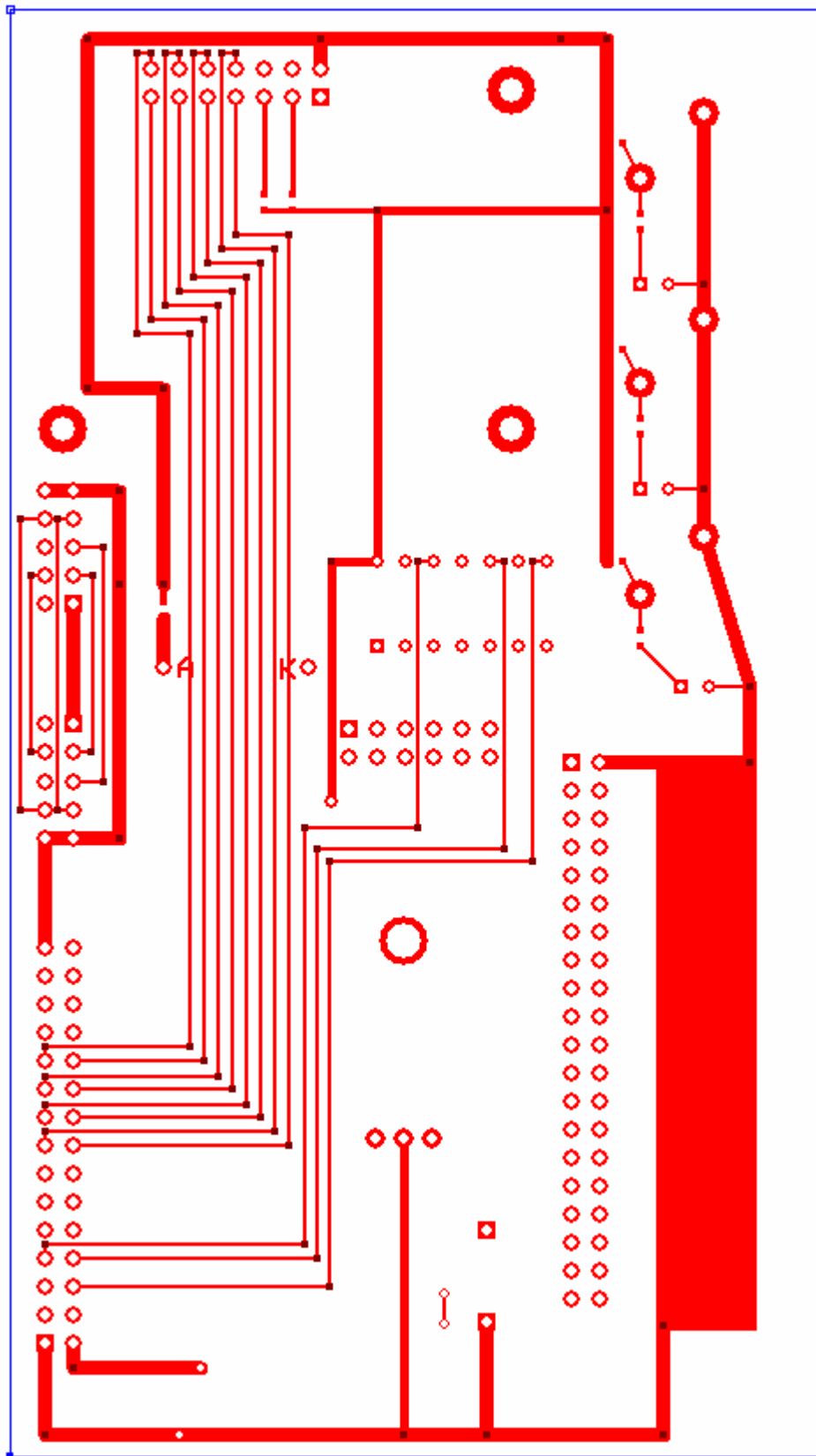
Top Layer
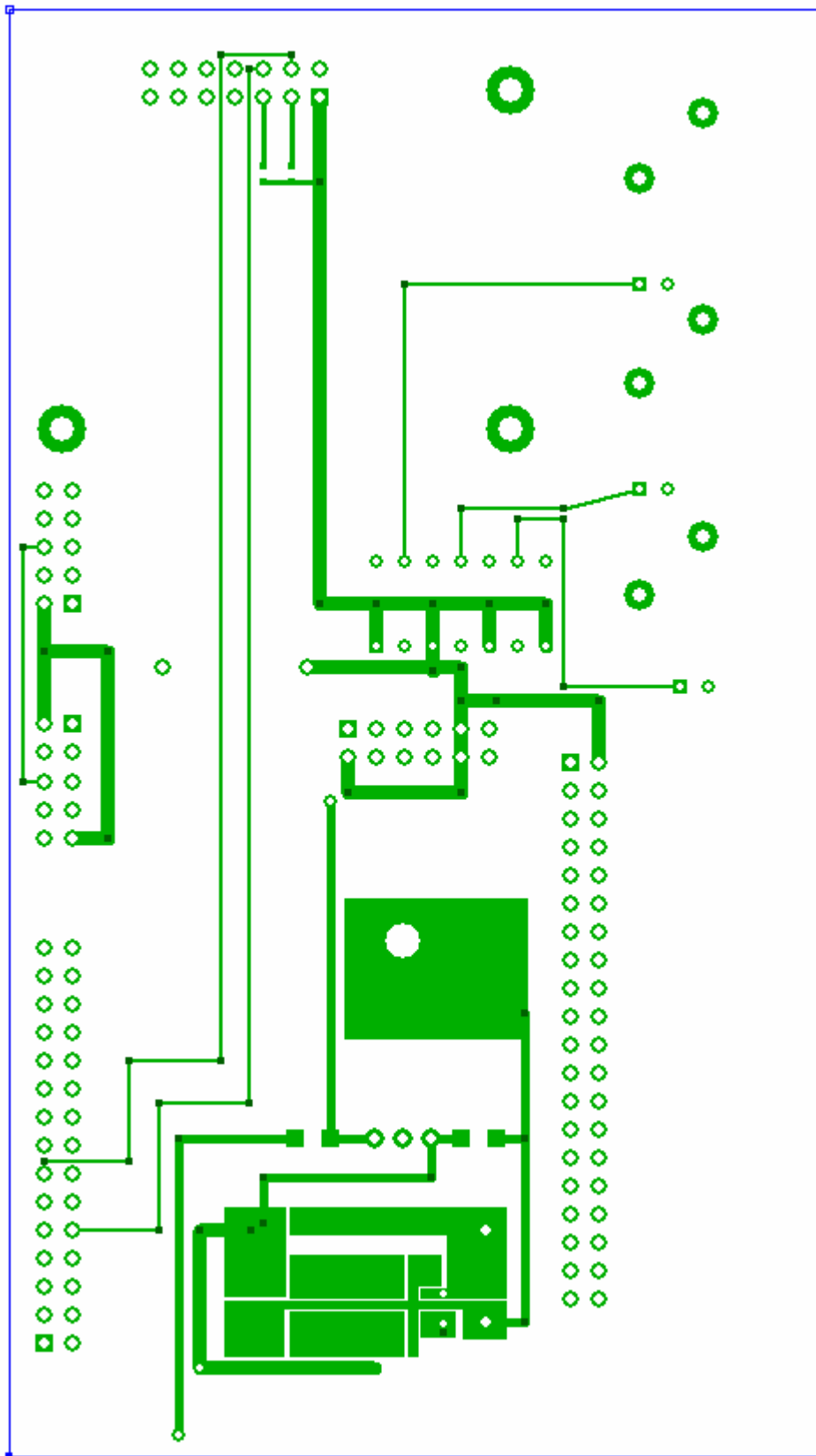
Bottom Layer

Appendix H – Flight Data System Program Code

```
extern "C"

{
      #include  "ae.h"                      // AE88 initialization
      #include  "ser1.h"
      #include "fileio.h"
      #include "string.h"
      #include "stdio.h"
}

#include <dos.h>
#include <string>
#include <stdlib.h>
#include "gpsstringreader.h"

#define BUFFSIZE 1024

#ifndef NULL
#define NULL 0
#endif

#define MIN_SATS 4

unsigned char inBuff[BUFFSIZE];
unsigned char outBuff[BUFFSIZE];
extern COM ser1_com;
COM* com1 = &ser1_com;

int FindNumSats(const char* gpsStr)
      {
            const char* curTok = gpsStr+41;
      int retVal = ((curTok[0]-0x30)*10)+(curTok[1]-0x30);
                        // returns number of satellites
            return retVal;                  // in decimal form
      }

double FindVelocity(const char* gpsStr)
      {
      char strVel[5]="";
      double velocity;
      for(int a=0; a<5; a++)
            {strVel[a] = gpsStr[a+41];}
            // Extracts velocity from GPS string
      velocity = atof(strVel);        // convert string to double
      return velocity;
   }
```

```
double FindAltitude(const char* gpsStr)
    {
    char strAlt[10]="";
    double altitude;
    for(int a=0; a<10; a++)
        {
            strAlt[a] = gpsStr[a+48];      // extract altitude from GPS string
            if(gpsStr[a+49]==',')          // Finds where altitude value ends
                {a = 11;}
        }
    altitude = atof(strAlt);        // Convert string to a float
    return altitude;
    }

void initports(void)
        {
            ae_init();                      // AE initialization
                                                    // Initialize all PIO
                            // Outputs
                                                    // pio_init(x,y)
                                                    // x = port number
                                                    // if y = 1, port is input
                        // if y = 2, port is output
            pio_init(0,2);     // DB5
            pio_init(3,2);     // Register Select (RS)
            pio_init(10,2);    // DB6
            pio_init(13,2);    // DB7
            pio_init(17,2);    // DB2
            pio_init(18,2);    // DB1
            pio_init(19,2);    // DB0
            pio_init(24,2);    // DB3
            pio_init(25,2);    // DB4
            pio_init(30,2);    // Enable
                                                    // Input with Pull Up/Down
            pio_init(15,1);    // Switch 1
            pio_init(5,1);     // Switch 2
            pio_init(6,1);     // Switch 3
        }

void  set_pio(int num_data)
        {
        int datatmp = num_data & 0x01;
            datatmp = datatmp/0x01;
            pio_wr(19,datatmp);                         // Set DB0

          datatmp = num_data & 0x02;
            datatmp = datatmp/0x02;
            pio_wr(18,datatmp);                         // Set DB1

            datatmp = num_data & 0x04;
            datatmp = datatmp/0x04;
            pio_wr(17,datatmp);                         // Set DB2

            datatmp = num_data & 0x08;
            datatmp = datatmp/0x08;
            pio_wr(24,datatmp);                         // Set DB3
            datatmp = num_data & 0x10;
            datatmp = datatmp/0x10;
            pio_wr(25,datatmp);                         // Set DB4
```

```
            datatmp = num_data & 0x20;
            datatmp = datatmp/0x20;
            pio_wr(0,datatmp);                              // Set DB5

            datatmp = num_data & 0x40;
            datatmp = datatmp/0x40;
            pio_wr(10,datatmp);                             // Set DB6

       datatmp = num_data & 0x80;
            datatmp = datatmp/0x80;
            pio_wr(13,datatmp);                             // Set DB7
   }

void initLCD(void)
     {
     delay_ms(15);                       // Allow Vcc to rise to 4.5V
  pio_wr(3,0);          // Set RS to Instruction
     pio_wr(30,1);
     set_pio(0x30);             // Function Set
     pio_wr(30,0);
     delay_ms(5);
     pio_wr(30,1);
     set_pio(0x30);             // Function Set
     pio_wr(30,0);
     delay_ms(1);
     pio_wr(30,1);
     set_pio(0x30);             // Function Set
     pio_wr(30,0);
     delay_ms(1);
  pio_wr(30,1);
     set_pio(0x38);             // Set interface data length(8 bits)
                       // and # of lines(2)
     pio_wr(30,0);     // font size(5x8 dots)
     delay_ms(1);
  pio_wr(30,1);
     set_pio(0x08);             // Display off
     pio_wr(30,0);
     delay_ms(1);
     pio_wr(30,1);
     set_pio(0x01);             // Clear display
     pio_wr(30,0);
     delay_ms(2);
     pio_wr(30,1);
     set_pio(0x06);             // Assign cursor moving direction (right)
     pio_wr(30,0);
     delay_ms(1);
     pio_wr(30,1);
     set_pio(0x0E);             // Set display, set cursor, blinking
cursor(off)
     pio_wr(30,0);
     delay_ms(1);
     }
void clr_scr(void)
     {
          pio_wr(30,1);
     pio_wr(3,0);
              set_pio(0x01);            //clears screen
        pio_wr(30,0);
     delay_ms(5);
```

```
      }

void char_pos(int row,int col)                // row 1-2, col 0-7
      {
            pio_wr(3,0);                        // RS=0 ==> Instruction
            pio_wr(30,1);

            if(row==2)
            {col = col+0x40;} // Adjusts DDRAM address to be written
                              // DDRAM address of LCD
                              // controls cursor position
      set_pio(col);          // Write DDRAM address to LCD
            pio_wr(13,1);
         pio_wr(30,0); //places cursor
      delay_ms(2);
   }

void str_wr(char* string)
      {
      int r=0;
            while(r<8)
                  {
            pio_wr(3,1);               // RS = data
            // string[r] extracts a single character
            // character is then set to the LCD using set_pio()
            set_pio(string[r]);
         pio_wr(30,1);
         pio_wr(30,0); //displays character
         r++;                          // Next character

         if(string[r]==0x00) // Checks end of the string
            {r=8;}
            }
      }

int btn_rd(int btn_num)            //reads btn status
{
      int btntemp;
      int btn_loc;                                 // determines how many bits
to shift btn

      if(btn_num == 1)             //  Checks the proper button
            btn_loc = 15;
      if(btn_num == 2)
            btn_loc = 5;
      if(btn_num == 3)
      btn_loc = 6;

   btntemp = pio_rd(0);
   btntemp = (btntemp >> btn_loc);
   btntemp = (btntemp & 0x01);
   return btntemp;
}

// ****************** End Functions *************************
```

```
void main(void)
{

// ******************* Initializations ************************
   initports();
   initLCD();
   unsigned char baud = 7;              //intialize the serial port  @ 4800 baud
   s1_init(baud, inBuff, BUFFSIZE,  outBuff, BUFFSIZE, com1);

   fs_descrip* file = NULL;   //initialize flash
   if(fs_initPCFlash()!=0)            // Checks for compact flash card
   {                                  // User must power down and
                                      // insert card if not found
      clr_scr();
      str_wr("Insert");
      char_pos(2,0);
      str_wr("CF Card");
      while(true)
      {
          delay_ms(999);
      }
   }
// ******************* Variable Declarations ************************
   int      btn1=0x00;                      // Holds button status
   int      btn2=0x00;                      //
   int      btn3=0x00;                      //
   int      disp_type = 0x00;                     // Determines what to display
   int      clear = 0x00;                   // Holds clear status
                                      // Determines if screen needs to be
                                      //  cleared for new display

      float LD = 0.00;                                 // Stores L/D Value
   long double LD_total =0;                 // Stores total of LD values
                                      // to be averaged
   unsigned int LD_num =0;             // Stores how many LD values have
                                      // been calculated
   float LD_avg = 0;                             // Average of all the
calculated LD's
   char     strLD[5]="";            // String format of LD to be displayed
   bool     calcLD = false;               // Used to determine whether or
                                    // not to calculate LD
   char     strType[6] = {0,};             // Stores NMEA GPS String type
   char     strnewTime[7] = "";            // Stores new time
   char     strTime[7] = "";        // Stores previous time
   char     strVel[5]="";                        // Stores Velocity -- string
format
   double   Vel = 0;               // Stores velocity in knots
   char     strAlt[10]="";                       // Stores Altitude -- string
format
   double   Alt = 0;                           // Stores Altitude in meters
      GPSStringReader sr(com1);         // Object to read GPS strings
   char* gpsStr = NULL;                   // The string being read in from
GPS
   char     curGPS[100] = "";             // The last gps string we care
about
   int      numSats = 0;                        // The current number of
satellites
                                    // in view
   char     strSats[2]="";                       // Sats in string
```

```
   char      strDate[9]="";
//***********************End Variables*******************************
   char_pos(1,0);                                    // Reset cursor to Row
1, Col 0
   str_wr("Begin?");                                 // Ask user if ready
   char_pos(2,0);
   str_wr("Yes 1");                     // If ready, press button 1

   bool not_ready = true;
   while(not_ready)
   {
      btn1 = btn_rd(1);
      if(btn1 == 0x01)                  // Check to see if button1
                                        // has been pressed
      {
           while(btn1 == 0x01)
         {
         btn1 = btn_rd(1);              // Check to see if button1
                                        // has been released
           if(btn1 == 0x00)
           {not_ready = false;}
           }
           clr_scr();
      }
   }

   NES:                                              // NES = Not Enough
Satellites
      char_pos(1,0);         // If there aren't enough satellites,
                             // program will jump here
        str_wr("Sats= ");    // and not allow anything to happen
                             // until there is enough
      str_wr(strSats);

      char_pos(2,0);
        str_wr("Wait");
      disp_type = 0x00;      // In the event there is a jump to NES
                             // during main loop
      clear = 0x00;                  // The display is reset back to
                             // satellite display
   while(true)
   {
   MAIN:
   /*****FILE Create / Open***************************/
         if(!file)
       {
         file = fs_fopen("avionics.txt", O_WRITE|O_APPEND);
         if(file&&(file->ff_status!=fOK))
         {//Make sure it opened ok
              fs_fclose(file);
            file = NULL;
         }
       }
```

```
 /******GPS STRING HANDLING**********************/

   if(gpsStr = sr.GetString())
   {
      for(int i=0;i<5;i++)
      strType[i] = gpsStr[i+1];                    // Figure out which GPS
                                        // string is being looked at

      if(!strcmp(strType, "GPGGA"))
      {
            for(int c=0;c<6;c++)                           // For
extracting Time
         strnewTime[c]= gpsStr[c+7];

         numSats = FindNumSats(gpsStr);   // Get number of
                                          // satellites
         itoa(numSats,strSats,10);
         Alt = FindAltitude(gpsStr);
         strcpy(curGPS, gpsStr);

         if(strcmp(strnewTime, strTime))        // Need to record
         {
                 strcpy(strTime, strnewTime);
            char_pos(2,7);
            char temp[1] = "";
            temp[0] = strTime[5];
            str_wr(temp);
         }
      }
   else if(!strcmp(strType, "GPRMC"))
   {
      Vel = FindVelocity(gpsStr);                // Get Velocity

      strDate[0]= gpsStr[55];       // Gets the date
      strDate[1]= gpsStr[56];            // Convert from DDMMYY format
      strDate[2]= '/';              // To MM/DD/YY format
      strDate[3]= gpsStr[53];
      strDate[4]= gpsStr[54];
      strDate[5]= '/';
      strDate[6]= gpsStr[57];
      strDate[7]= gpsStr[58];
      strDate[8]= '\0';
   }
}
      if(numSats < MIN_SATS)
      {goto NES;}
/*****Screen Scrolling************************/
      btn2 = btn_rd(2);
      if(btn2 == 0x01)                        // Checks to see if button 2
                           // has been pressed
      {
            while(btn2 == 0x01)
         {
                 btn2 = btn_rd(2);               // Checks to see if button 2
                           // has been released
         }
      disp_type++;                              // Scrolls to next screen
            if(disp_type > 0x05)         // If on final screen,
                           // go back to first one
```

```
        {disp_type = 0x00;}
    }

        btn3 = btn_rd(3);                       // Checks to see if button 3
                            // has been pressed
    if(btn3 == 0x01)
    {
        while(btn3 == 0x01)         // Checks to see if button 3
                            // has been released
    {
            btn3 = btn_rd(3);
    }
    calcLD = true;                      // If button 3 has been pressed,
                            // go to calculated L/D mode
    goto LD;
    }

        if(disp_type == 0x00)         // Screen 1 - Number of satellites
    {
        if(clear == 0x00)
    {
    clr_scr();                      // Refresh screen
    clear=0x01;
        str_wr("Sats= ");
    }
    char_pos(2,0);
    str_wr(strSats);
    str_wr(" ");                    // In the event new value is less
                        // than the previous value
                        // by at least a significant digit
                        // i.e - old value = 10, new value = 9
                        // --> the zero will still be displayed
                        // by adding the spaces,
                        // the excess digits are "erased"
    }
if(disp_type == 0x01)      // Screen 2 - Velocity
    {
        if(clear == 0x01)
    {
    clr_scr();
    clear=0x02;
    str_wr("Vel=");
    }
    char_pos(2,0);
    gcvt(Vel,4,strVel);
    str_wr(strVel);
    str_wr(" ");
    }

    if(disp_type == 0x02)         // Screen 3 - ALtitude
    {
        if(clear == 0x02)
    {
    clr_scr();
    clear=0x03;
    str_wr("Altitude");
    }
    char_pos(2,0);
    gcvt(Alt,6,strAlt);
```

```
        str_wr(strAlt);
        }

        if(disp_type == 0x03)          // Screen 4 – Last L/D Calculated
        {
            if(clear == 0x03)
        {
        clr_scr();
        clear = 0x04;
        str_wr("Last L/D");
        }
        char_pos(2,0);
        gcvt(LD,5,strLD);
        str_wr(strLD);
        }

        if(disp_type == 0x04)          // Screen 5 – Average of all L/D's
        {
            if(clear == 0x04)
        {
        clr_scr();
        clear = 0x05;
        str_wr("Avg L/D");
        }
        char_pos(2,0);
        gcvt(LD_avg,5,strLD);
        str_wr(strLD);
        }
if(disp_type == 0x05)  // Screen 6 – Date and Time
        {
            if(clear == 0x05)
        {
        clr_scr();
        clear = 0x00;
        }
        str_wr(strDate);

        long temp_time = atol(strnewTime);
            if(temp_time >= 50000)
            temp_time = temp_time – 40000;
            // Convert UTC to Eastern Time (UTC from 5am – 11:59pm)
        if(temp_time <= 40000)
            temp_time = temp_time + 80000;
            // Convert UTC to Eastern Time (UTC from 12am to 4:59am)
        if(temp_time >= 130000)
            temp_time = temp_time – 120000;
            // Convert from military time to standard time
        temp_time = temp_time + 100000;
            // Place holder for conversion from int to string
        ltoa(temp_time,strTime,10);
        char_pos(2,0);
        char temp[8];
        temp[0] = strTime[0] – 1;
            //    Handles hour correction from place holder
        temp[1] = strTime[1];   // Convert time format from HHMMSS format
        temp[2] = ':';          // To HH:MM:SS format
        temp[3] = strTime[2];   // Also converts to standard time
        temp[4] = strTime[3];   // Instead of military time
        temp[5] = ':';
```

```
      temp[6] = strTime[4];
      temp[7] = strTime[5];
      str_wr(temp);
      temp_time = 0;                    // reset temp_time
      }
/**************************************************/
LD:
      while(calcLD == true)
   {
   char    cnt_dwn[2]="";
   char    strCnt5[1]="";
   int     cnt = 30;                                // Time remaining
   int     cnt5 = 5;                                // 5 second setup time
   int     count=0;                                    // counter
   double  secVel[10]={0};         // Stores Velocity of every second
   double  alt_init=0;             // Stores Initial Altitude
      double    alt_final=0;            // Stores Final Altitude

      clr_scr();
      str_wr("Hold Vel");

while(count <= 15)
        {
          if(gpsStr = sr.GetString())
                {
                      for(int i=0;i<5;i++)
                      strType[i] = gpsStr[i+1];                        //
Get string type

                      if(!strcmp(strType, "GPGGA"))
                      {
                              for(int c=0;c<6;c++)
      // For extracting Time
                      strnewTime[c]= gpsStr[c+7];

                  numSats = FindNumSats(gpsStr);
                  strcpy(curGPS, gpsStr);

              if(count == 5)                               //extract initial
altitude
              {alt_init = FindAltitude(gpsStr);}

                if(count == 15)                             //extract initial
altitude
              {alt_final = FindAltitude(gpsStr);}

              if(strcmp(strnewTime, strTime))
              {
                 strcpy(strTime, strnewTime);
                 char_pos(2,6);
                 if(cnt5 >= 0)          // 5 second setup delay
                 {
                 char_pos(2,0);
                 str_wr("StartIn");
                 itoa(cnt5,strCnt5,10);
                 str_wr(strCnt5);
                 cnt5--;
                 }
                 if(count >= 5)         // 10 second data record
```

```
                      {
                            clr_scr();
                      str_wr("Hold Vel");
                      char_pos(2,6);
                          itoa(cnt,cnt_dwn,10);
                          if(cnt < 10)
                                  str_wr("0");
                      str_wr(cnt_dwn);
                      cnt--;
                      }
                            count++;
                      }
              }

              else if(numSats < MIN_SATS)
                      {calcLD = false;
                 disp_type = 0x00;
                 clear = 0x00;
                 goto NES;}

              else if(!strcmp(strType, "GPRMC") && count >= 5)
              {
                      Vel = FindVelocity(gpsStr);
                 secVel[count-5] = Vel;
// Records Velocity over 10 seconds to find average
              }
                      }
        }

     calcLD = false;
           // Done with extraction, time for calculations.

     double avg_vel = 0;
     double temp_vel = 0;

     double delta_alt = alt_init - alt_final;
           // change in altitude over 10 sec
     delta_alt = 6 * delta_alt;
           // approximation of change in altitude over 1 min

     if(delta_alt <= 0)
     goto MAIN;

           // If there was no change in altitude, result is invalid

     for(int vel_t = 1; vel_t<=10; vel_t++)
                 {temp_vel = temp_vel + secVel[vel_t];}
           // Average velocity over 10 seconds
     avg_vel = temp_vel/10;

     float vel_mpm = avg_vel * 30.8666667;
           // Convert knots to meters per min


           // (1 knot = 30.8667 meters per minute)
     LD = vel_mpm/delta_alt;
           // L/D Approximated using Distance / Sink Rate
     LD_num++;
     LD_total = LD_total + LD;
```

```
        LD_avg = LD_total/LD_num;
             // Calculates the average of all the L/D's calculated

    clr_scr();
    disp_type = 0x03;
    clear = 0x03;
    fs_fprintf(file,"%s \r\n",strDate);

    fs_fprintf(file,"L/D = %lf; Average Velocity = %lf; Initial
              Altitude = %lf; Final Altitude = %lf \r\n\r\n",
                           LD,avg_vel,alt_init,alt_final);

    fs_fflush(file);
   }
   goto MAIN;
    }          // end the main loop
}                     // end main procedure
```

Appendix I – Used GPS Strings[36]

***GPGGA** - essential fix data which provide 3D location and accuracy data.*

```
 $GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

Where:
    GGA          Global Positioning System Fix Data
    123519       Fix taken at 12:35:19 UTC
    4807.038,N   Latitude 48 deg 07.038' N
    01131.000,E  Longitude 11 deg 31.000' E
    1            Fix quality: 0 = invalid
                             1 = GPS fix (SPS)
                             2 = DGPS fix
                             3 = PPS fix
                         4 = Real Time Kinematic
                         5 = Float RTK
                          6 = estimated (dead reckoning) (2.3
feature)
                         7 = Manual input mode
                         8 = Simulation mode
    08           Number of satellites being tracked
    0.9          Horizontal dilution of position
    545.4,M      Altitude, Meters, above mean sea level
    46.9,M       Height of geoid (mean sea level) above WGS84
                     ellipsoid
    (empty field) time in seconds since last DGPS update
    (empty field) DGPS station ID number
    *47          the checksum data, always begins with *
```

---

[36] The following have been taken from
http://www.gpsinformation.org/dale/nmea.htm#GGA

GPRMC - NMEA has its own version of essential gps pvt (position, velocity, time) data. It is called RMC, The Recommended Minimum, which will look similar to:

*$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A*

Where:
```
    RMC           Recommended Minimum sentence C
    123519        Fix taken at 12:35:19 UTC
    A             Status A=active or V=Void.
    4807.038,N    Latitude 48 deg 07.038' N
    01131.000,E   Longitude 11 deg 31.000' E
    022.4         Speed over the ground in knots
    084.4         Track angle in degrees True
    230394        Date - 23rd of March 1994
    003.1,W       Magnetic Variation
    *6A           The checksum data, always begins with *
```