General Purpose GPS Navigation Module

MQP Ver. 1.9

3/4/10

Nicholas Granata – Cappabianca

capps@wpi.edu

WPI box number 724

ECE box number 7
Table of Contents

Table of Contents .......................................................................................................................................... 2
1.0 Introduction ............................................................................................................................................. 4
1.2 Project Statement ................................................................................................................................... 4
1.3 Summary ................................................................................................................................................. 5
2.0 Introduction ............................................................................................................................................. 6
2.1 Soaring ..................................................................................................................................................... 6
2.1.1 Glider Lift ............................................................................................................................................ 10
2.1.2 Glider Performance ............................................................................................................................ 11
2.2 GPS ........................................................................................................................................................ 12
2.3 Flight Data Systems ............................................................................................................................... 14
2.4 Summary ............................................................................................................................................... 18
3.0 Introduction ........................................................................................................................................... 19
3.1 Project Statement ................................................................................................................................. 19
3.2 System Requirements ........................................................................................................................... 19
3.3 Summary ............................................................................................................................................... 19
4.0 Introduction ........................................................................................................................................... 20
4.1 Block Diagram ........................................................................................................................................ 20
4.2 Description of Modules ......................................................................................................................... 22
4.2.1 User Display ........................................................................................................................................ 22
4.2.1.1 Detailed Data Display Formats ........................................................................................................ 22
4.2.2 GPS ..................................................................................................................................................... 23
4.2.3 Antenna .............................................................................................................................................. 25
4.2.4 Input ................................................................................................................................................... 26
4.2.5 MCU – Micro Controller Unit ............................................................................................................. 26
4.2.6 USB ..................................................................................................................................................... 27
4.2.7 Batteries ............................................................................................................................................. 28
4.2.8 Power Circuitry .................................................................................................................................. 29
4.3 Summary ............................................................................................................................................... 31
5.0 Introduction ........................................................................................................................................... 32
5.1 The Device ............................................................................................................................................. 32
Chapter 1

Introduction

1.0 Introduction

Every glider pilot loves to be in the air, soaring amongst the clouds thousands of feet above the ground, flying for miles, being one with the aircraft. Although all glider pilots have a sense of where they are and if they can make it back, to their home airport, it is always nice to fly for as long as possible and to have an instrument of some sort that clearly indicates where home is, how far away the home airport is, and based on the particular performance characteristics of the glider the pilot is flying, whether the glider is within gliding range of the home airport. Fortunately, such instruments do exist and instruments that use GPS can calculate the sailplane’s position and accurately indicate to the pilot the information needed.

Unfortunately, most aircraft GPS navigation systems are not designed for use in gliders and the ones that are can be very expensive (Chapter 2). As a result, many glider pilots use a PDA which must be mounted in a large bulky arm system. The PDA either uses an external Bluetooth GPS receiver or has a cable to a GPS system somewhere within the cockpit. To complete the system, the pilot purchases and installs software specifically designed for glider navigation and then uses this system to navigate while flying. Pilots who are fortunate enough to be able to buy a GPS system that fits in their instrument panel are lucky since the panel mounted glider GPS systems are more functional and fit perfectly into the panel.

The problem with panel mounted GPS systems is that they can be very expensive (Chapter 2, Background). Also, both systems, the external PDA and the panel mounted GPS systems, are typically permanently mounted in the sailplane (with the exception of the PDA itself) and are not easily removed. This, in turn, is a problem for any pilot who flies many different gliders such as instructors or students. What would be useful in this situation would be a small, limited capability, battery powered, fully self contained glider GPS navigation system that could be easily moved from glider to glider.

1.2 Project Statement

The purpose of this project was to create a portable GPS device intended for use in a sailplane. The device will be very low power, low profile, relatively inexpensive, programmable, and small in size to
save cockpit space. Also, it will display bearing, altitude, speed, and final glide to the designated home airport based on the glider performance entered by the pilot.

1.3 Summary

We have presented a brief overview of the rationale behind this project and briefly stated the purpose of this project activity. In chapter 2 we will provide background material on soaring, how GPS works, and similar products that are already on the market. In chapter 3 a detailed project statement is given including the product requirements and desires. In chapter 4 a system level design is described. Chapter 5 provides a circuit level design. Chapter 6 describes the programming used in the device. Chapter 7 discusses the results and testing of the final device.
Chapter 2

Background

2.0 Introduction

This chapter presents an overview of material that is pertinent to understanding this project report. Subjects covered include soaring, GPS, and an overview of products already on the market.

2.1 Soaring

Gliding, or soaring, is a sport in which pilots fly unpowered aircraft, called gliders or sailplanes (Figure 1). Sailplanes consist of three main parts; the fuselage, the wings, and the tail. The fuselage is the main body of the sailplane and contains the cockpit which holds the controls and instrument panel (Figure 2). The wings provide the lift the sailplane needs to fly. They also house the ailerons, which control the roll of the plane, and the air brakes, which increase drag and the angle of approach while landing. The tail holds the elevator, which controls the pitch of the aircraft, and the rudder, which controls the yaw. Figure 3 depicts roll, pitch, and yaw and also the parts of the plane.

Figure 1 – A standard class high performance sailplane (Picture provided by F. Looft WPI).
Figure 2 – The cockpit of the sailplane from the viewpoint of the pilot. The displays are the instruments that are mounted in the instrument panel (Picture provided by F. Looft WPI).
Figure 3 – This shows the effect of pitch, roll, and yaw on the sailplane. Also the part of the plane and their names are shown (Gliding, A Handbook on Soaring Flight, Piggott 1997).

The instruments that a sailplane pilot has access to are located in the instrument panel of the cockpit, (Figure 2) located directly in front of the pilot. The instruments, shown in Figure 4, that all gliders have are the airspeed indicator, the altimeter, the variometer, a yaw string, and a compass. The air speed indicator works by comparing the pressure in a forward facing tube and the static air pressure. The altimeter indicates altitude and works by using a sensitive barometer calibrated to read height. The variometer measure the rate of climb or decent of the aircraft. There are different ways a variometer can be implemented but the basic mechanical concept is that a small tank of air is monitored and the
rate at which air flows through the instrument indicates the rate of climb or descent. When the aircraft is climbing air flows from the tank through the instrument and then outside. When the aircraft is descending the air pressure outside is greater than inside the tank and air flows through the instrument into the tank registering descent. The yaw string indicates slip and it does not require the pilot to look down at the instrument panel. It is made from a string that is taped to the outside of the cockpit. The compass is traditionally constructed, a magnet suspended in liquid, and indicates the direction the pilot is heading.

Figure 4 – From top left to bottom right airspeed indicator, altimeter, variometer, compass, and yaw string (red string on top of image):

variometer [http://upload.wikimedia.org/wikipedia/commons/b/bc/Cair-Xk10-vario.jpeg](http://upload.wikimedia.org/wikipedia/commons/b/bc/Cair-Xk10-vario.jpeg),
Most sailplanes are incapable of taking off themselves so they must be assisted. This is traditionally done in one of three ways; aerotow, winch, or autotow. Aerotow launching is done by attaching a tow rope from a tug plane, usually a single engine light aircraft, the slack is drawn out and then the tug plane takes off delivering the glider at the desired altitude and destination, usually near a place of rising air. Winch launches start with a long tow cable, usually 1000 – 1600 meters in length, which is reeled in fast enough that lift is generated by the gliders wings and the plane can take off. This method achieves a lower altitude for the glider and is usually not near a place of lift but it has the advantage of significantly lower cost. Autotowing involves a long runway, 3 km or more, and a shorter cable than winch launches, about 500m. Much like a child running with a kite the car accelerates hard and pulls the sailplane into the air achieving about 400m of altitude.

2.1.1 Glider Lift

In all gliders potential energy (height) is converted into horizontal motion by the interaction of lift generated by the wings of the aircraft and the downward pull of gravity. The losses in potential energy due to drag, in powered planes, is augmented by the engine which pulls the aircraft to high heights. In sailplanes this is not an option; the only sources of height gain are places of rising air. These places include thermals, ridge lift, and wave lift. Thermals are places of rising warm air created by the Sun heating up the ground. If the air is humid enough a cumulus cloud, the big fluffy low altitude clouds shown in Figure 5, will form at the top of the thermal. Glider pilots before launch will look for these forming clouds and ask their tug pilot to bring them near the clouds. Ridge lift is created when wind blows against a ridge forcing air to go up over the ridge. Thermals can also form on top of the ridge if the face of the ridge faces the sun. If a steady wind blows ridge lift can last all day. Mountain waves, shown in Figure 6, are created when wind blows over a mountain, these waves have been used to achieve the current height record (15,453m) and distance record (3,008km).
2.1.2 Glider Performance

Glider performance is measured by the glide ratio, or lift to drag ratio (L/D). L/D is defined as the amount of lift generated by the wings divided by the amount of drag created by the movement of the aircraft through the air. The glide ratio is also defined as the ratio of the sailplanes forward speed or distance covered to its rate of descent or height loss per unit time. High performance sailplanes achieve glide ratios of 60:1 (units of distance forward/units of distance descent) while standard class gliders are in the range of 40 – 45:1 and training gliders are in the range of 20 – 30:1.
2.2 GPS

The United States NAVSTAR GPS is the only fully operational GPS in the world. It is a space based radio navigation system that provides positioning, navigation, and timing services to military and civilian users on a continuous worldwide basis. (footnote [www.gps.gov](http://www.gps.gov)) GPS was originally intended for military use only but after an incident where Korean Airlines flight 007 was shot down by the U.S.S.R after it strayed into restricted air space President Ronald Reagan issued a directive making it available for civilian use. The system became fully operation on April 27, 1995 allowing civilians to be able to track their position to within 100m. The degraded resolution was purposely created until a Presidential mandate removed it in 2000, now the resolution can be as good as 20m.

The GPS consists of three segments; the Space Segment (SS), the Control Segment (CS), and the User Segment (US). The Space Segment consists of 24 positioning satellites and 6 correction satellites on six planes with four satellites per plane, centered on the earth, shown in Figure 7. The satellites have approximately a 55 degree inclination and are separated by a 60 degree right ascension of the ascending node. The orbits are arranged so that at least 6 satellites are always in line of sight, no more than 12 at ground level, and they complete two orbits every day. The Control Segment consists of installations that monitor flight paths in Hawaii, Kwajalein, Ascension Island, Diego Garcia, and Colorado Springs Colorado. The collected information is sent to Colorado Springs which then sends navigation updates and atomic clock synchronization, to the nanosecond, to the satellites. The User Segment is made up of the GPS receivers that people have in their cars, cell phones, airplanes, and many other places.
Figure 7 – The orbits of the GPS satellites around the Earth. The blue dot is a receiver and the dashed green lines indicate which satellites are visible to it, there are 12 visible in this position (http://upload.wikimedia.org/wikipedia/commons/9/9c/ConstellationGPS.gif).

The signal that the satellites transmit, for civilian use, is on a frequency of 1.57542 GHz\(^1\) and contains 37,500 bits sent at 50 bits/second requiring 12.5 minutes for a full transmission. The data signal is divided into 25 frames, each 1500 bits long (30 seconds per transmission), then subframes 300 bits long, then 10 words 30 bits long. The first word is the telemetry word (TLM) and contains information about the age of the ephemeris data. The ephemeris is a table of values that gives information on the position of astronomical objects in the sky at a given time. The next word is the Hard Over Word (HOW), it contains data on the time of last restart. The rest of the first subframe contains data about status and accuracy of the transmitting satellite and clock correction data. Subframes 2 and 3 contain more ephemeris data. Subframes 4 and 5 contains the almanac data. This data consists of orbit parameters of all the satellites, technical status and actual configuration, and identification numbers. Subframe 4 contains almanac data for satellites 1 through 24, the positioning satellites, and subframe 5 contains the almanac data for satellites 25 through 32, the correction satellites.

The signal information is received by a GPS receiver which then calculates the position by timing the signals and finding the distance to each satellite. Geometric trilateration, depicted in Figure 8, is used to combine these distances with the satellites’ locations to obtain position. Four satellites are need for trilateration, the first satellite (sphere A) only shows that the receiver is within range of it. The second (sphere B) shows that it is in a region of the two intersecting spheres, this is where sphere A and sphere B overlap. The third satellite (sphere C) creates an intersecting circle, illustrated by the blue circle in Figure 8, that gives two points where the receiver might be, a good receiver will attempt to give a

\(^1\) http://en.wikipedia.org/wiki/Global_Positioning_System
position based on other information such as one point is a few miles in the sky. A fourth intersecting sphere eliminates one of the two points and gives the exact position.

Figure 8 – This is an example of geometric trilateration with three spheres giving two possible points of location on the blue circle in the figure. Can you picture the fourth sphere?

(http://ixbtlabs.com/articles/gpssystem/pic2.gif)

2.3 Flight Data Systems

There are many GPS devices currently on the market, and many of those are made for handheld or automotive use. There are a few GPS systems designed for use in aeronautics, and a few of those are made specifically for sailplanes. A brief summary of currently available sailplane GPS systems, which are more often than not part of a Flight Computer, follows.

The most common, low end, GPS system/Flight Computer used is a PDA system that includes the PDA, a Bluetooth GPS receiver, an arm mounting to hold the PDA in view of the pilot, and software such as SoarPilot. The software is written for PalmOS or Windows Mobile. SoarPilot is a typical GPS navigation software for gliders and it provides a moving map along with a final glide screen which displays all of the required information (Figure 10). Battery life of one of the last models produced, the Tungsten E2 (Figure 11), is claimed to be very good, a few days with heavy use. A Bluetooth GPS receiver is also needed for this setup, a common one is the Belkin F8T051 (Figure 11). This is a 12 channel sensitive GPS receiver with a 10 hour battery life. The PDA is of $199 and the GPS receiver is available for around $90
making this system inexpensive in the sailplane Flight Computer market\(^2\). This system does not fit in the instrument panel and must be mounted with an arm to hold it in view of the pilot, see Figure 9. This takes up space which is at a premium in the tight cockpit of a sailplane and because the arm is generally permanently mounted to the instrument panel and is not portable.

Figure 9 – An arm mounted PDA system (Picture provided by F. Looft WPI).

\(^2\) [http://en.wikipedia.org/wiki/Tungsten_%28handheld%29#Tungsten_E2](http://en.wikipedia.org/wiki/Tungsten_%28handheld%29#Tungsten_E2)
A mid-range instrument panel soaring flight computer is the SN-10 by Ilec, shown in Figure 12. The Sn-10 boasts a laundry list of features including a variometer, an estimation of wind strength, database of airfields and turn points, final glide around turn points, integrated G-meter, and a thermal height band display. The software of this unit has pages that displays all of these features, the brochure that describes these features is in Appendix A. This unit does not come with a GPS receiver and one must be purchase separately. This instrument is mounted in the instrument panel and runs off of the power
provided by the sailplane. This device displays all of the required information and more but it is mounted in the instrument panel and is not portable. It carries a price tag of $3195 plus $188 for an installed Garmin GPS 16-HVS GPS receiver, from cumulus-soaring.com.

![Figure 12 – The Ilec SN-10](http://www.ilec-gmbh.com/sn10.htm).

A high-end in instrument panel soaring flight computer is the LX8000 by LX navigation, shown in Figure 13. This unit also has many features including a color display that changes backlight intensity according to light levels, uses a linux operating system, it is preloaded with map and airport databases, has an integrated flight recorder and data is downloadable through an SD card, real-time flight optimization, full flight tasks and statistics, integrated FLARM collision avoidance system, and a scrolling map. A full feature list from the products manual is provided in Appendix B. This device is mounted in the instrument panel so it is not portable and runs off of the sailplane’s battery using 430mA at 12v. It is priced at $5895 from cumulus-soaring.com.

![Figure 13 – The LX8000 showing the scrolling map (left) and one of the information screens (right)](http://www.lxnavigation.si/avionics/products/lx8000.cfm).
2.4 Summary

This chapter has provided an explanation of soaring, GPS, and an overview of products already on the market. Soaring involves moving the plane in three dimensions, using the instrument panel, and finding and exploiting sources of lift. GPS uses 32 satellites transmitting data to receivers that are controlled by a controlled station. The receiver then uses geometric trilateration to find location. A few products that are already on the market we discussed. None of these products fit exactly what will be specified in the next section.
Chapter 3

Project Statement and Goals

3.0 Introduction

In this section a description of the project is given. The project statement describes the qualitative goals of the product and the system requirements quantify these goals.

3.1 Project Statement

The purpose of this project is to create a portable GPS device intended for use in a sailplane. The device will be very low power, low profile, relatively inexpensive, and programmable. Also, it will be programmed to display bearing, altitude, speed, and final glide.

3.2 System Requirements

- Very low power: sufficient for 8 – 10 hours of continuous self contained battery operation.
- USB chargeable and programmable.
- Portable: about 6 inches long, 2 inches high, and 3 inches deep.
- < $200.
- Able to operate up to 18,000 feet.
- Able to operate between 0 to 50 degrees Celsius.
- Displays: bearing, altitude, speed, and final glide to home airport.
- Programmable Features:
  - Glider L/D
  - Home airport location
  - User defined functions

3.3 Summary

This section provided a list of requirements that the device must meet.
Chapter 4

System Design

4.0 Introduction

This section provides a system level design, a block diagram of the device, and a description of the parts to be used.

4.1 Block Diagram

A block diagram of the system to be built is shown in Figure 14. The figure illustrates the links between the modules required for this system. Each module in the diagram is necessary to the function of the system in order to accomplish the functionality described in section 3.2. All of the data that the device is to collect can be extrapolated from a GPS receiver as described in section 2.2. The data from the GPS then needs to be interpreted and displayed on a screen that the user can see. This can be accomplished by using a microcontroller, to decipher the GPS data, and an LCD display to show the information to the user. Some user input will be needed so that the user can specify some information such as the performance of the sailplane so that final glide can be calculated, buttons will be needed to achieve this. In order to power the device batteries will be needed and to accomplish the battery life specified in section 3.2 a voltage regulator will have to be employed so that as the batteries are drained the voltage remains constant. To charge the batteries over USB a small charging circuit will be needed in order to limit the charging current to a current that will not harm the batteries. The same USB will also be connected to the microcontroller to program it. Each of the afore said concepts are individual modules of the device shown in Figure 14, the interconnections between the modules were determined after a suitable part for each module was chosen as described in the next section.
Figure 14 – The Block Diagram. Input to the system is shown in normal squares, MCU (Microcontroller Unit) and GPS modules are in raised squares, the display module is shown in an octagon, power modules are in boxes, and recharging/JTAG is shown in rounded squares. The interfaces and power is indicated by the text along the connecting lines.
4.2 Description of Modules

4.2.1 User Display

This module will display the data processed by the MCU in an easy to read interface. Requirements specific to this module include the following:

- The display will need to be high contrast so that it is easy to read against the background and high light levels of the sky.
- It will need to operate at a low voltage, 3.3v or less, so that the system will be able to be rechargeable over USB and easily interfaced to the MCU.
- The display needs to have a low operating current to save battery life.
- It is desirable that the display have either a simple 8 bit parallel interface or a standard serial interface for ease of interfacing.
- The display must also work in the specified temperature range of 0 to 50 degrees Celsius.

The module will need to be large enough to display detailed information regarding glider performance, distance to home airport, and other fields. A display that would be adequate for data presentation is shown in Figure 15. The example includes all the required information that was stated in the system requirements (section 3.2) and the information fits in a 16 character long by 2 line format.

4.2.1.1 Detailed Data Display Formats

In the following description the “Δ” symbol implies a space while the # implies a digit. The types of information that need to be displayed include:

<table>
<thead>
<tr>
<th>Displayed Data</th>
<th>Data Range</th>
<th>Example Display</th>
<th>Chars required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance to home airport</td>
<td>00.0 to 99.9 M</td>
<td>DSTΔ#.##</td>
<td>8 characters</td>
</tr>
<tr>
<td>Current altitude</td>
<td>00000 to 99999 ft</td>
<td>ALTΔ####</td>
<td>9 characters</td>
</tr>
<tr>
<td>Bearing to home airport</td>
<td>000 to 359 degrees</td>
<td>BRΔ#####</td>
<td>6 characters</td>
</tr>
<tr>
<td>L/D</td>
<td>00 to 99</td>
<td>LDΔ##</td>
<td>5 characters</td>
</tr>
<tr>
<td>Air speed</td>
<td>000 to 150 kts</td>
<td>SΔ###</td>
<td>5 characters</td>
</tr>
<tr>
<td>Satellites</td>
<td>00 to 15</td>
<td>SATΔ#</td>
<td>6 characters</td>
</tr>
<tr>
<td>Final glide calculation</td>
<td>-9999 to +9999 ft</td>
<td>FGΔ±####</td>
<td>8 characters</td>
</tr>
</tbody>
</table>

Assuming that the following data would be grouped in various screen displays:

**Screen I**
- Alt, air speed
  9+5 character, plus spaces between values(16 chars total)
- L/D, satellites
  5+6 characters, plus spaces between values(14-16 chars total)

**Screen II**
- Distance
  8+6 character, plus spaces between values(16 chars total)
- Final glide
  8 characters (8 char total)
An LCD display that has all of these features and characteristics is shown in Figure 16. A List of its features follows:

- Model: ADM1602K-NSA-FBS/3.3v (Figure 16).
- 16 characters by 2 lines.
- HD44780 compatible controller which has libraries readily available for easy programming.
- 8 bit parallel interface.
- 3.3v @ 17.5mA.
- Operational temperature 0 – 50 degrees C
- Amber on Black for high contrast.
- Cost: $14.95


4.2.2 GPS

This module will receive GPS signals, interpret the signals, and output them as NEMA-0183 codes (Appendix D). Requirements specific to the GPS module include the following:

- The NEMA codes will be sent to the MCU over a 3.3v UART interface.
- The module will need at least 12 tracking channels as this is the max number of satellites that can be in line of sight on the ground at any one place.
- It will need an update rate of at least 1Hz because the sailplane may travel at over 100 miles per hour.
- It must operate at 3.3v or less so that the system can be USB chargable.
- The module will have to have a low operating current to save battery life.
- It will have to be small enough to fit in the size parameters given in section 3.2.

---

- A beneficial, but not required, functionality would be if the module used WAAS and other satellite based augmentation systems (Appendix C).
- The module will have to work in the 0 to 50 degree Celsius temperature range.

A GPS module that meets all of these requirements is shown in Figure 16. This module has many features given in the list below. The -161dB sensitivity is very sensitive, GPS is guaranteed a -130dB signal and with 10-20dB loss per wall this would give one to two walls of indoor GPS use with this receiver\(^4\). The integrated LNA and the multipath detection and suppression will help signal reception with the receiver being located under the plastic dome of the cockpit. Also, this module offers the added functionality of logging which may be used in future applications.

Feature List:

- Venus GPS Logger with SMA connector (Figure 16)
- Uses the VENUS634FLPx GPS chip
- 51 channel acquisition and 14 channel tracking
- Outputs NEMA-0183 codes
- 10Hz max update rate
- Integrated LNA
- Multipath detection and suppression
- 2.7-3.3v at 28mA tracking
- UART TTL 3.3v at 4800/9600/38400/115200 baud
- 32Mbits of flash
- -161 dB sensitivity
- Accurate to less than 2.5m
- Hot start 1 second
- Cold start 29 second
- Supports SBAS (WAAS, EGNOS, MSAS) (WAAS is assisted aviation GPS, see Appendix C)
- 1.25 x .75 inches
- Operational temperature -40 to 85 degrees C
- Cost: $59.95\(^5\)

---

\(^4\) The signal strength and loss per wall is taken from Frank Van Diggelen (Technical Director of GPS Systems, Broadcom) in a video ([http://media.wpi.edu/Academics/CWINS/June08/01_Hybrid_Positioning.asx](http://media.wpi.edu/Academics/CWINS/June08/01_Hybrid_Positioning.asx)) from WPI Invitational Workshop on Opportunistic RF Localization for Next Generation Wireless Devices ([http://www.cwins.wpi.edu/workshop08/program.html](http://www.cwins.wpi.edu/workshop08/program.html)).

4.2.3 Antenna

The GPS module will need an antenna to receive the GPS signals. Requirements specific to this module include the following:

- This module will need an SMA interface to connect to the GPS module.
- The antenna will not need gain because the application is in the sky where there is nothing obstructing the signal.

An antenna that meets the requirements is shown in Figure 17. The antenna does have gain and will require power but this will allow for future applications where gain may be needed. A List of features for this antenna follows:

- VTGPSIA-3 internal active antenna (Figure 17)
- 26dB gain
- 3.3v @ 12mA
- SMA interface to the Venus GPS module
- Cost: $11.95

---

4.2.4 Input

The user input interface will be three push buttons and one on/off switch. Requirements specific to this module include the following:

- These buttons will have to be big enough so that the user can easily push them with gloves on if it is cold outside.
- The interface to the MCU will be pull down; logical low will mean that button is pushed.

Buttons that meet these requirements are shown in Figure 18.

- 3 pull down push buttons (Figure 18)
- One on/off switch
- Cost: about $10
- ADD PICTURE (I only have one button and no switch. I will add the picture when I have all the parts)

4.2.5 MCU – Micro Controller Unit

The MCU will receive user input, NEMA codes from the GPS module, send the data to the LCD module to be displayed, and will be USB programmable. Requirements specific to this module include the following:

- 3.3v 9600 baud UART for the GPS module.
- 3 GPIO pins for the buttons.
- 8 GPIO pins for the 8 bit parallel interface to the LCD display.
- 4 wire JTAG for programming.
- It will need a large amount of flash program memory to hold user functions for operation.
- The module will also need enough RAM to hold the NEMA strings as they are processed and also run fast enough to process the NEMA strings as fast as they come over the UART.
- It will need to run at 3.3v or less so that the system can be USB chargeable and run at low current to save battery life.
- The module will have to run in the operational temperature of 0 to 50 degree Celsius.
An MCU that meets all of these requirements is shown in Figure 19. A list of features for this MCU follows:

- MSP430F5419 Microcontroller (Figure 19)
- Up to 25 MIPS
- 18Mhz
- 128Kb flash program memory
- 16Kb RAM
- 100 LQFP package
- 2 USCI (UART/IRDA/SPI/I2C)
- 87 GIPO
- 1.8-3.6v @ 165μA/MIPS active
- Cost: $3.65 | 1ku

![MSP430 MCU](http://en.wikipedia.org/wiki/File:MSP430_chipshot.jpg)

**4.2.6 USB**

The USB module will need to provide power so that the batteries can be rechargeable and will program the MCU with a four wire JTAG interface.

A USB module that has this functionality is shown in Figure 20. Power will be taken directly from the USB pins. The 4 wire JTAG is seen as the black connector to the target board in the figure. The target board will be removed and the JTAG will be connected the MCU module. A feature list follows:

- EZ430-F2013 development board with removable target board (Figure 20)
- USB connectivity
- 4 wire JTAG

---

7 [http://focus.ti.com/docs/prod/folders/print/msp430f5419.html](http://focus.ti.com/docs/prod/folders/print/msp430f5419.html)
• The front half with the USB connector will provide power, to recharging circuitry, and a JTAG for the MCU
  o The target board will not be used

Figure 20 – The EZ430-F2013 development board (http://focus.ti.com/graphics/tool/ez430-f2013.jpg)

4.2.7 Batteries
The batteries will have to provide 3.3v to all the other components in the device. The total theoretical power consumption is the sum of the current of all the components, which is 61.625mA. The batteries will have to provide enough power to run the device for 8 – 10 hours, this means 61.625mA x 10 hours = 616.25mAh.

A picture of a rechargeable battery that with two more will provide the required power to run the device is shown in Figure 21. A feature list of the battery follows:

• Greenbatteries brand NiMH AAA 850mAh rechargeable battery

Figure 21 - Greenbatteries brand NiMH AAA 850mAh rechargeable battery (http://www.greenbatteries.com/grbrniaa85b.html)
4.2.7 Power Circuitry

The power circuit will have to provide 3.3v at 61.625mA over the entire charge of the batteries. Typical nickel metal hydride batteries are fully charged at about 1.35v and are fully discharged at about 0.9v. A few common batteries, in a two cell holder, were discharged from a full charge through a 10 ohm resistor. The discharge curves were recorded and are displayed in Figures 22. As the figures show the Duracell has the most stable discharge at an almost constant voltage of 2.4v. The curve also shows that the power circuit will have to be able to maintain 3.3v over the discharge from about 2.7v down to 0.9v.

An integrated circuit that will accomplish this is the TI TPS61016 High-Efficiency 1-Cell and 2-Cell Boost Converter shown in Figure 23. A feature list follows:

- Integrated Synchronous Rectifier for Highest Power Conversion Efficiency (>95%)
- Start-Up Into Full Load With Supply Voltages as Low as 0.9 V, Operating Down to 0.8 V
- 200-mA Output Current From 0.9-V Supply
- Powesave-Mode for Improved Efficiency at Low Output Currents
- Autodischarge Allows to Discharge Output Capacitor During Shutdown
- Device Quiescent Current Less Than 50 µA
- Ease-of-Use Through Isolation of Load From Battery During Shutdown of Converter
- Integrated Antiringing Switch Across Inductor
- Micro-Small 10-Pin MSOP or 3 mm x 3 mm QFN Package

Figure 23 – Recorded discharge curves of a Duracell, Energizer, and Radio Shack NiMH batteries

An integrated circuit that will accomplish this is the TI TPS61016 High – Efficiency 1-Cell and 2-Cell Boost Converter shown in Figure 23. A feature list follows:
4.2.8 Charging Circuit

The Charging Circuit will have to be able to charge nickel metal hydride batteries from dead, about 0.8v, to full charge, about 2.7v. The voltage source will be a USB cable from a computer which will supply 5v and up to 500mA. The circuit will have to charge the batteries at 0.1 Charge, 0.1 * the battery mAh rating, in order to not over charge the batteries and shorten their life span. The Circuit will have to hold a voltage above the full charge voltage of the batteries and provide about 220mA in order to charge the batteries at .1Charge.

A circuit that will accomplish this is shown in Figure 24. The VCC represents the USB supply of 5v, D1 is diode protection to make sure current does not flow into the USB, R13 is a current limiting resistor so that the batteries will not charge too fast, and D2 is a zener diode intended to keep the voltage at point 1 at about 3v.
4.3 Summary
This section provided a system level design, a block diagram, a description of what each of the modules within the diagram have to accomplish, and a part that would meet the requirements of the module descriptions.
Chapter 5

Circuit and Software Design

5.0 Introduction

This section provides a circuit level description of how the final design works and also a description of the software that demonstrates the functionality of the entire device. The development of the schematic, and the device, was done modularly. Once one module worked development moved onto the next logical module. Development started with the MSP430 then the LCD display followed by the buttons, GPS, the power circuit, the charging circuit, a UART to USB module, and finally a JTAG. Each and every module was prototyped before a final design was created. Once the final design and schematic was created a printed circuit board (PCB) was designed and made. Finally a project box was created to fit the device into for protection during testing.

5.1 The Device

A picture of the prototype and the final design is shown in Figures 25 and Figure 26.

![Figure 25 – The final working prototype](image-url)
Figure 26 – The working device (top) followed by the Copper Top layer of the populated PCB (middle) and then the Copper Bottom layer of the PCB (bottom)
5.2 Circuit Diagram
A schematic of the entire device is shown in Figure 27. This figure shows all of the components, integrated circuits, and connections that make up the device. The schematic was created in Multisim.
Figure 27 – The Device schematic, the red wires are interconnections, the green boxes are IC’s, and the blue symbols are components.
5.3 Description of the Circuit

5.3.1 MCU – MSP430F47196
The MCU used in the design is the MSP430F47196, this is not the desired MCU, the MSP430F5419, because at the time the MSP430F5419 was on a 3-5 month backorder and this did not fit into the time schedule available for prototyping. The replacement MCU, the MSP430F47196, is almost identical to the desired one with the exception of a slower clock speed and the JTAG (the JTAG will be discussed in a subsequent section). This microcontroller was chosen over other processors, such as a PIC or an ARM, because the inventor is familiar with the MSP430 series, as opposed to a PIC, and an ARM would be too expensive, overly complex, and unnecessary for the more simplistic nature of the device. The MSP-TS430PZ100A target board, seen in Figure 28, was purchased to prototype the MCU. This target board makes all 100 pins easily available for prototyping, it has a JTAG connector, and it has a solderless socket for the MSP430. After inserting an MSP430, soldering in a 32.768 kHz watch crystal into the pins marks Q1 (found just to the left of the socket in Figure 28), connecting the JTAG, and downloading a test program the MSP430 worked on the first time.

![Figure 28 - MSP-TS430PZ100A target board](http://focus.ti.com/graphics/tool/TS430PZ100A.jpg)

5.3.2 User Display
The LCD display that is used in the final design is the ADM1602K-NSW-FBS/3.3v which is a white on black display shown in Figure 29. This display was chosen over the original ADM1602K-NSA-FBS/3.3v amber
on black display because it was found to have a better contrast in the direct sunlight that a sailplane cockpit is found to have.

![Image of LCD display](http://static.sparkfun.com/images/products/serialLCD3.3whiteonblack.JPG)

Figure 29 – The ADM1602K-NSW-FBS/3.3v white on black LCD display

The first step in prototyping the LCD was to solder headers to the MSP430 target board and then create a harness to connect the target board to the display; this can be seen in Figure 30.

![Image of harness](image)

Figure 30 – The harness that connect the MSP target board to the display
The data lines of the display, pins 4-14 (RS, R/W, E, DB0 – DB7) on the display, are connected to port 10.3 – 10.0 and port 9.7 – 9.0 which are pins 23 to 30 on the MSP30. The 8 bit parallel data lines of the LCD display are connected backwards of convention, i.e. little endian, which made easier programming by the inventor but makes for confusing programming for everyone else (this will be discussed in a subsequent section). The corrected version of this connection, i.e. big endian, in schematic form can be seen in Figure 31.

Figure 31 – The LCD display connection to the MSP430
The remaining LCD pins are connected as follows. Vss is grounded, Vdd is connected to 3.3v, V0 controls the contrast ratio and is connected through a potentiometer (X1 in Figure 31), LED+ is power for the backlight and is connected through a 100 ohm resistor to 3.3v, and LED- is the ground for the backlight and it is grounded. All of these connections are shown in Figure 31.

### 5.3.3 Buttons

The user input is taken through three push buttons. The MSP430 is connected to the NO terminal on the buttons on port 1.7 – 1.5 which are pins 86, 85, and 84. The buttons are active low so the NO pin is pulled high through 100Kohm resistors, and the C pin is grounded to create the low needed. This circuit is shown in Figure 32.

![Figure 32 – The Buttons and their connections](image)

### 5.3.4 GPS

The GPS module chosen in section 4.2.2 and shown in Figure 16 was used. The connections that had to be made were 3.3v, ground, TX, and RX. The TX and RX are for the UART to the MSP430 over which the NEMA strings are sent. Figure 33 shows the prototype of the GPS, and Figure 34 shows the relevant portion of the schematic. Originally the TX and RX pins were connected backwards which caused system instability. Once this was fixed and the TX RX pins were connected to pins 78 and 79, UART RX TX on the MSP430, respectively the only problems encountered were in software (this will be explained in a subsequent section).
5.3.5 Power Circuit

The original TI TPS61016 High – Efficiency 1-Cell and 2-Cell Boost Converter from section 4.2.7 and shown in Figure 23 also shown prototyped in Figure 35 and in the schematic of Figure 36 was used in the final design.
As can be seen in Figure 23 the boost converter requires a few external components. The values of these components were calculated using equations from the data sheet as follows.

\[
R_2 = 500K\Omega \text{ (recommended)}
\]

\[
R_1 = 500K\Omega \left( \frac{V_{BAT}}{500mV} - 1 \right) = 500K\Omega \left( \frac{2.2v}{500mV} - 1 \right) = 1.7M\Omega
\]

These two resistors form a voltage divider the voltage they create, 2.2v, is used to indicate a low battery warning.
\[ I_L = I_{out} \times \frac{V_o}{V_{BAT} \times 0.8} = 100mA \times \frac{3.3V}{1.6V \times 0.8} = 257.8 mA \]

\[ L = \frac{1.6v(3.3v - 1.6v)}{.2(257.8mA)(500k\Omega)(3.3v)} = 32 \mu H \]

These two equations determine the size of the inductor to be used. The first equation is the current through the inductor, \( I_{out} \) is the estimated output current, \( V_o \) is the output voltage, \( V_{BAT} \) is the lowest battery voltage(0.8v *2 batteries). The second equation calculates the inductor size needed for the specifics of this application based on the lowest input voltage, the output voltage, and the inductor current.

\[ C_{in} = \frac{I_{out} \times (V_{out} - V_{BAT})}{f \times \Delta V \times V_{out}} = \frac{100mA \times (3.3v - 1.6v)}{500kHz \times 15mV \times 3.3v} = 7\mu F \]

\( C_{in} \) is used to remove input ripple.

\( C_{out} = \) recommended at 22\( \mu F \)

\( C_{C2} = L \) in nF that is \( L = 32\mu H \) therefore \( C_{C2} = 32nF \).

\[ R_C = \frac{1mS}{C_{C2}} = \frac{1mS}{32nF} = 31250\Omega \]

\[ C_{C1} = \frac{C_{out} \times .2}{R_C} = \frac{22\mu F \times .2}{31250\Omega} = 141pF \]

\( C_{C1}, C_{C2}, \) and \( R_C \) form a control loop that regulates the charge and discharge of the inductor.

\( R_3 = 1M\Omega \) (recommended) it is used as a pull up resistor for the low battery indicator.

The values calculated are ideal values, the values used were standard values that were available for purchase. These values and all the information pertaining to the actual parts used can be found in the Bill of Material in Appendix E.

**5.3.6 The Charging Circuit**

The charging circuit of section 4.2.8 and Figure 24 was used in the final design with a modification for power concerns. This circuit had to take into account being able to run the device without batteries, run the device and charge the batteries, or just charge the batteries. The circuit that can handle all of these situations is shown in Figure 37 and the prototype is in Figure 38. The Diode D1 is so that current will not flow in the direction towards the USB connector, the two 15 ohm resistors in parallel accomplish the needed 7.5 ohms but with greater power dissipation capability and the zener diode holds the input voltage to the batteries or the system to 3v. The 7.5 ohms was chosen to limit the current to about 500mA, and the 3 volt zener diode was chosen to keep the voltage higher than the battery voltage, so that they will charge, but lower than the 3.6 volt maximum input to the power circuit.
Figure 37 – The battery charging circuit. The input is from the USB and the output is the positive terminal of the battery pack.

Figure 38 – The prototype of the charging circuit

5.3.7 UART to USB

A UART to USB module was added next. This module allows communication from the GPS directly to a PC and was added for the future functionality of being able to retrieve logged GPS data. The schematic of this chip is seen in Figure 39 and the prototype is in Figure 40. The Chip is made by FTDI and its number is FT232RL. The prototype was purchased on a PCB from Sparkfun.com and the implementation was very simple, just connecting the TX RX from the GPs and the PC started receiving NEMA strings. The implementation was almost as easy; a USB connector, a fuse, and a couple filter capacitors were added and the chip was working in the design.
5.3.8 JTAG

The JTAG connector was the last part of the prototype to be added. This was because the target board already had a JTAG onboard. The circuit for the JTAG was taken from the target board and was considered to be simple enough to not prototype. The schematic of the JTAG is found in Figure 41.
5.4 Printed Circuit Board

Once the prototype was completed and the schematic received a final revision a printed circuit board was designed. This started by searching through Mouser.com and Digikey.com for all the surface mount components needed, a list of all these parts can be found in the Bill of Materials in Appendix E. Once all the parts were found the footprints of those parts were entered into the schematic in Multisim. Footprints that were not found in the Multisim database, such as the mini USB connector, had to be created.

After all the parts were picked and the footprints entered the export to Ultiboard option in Multisim was utilized, Ultiboard is the National Instruments PCB creator. Once in Ultiboard the size of the PCB was entered, 36mm by 80mm the same size as the LCD display, and the drill holes in the corners of the board were made, also matching the pattern of the LCD display. Then the components were placed and all of the wires were routed by hand.

Special care had to be taken for the power circuit as it had special grounding and component placement requirements that were stated in the data sheet. All of the grounds for the power circuit had to be run to pin 4 of the TPS61016 chip and then that pin was grounded to the ground plane.

The placement of the rest of the components was done so that they were as close together as possible, but not too close, keeping in mind that they would have to be hand placed and soldered onto the board later. The board is only two layers both of which were made ground planes to minimize noise. The PCB as it appears in Ultiboard and the fabricated PCB is shown in Figure 42 and Figure 42 - B.

Once the PCB was finalized and doubled checked for accuracy it was sent out to APCircuits.com to be made, the turnaround time was 4 business days. The unpopulated PCB is shown in Figure 43. When the PCB and the parts came in a hot air rework station (seen in Figure 43), a device which consists of a soldering iron and a kind of small hair dryer like wand that blows air up to a temperature of 480 degrees Celsius, was purchased to make the population of the board easier. To populate the PCB a dab of solder paste, a putty like substance that holds parts down as they are soldered, was placed on each solder pad and the corresponding part to the solder pad was placed. After all the parts were placed the hot air
rework station was used to blow every part with hot air until the parts were soldered to the board. The only parts that caused trouble were the IC’s, after some practice a bead of solder was placed across all the pads of an IC and the entire part was heated so that it floated and centered itself atop the mounds of molten solder. When all the solder pads were heated at once the bead of solder paste that ran across all the pads disconnected itself from the other pads and only soldered one pin to one pad. The populated PCB is shown in Figure 26.

The populated PCB was then placed behind the LCD to create the picture shown in Figure 26. There were no mistakes in the design of the PCB but one change could be made so that the data lines from the LCD to the MSP430 are big endian instead of little endian.

Figure 42 – The PCB as it appears in Ultiboard. The top is the Copper top layer and the bottom is the Copper bottom layer. Actual size is 80mm x 36mm.
Figure 42 – B – The Fabricated unpopulated PCB, the Copper top layer is shown in the top picture and the Copper Bottom layer is shown in the bottom picture. The populated PCB is shown in Figure 26.

Figure 43 – A hot air rework station used to heat the PCB when populating it. 
5.5 Software

The software was written in conjunction with the development of the prototype. The entire program can be found in Appendix F. First code was written to display something on the LCD, then expanded for the buttons, and then the GPS. The first hurdle was the display. After much research and pouring over the datasheet for hours an initialization sequence was programmed. This starts with telling the LCD controller that the display attached to it is 8bit, 2 rows, and 5x7 dots per segment, then the display is turned off, the cursor off, and no blink for the cursor, then the display is cleared, after that the display is turned back on with the cursor off, and finally the controller is told to auto increment and use a shift cursor. After this start up sequence the display will then accept write commands, data written to the RAM of the controller will now be displayed. In order to write data to the controller the data must be available on the data line for about 10ms, then the Enable line must be toggled high and low for another 10ms. After this the data written to the controller will appear on the display. The timing was accomplished by making the MCU count using the swDelay(100); routine. The displaying of character was eventually streamlined by using the dispString, nextLine, clearLCD, and setToHome functions. These functions will display a string stored in a variable, move the cursor to the second line of the display, clear the screen, and set the cursor to the home position.

The next task was to receive NEMA strings from the GPS. The first attempt ended in the TX and RX being wired backwards. After this was fixed the information that was being received from the GPS was displayed directly onto the display so that it could be checked for accuracy. The characters that the LCD displayed were gibberish, knowing that the code for the display worked other causes for the problem were looked into. After looking into many options the problem was found to be that the baud rate generator was not accurate enough. The system clock was running at 32.768kHz, which divided by the 9600 desired baud rate turns into a 3.41 clock divider. Since the clock can only be divided by whole numbers to generate the baud rate, 32.768kHz divided by 3 gives a baud rate of 10922.7 which is too far off. Code was then written to multiply the clock rate up to 2.45MHz, this code can be found in the beginning of the main. 2.45MHz divided by 9600 gives a 255.208 divider, 2.45MHz divided by 255 gives a baud rate of 9607.84 this is very close to the desired baud rate and fixed the problem.

Now that the GPS and the display worked a 23 state, state machine was written to handle the input of the NEMA strings. The state machine examines the pattern of letters and numbers of each string and pulls out the desired information from each NEMA string. The information is then stored and displayed according to which screen the user currently has on the display.

The interrupt service routine for the buttons was then written and tested. The buttons work as up, down, and enter from left to right. Once this was known to work a test program was written.

This test program starts by waiting for the GPS to pick up at least 4 satellite signals. Once this is accomplished the device will ask the user is the current location is “home,” home is presumably the beginning of the runway where the pilot will land. The next question the device asks the pilot is is L/D, this can be entered but using the up or down button to get the correct number and then the enter button is pressed advancing the program. the next question is the desired final glide altitude, this number increments or decrements in hundreds of feet by pressing the up or down button, again enter
advances the program. After this information is entered the device will start displaying information
gathered from the GPS in one of three formats or screens. The first screen will display the current
bearing, speed and final glide. The second screen will display the bearing, speed, altitude and final glide.
The third screen will display just the time and the final glide. Final glide is calculated, in the fnlGld() function, using the haversine formula for calculating the distance between two latitude and longitude coordinates. The haversine formula is not the most accurate formula, but it was found to be only off by less than 4 feet over 6 miles. The haversive formula is as follows:

\[ R = \text{earth’s radius (mean radius} = 6,371\text{km)} \]
\[ \Delta \text{lat} = \text{lat}_2 – \text{lat}_1 \]
\[ \Delta \text{long} = \text{long}_2 – \text{long}_1 \]
\[ a = \sin^2(\Delta \text{lat}/2) + \cos(\text{lat}_1).\cos(\text{lat}_2).\sin^2(\Delta \text{long}/2) \]
\[ c = 2 * \arctan(\sqrt{a}, \sqrt{1-a}) \]
\[ d = R * c \]

After the distance to home is calculated then that number is divided by the L/D, this gives the required altitude needed to get home. Then the current altitude is subtracted from the altitude at the home location and then the required altitude to get home is subtracted from that giving the final glide number.

This software works and demonstrates the entire functionality of the device but there is currently one bug, sometimes when moving between screens some of the data is not completely cleared from the previous screen, and the available screens are not every useful to a glider pilot. Theses shortcoming could easily be fixed if more time was available to work on the project.

5.6 Summary

This section provided a description of how the device was created and how it works. It described the development of the prototype, how the prototype moved to a PCB, how the PCB was populated, and how the software was created and functions.
Chapter 6  
Testing and Evaluation

6.0 Introduction

This section will describe how the device was tested against the requirements listed in section 3.2.

6.1 Testing

The time and weather during which the device was finished, February 2010, did not allow for a test run in a sailplane so a car and a hill were used instead. The device was driven out to Lake Quinsigamond, on the Route 9 bridge crossing over the lake. At the traffic light facing up the hill, towards Worcester, the device was turned on and the home location was set. Then the device was driven up Route 9 to the top of Green Hill in Worcester. The display, besides reading the correct Greenwich Mean Time, showed a final glide of 243ft. This number was then checked against Google Earth. Google Earth found the distance between the two points to be 1.1miles, the elevation at the home location to be 384ft, and the elevation at the top of the hill to be 774ft. This can be seen in Figure 44. The final glide from Goggle Earth is calculated as follows: 1.1 miles * 5280ft = 5808ft divided by the L/D, 42, = 138ft of altitude required by the sailplane to reach home from the top of Green Hill. Next the correction for the altitude above sea level, 744ft at the top of the hill - 384ft at the home location - 138ft needed to get home = 222ft. This error seems large but when calculating the distances and elevations on Google Earth the pictures used are flat pictures put onto a curved Earth which accounts for about +/- 10ft over the distance used in this calculation. The final glide calculated by the device and the one from Google Earth are close enough to consider the device to be correct.

Next the system requirements were tested against those listed in section 3.2. The requirement list is copied below followed by the means of which the requirement was tested.

- Very low power: sufficient for 8 – 10 hours of continuous self contained battery operation.
  - The device was left on over night, after ten hours the device was still operating.
- USB chargeable and programmable.
  - The charging circuit was tested after the battery life test. The device was plugged into a USB port overnight, ten hours later the battery voltage was tested to be 2.7v which is a full charge according to Figure 23.
- Portable: about 6 inches long, 2 inches high, and 3 inches deep.
  - The project box that was created to hold the device measures 3.5” wide, 2” tall, and 1.3” deep.
• < $200.
  o The total cost for one device, ordered from distributors in single quantities, is the total of the prices found in the Bill of Material in Appendix E. This total is $151.22.

• Able to operate up to 18,000 feet.
  o Without being able to test the device in a sailplane and having no access to a pressure chamber this requirement was not tested but no components that are affected by altitude, such as electrolytic capacitors, were used so on paper the device should work at high altitude.

• Able to operate between 0 to 50 degrees Celsius.
  o The low range of the temperature requirement was tested by leaving the device in a freezer for ten minutes; the result was that the device still worked but the response time of the display increased. The top end if the temperature range was tested by placing the device into an oven; the result was that at 50 degrees Celsius the device still worked.

• Displays: bearing, altitude, speed, and final glide to home airport.
  o The information listed in this requirement is extrapolated from the GPS and is displayed on the LCD in the manner described in section 5.5. The bearing was tested in a similar manner to that of the final glide. The bearing while driving up to Green Hill was recorded and checked against that given by Google Earth, it was found to be within +/- 3 degrees. The speed was checked against the speedometer of the vehicle, also while driving up to Green Hill, this was found to be within +/- 2 miles per hour. The altitude and final glide was tested in the manner described in the beginning of this section.

• Programmable Features: Glider L/D, Home airport location, and User defined functions.
  o The glider L/D and the location of the home airport are asked for at device startup as described in section 5.5. User defined functions can be programmed by the user using the JTAG interface.
Figure 44 – Google Earth check of the final glide given by the device. The left red dot is the first parking space behind the baseball field of Worcester Vocational High School, the right red dot is the middle first position for a car at the intersection of route 9 and Lake Ave.

6.2 Summary

This section described how the device was tested against the requirements listed in section 3.2.
Chapter 7

Summary and Conclusions

7.0 Introduction

This section will provide a summary and conclusions about the project including what worked, what should have been done differently, and recommendations for next time.

7.1 Conclusions

The device works entirely and every testable requirement was met, the device can be seen in Figure 45. Only a few things should have been done differently. The data lines from the MSP430 to the LCD display should have been wired to be big endian. The charging circuit could use a schottky diode placed just before the batteries to prevent the draining of the batteries by the zener diode. More thought should have been put into the packaging, the design of the PCB was to make it as small as possible but the orientation of the GPS does not allow easy incorporation of the battery pack. To solve this either the orientation of the GPS to put it on the same side as all the other components, leaving the back of the PCB flat for the battery pack, or a flatter battery needs to be used, such as a lithium ion battery used in cell phones. The design of the PCB could not be any smaller but more though should be put into a box that this device could be sold in.

A recommendation for the next model would be to use a slightly larger touch screen display. A display too large would defeat the purpose of this project, to create a small portable sailplane computer; it would be too tall for the pilot to see over if it is put in the desired place the top of the instrument panel. A display just a little larger would allow for more information to be displayed increasing the usefulness of the device and the touch screen would allow for the mechanical buttons to be replaced making the device smaller and allowing for more buttons on the screen. A larger screen would also allow for a larger PCB to be designed, this would increase space for more components, more sensors could be added to allow for more data gathering.
Figure 45 – The device in a project box, the first picture is the front, the second is the top, and the last picture is the back.
7.2 Summary

This section provided a summary and conclusions about the project including what worked, what should have been done differently, and recommendations for next time.
Appendix A – Example of a Glider Flight Computer

This appendix provides an overview of a popular flight computer used by many glider pilots. The material below is copied with a few editing changes from:
http://www.ilec-gmbh.com/brochures/sn10pe09.htm where complete details on the unit can be found.

ILEC SN10 Flight Computer

The ILEC SN10 combines the latest Computer, Display and Friendly User Interface Technology with the ILEC’s renowned Variometer and Sensors. Features of this System include;

- World-famous ILEC Variometer Response
- Automatic Wind calculation (direction and speed)
- GPS input from your GPS or Data-Logger
- Easy Task Problem Solving (Speed-Only or POST)
- Built-in Database of airfields and turning points
- Easy Final glides around turnpoints
- Flight Recorder, optional with comprehensive Flight Analysis Software
- Integrated g-meter
- Exclusive Thermal Height-Band Display (picture)
- Instant on screen Help for all information and settings
- Integral local-area network for future options
- RS-232 interface for easy data transfer to/from PC or GPS
- Easy Software updates using your PC (no factory service require)
- All ILEC products come with a 2years warranty

With a built-in local area network and a powerful processor, the SN10 is expandable with external instruments, e.g. compass (in development), or other components to be announced in the future. Options include two-seater display, remote control units and 57mm circular meter for the vario indication.

The Most Important Features of the SN10

During the development of the SN10 every effort was made to make the instrument very easy, almost intuitive, to use and to learn. The following text pursues two different aims: first it will show the future customer many of the advantages the SN10 offers, and second, it will demonstrate the user-friendly operation.

The pages available in Flight-Mode of the SN10 will be shown in the following. Additional pages available in Setup-Mode concern the settings of the SN10 (e.g. the indication of the different installed units, audio settings); these are shown in the SN10 Manual.
**Simple Final Glide Page:**

Rapidly spinning the Page knob clockwise displays the Simple Final Glide Page. This contains all essential information concerning a final glide to an airfield or over an obstacle, but does not use the navigation and task planning features of the SN10. For the final glide of a task including final glide around turnpoints, the Status Page should be used.

The bottom line shows the distance to the finish (out) and the total energy height above (+++) or below (---) glideslope. When the glider symbol in the middle of the page is above the dashed line in the page center you are above glideslope.

Setting the Cursor on the field Fin and pressing the Help button shows the screen displayed beside, as an example of using the Help function.

**The Task Page:**

At the left side of the page the start and turn points are shown. The destination is displayed at the top of the right side. Enter a waypoint simply by placing the Cursor at the desired position and turning the Value knob until the desired waypoint is displayed from the site library. For every region a suitable library is loaded into the SN10.

The SN10 calculates the estimated time of flight (ETF). The On Screen Help for this field is displayed beside. ETF takes into account the wind (from the Status Page) and the MacCready setting (MC) as well as the other settings concerning the glider. Also displayed is the time remaining for the task, Rema. This could be the time remaining for the POST or Speed Only as well as the end of the thermals. Dis shows the distance remaining.

The glider symbol again indicates whether you are above or below glide slope to complete the task.

**Height Band Graph:**

High average speeds require not only excellent routing and locating of thermals, but also the choice of the best thermal height band and leaving thermals at the optimum time. This page displays graphically the last three thermals, and the pilot can easy detect the height band offering the best rate of climb.

To choose the proper MacCready setting (MC) we show on this page not only the average climb rate (Avg), but also the thermal average (TAv). TAv is the climb rate over the whole
thermal since switching from the cruise into the climb mode. The thermal average is normally a good estimate for the MC setting.

The glider symbol at left shows your current altitude with respect to the thermal heightband.

The Status-Page:

This page shows everything necessary in cruise, including all navigation and final glide information. Pilots get quickly used to the layout of this page and don’t want to miss any of the information displayed!

The left column starts with the 20-second averager (Avg). Below is the name of the next turnpoint followed by the distance (Out) to the turnpoint and the distance left or right of the courseline. Bearing to the turnpoint is displayed with an arrow showing the direction which you should turn in order to track towards the turnpoint. Below Q, your current ground-track as received from the GPS is displayed. The On Screen Help for this field is shown below.

At the right side the MacCready (MC), Water ballast, Bugs influence on the glider polar, and the estimated Wind (direction and magnitude) are set.

Below the altitude, Fin indicates the altitude necessary at the destination air field (field elevation + reserve). Finally, at the bottom right, is the altitude you are currently above or below glideslope to complete the task. --- indicates that you are below, +++ says that you are above. This surplus or deficit altitude is displayed as total energy height(!) and is therefore independent on pull or push actions! The glider symbol in the center of the page gives the glideslope indication graphically, it begins to rise if you are less than 130 m below glide slope.

You can see immediately how far you are above or below glideslope! No extra work is required for final glides around a turnpoint.

If you use your SN10 without GPS-coupling, this page is slightly different, to better support navigation and wind calculation in this mode of operation.

SN10 Wind Page:

If a GPS receiver is connected to the SN10, the system calculates automatically the current Wind (velocity and direction). Certain indicates the reliability of the calculated value and Headwind shows the current headwind.

The five lines below are giving the wind in the different flight levels. The first column shows the altitude, the second the wind direction and velocity. And the last indicates how old the last wind calculation in this layer was.
The arrow at the upper right gives the wind direction with respect to the flight direction (heading). This is very helpful at ridge soaring and searching for thermals.

Specially for the contest: Assistance before going through the start gate and during time limited tasks as Speed-Only or POST.

**the L/D Page:**

The L/D Page shows changes of glider performance data with changing MacCready setting (MC) and Water ballast.

For the given MC value and water ballast, L/D at normal cruise, the estimated average speed (Est speed) and the MC speed which must be flown between the thermals (S to fly) are shown.

The bottom row again shows the altitude below (---) or above (+++), the task glide slope.

**Technical Data:**

- **Power Supply:** 9 - 15 DCV
- **Current (volume off / full):** ca. 100 / 150 mA at 12 V (without GPS)
- **Housing, Main Unit:** ca. 250 / 300 mA at 12 V (with GPS)
- **Temperature Range:** ø 80 mm, 150 mm long (without adapter)
- **Weight:** -20 to +70 °C
  ca. 0.7 Kg
Appendix B - Example of a Glider Flight Computer

This appendix provides an overview of an expensive but popular flight computer used by many glider pilots. The material below is a list of features copied from the manual of the flight computer. The entire manual can be found at:

3.1 LX8000 at a glance
The instrument consists of two units, the LX8000 digital unit and the LX8000 vario unit. Inside LX8000 digital unit an integral 16 channel GPS receiver and high brightness color display with 320x240 pixels are fitted. For user friendly data exchange an integrated SD card or USB interface is used. The LX8000 has, as an option, a built in flight recorder according to the last IGC specification for all flights. Optionally FLARM collision avoidance system is integrated in LX8000 digital unit.
The 57 mm diameter (2 1/4”) LX8000 vario unit is a modern designed vario unit with its own micro processor. The unit communicates with the LX8000 digital unit over the RS 485 system bus. Optionally, additional LX digital vario indicators and a wide range of interface devices can be daisy chained using the RS 485 bus.

3.1.1 Display unit features
• Extremely bright 3.5” (8.9cm) color display readable in all sunlight conditions with backlight automatically adapted using ambient light sensor (ALS).
• Using Linux operating system (not CE Windows) ensures fast and stable operation of firmware.
• 6 push buttons and 4 rotary switches are used for input which compromise well known LX user interface. Remote stick is available optionally for more comfort.
• Preloaded with worldwide terrain maps, airspace and airport databases.
• Unlimited number of waypoints.
• Unlimited number of tasks (with assigned area support).
• Comprehensive flight and task statistics.
• Display of nearest airports and out landing fields.
• Unlimited number of pilots/profiles.
• Integrated flight recorder according to high-level IGC specification.
• Real-time flight optimization according to FAI and OLC rules.
• Flights stored in IGC format are downloadable using integrated SD Card.
LX8000 version 2.1 June 2009
Page 10 of 97
• Flight recorder functions include integral pressure transducer based on 1013mbar level for altitude recording, engine noise level sensor, memory to store more than 1000 hours of flights and digital and mechanically security device to ensure high level of security.
• Integrated FLARM collision avoidance system with graphic, sound and voice (optional) presentation.

3.1.2 Vario unit features
• Vario with many custom audio settings such as netto, relative (super netto) and average
• Smart vario
• Speed command
• Final glide calculator
• TE compensation is selectable for either pneumatic TE tube, or electronic TE

3.1.3 Interfaces
• RS232 interface has NMEA output, for external devices.
• SD Card interface
• USB for data transfer using USB memory stick.
• IGC interface to connect and power Colibri or LX 20. The connector pin out corresponds to the IGC standard.

3.1.4 Options
3.1.4.1 Internal options
If ordered, integrated Flarm electronics can be built in to the LX8000 digital unit. All necessary connectors are available on the rear side of the unit (Flarm external indicator, Flarm antenna), which guarantees minimum the same comfort like using of original Flarm devices. It is very important to point out that the whole system uses only one GPS receiver and therefore offers a low power solution. One bicolor LED Flarm External display is an apart of delivery, extension to second seat is possible.

3.1.4.2 External options
By using a RS485 bus system a wide range of optional interfaces can be easily connected to the basic configuration, without any significant installation works. The LX 8000 bus system can be extended easily by use of RS485 splitting units, which allow plug and play connection of optional devices. Following units can be connected to LX 8000 bus system:
• Second seat device (LX7007D or LX8000D). The unit installed in the rear seat of the glider, is powered and receives all necessary data from the main unit. The communication between both units is exclusively via RS485 bus system.
• Remote control. An extremely ergonomic leather coated handle which includes 8 push buttons to operate LX 8000 and also two additional buttons with open wires. These two buttons can be used for instance as PTT for radio and SC/Vario changeover command.
• Electrical compass device
• Secondary LCD indicators
• Voice Module, for speech messages and warnings

3.1.4.3 Simulator
Condor PC flight simulator (www.condorsoaring.com) data can be received via RS232 port after input of suitable passwords (See chapter 5.1.15). This feature is extremely useful for the pilots who want to learn about LX8000 and also like refreshment after winter period. Please note that altitude data will be sent from the simulator, meaning that real final glide training will be possible.

LX8000 version 2.1 June 2009

Page 11 of 97

3.1.5 Technical data
• Power input 10-16 V DC
• Consumption at 12V:
  • 290 mA - minimum brightness without audio and options
  • 300 mA - minimum brightness without audio and with Flarm
  • 380 mA - maximum brightness without audio and options
  • 390 mA - maximum brightness without audio and with Flarm
• 70 mA – additional for LX8000 vario unit
• Cut-out dimensions of LX 8000 digital unit 93.5x81.5mm, outline dimensions: 98x88x115mm inclusive connector
• 57mm (2 1/4") standard aircraft cut-out for LX 8000 vario unit length 120 mm (inclusive connector)

3.1.6 Weight
• 580g LX8000 digital unit
• 300g LX8000 vario unit
Appendix C – WAAS (Wide Area Augmentation System)

This appendix provides an overview of WAAS. The material below is copied with some editing from:
http://en.wikipedia.org/wiki/Wide_Area_Augmentation_System

The Wide Area Augmentation System (WAAS) is an air navigation aid developed by the Federal Aviation Administration to augment the Global Positioning System (GPS), with the goal of improving its accuracy, integrity, and availability. Essentially, WAAS is intended to enable aircraft to rely on GPS for all phases of flight, including precision approaches to any airport within its coverage area.[1]

WAAS uses a network of ground-based reference stations, in North America and Hawaii, to measure small variations in the GPS satellites' signals in the western hemisphere. Measurements from the reference stations are routed to master stations, which queue the received Deviation Correction (DC) and send the correction messages to geostationary WAAS satellites in a timely manner (every 5 seconds or better). Those satellites broadcast the correction messages back to Earth, where WAAS-enabled GPS receivers use the corrections while computing their positions to improve accuracy.

The International Civil Aviation Organization (ICAO) calls this type of system a Satellite Based Augmentation System (SBAS). Europe and Asia are developing their own SBASs, the Indian GPS Aided Geo Augmented Navigation (GAGAN), the European Geostationary Navigation Overlay Service (EGNOS) and the Japanese Multi-functional Satellite Augmentation System (MSAS), respectively. Commercial systems include StarFire and OmniSTAR.

Benefits

WAAS addresses the entire “navigation problem”, providing highly accurate positioning that is extremely easy to use, for the cost of a single receiver installed on the aircraft. Ground- and space-based infrastructure is relatively limited, and no on-airport system is needed. WAAS allows a precision approach to be published for any airport, for the cost of developing the procedures and publishing the new approach plates. This means that almost any airport can have a precision approach and the cost of implementation is dramatically reduced.

Additionally WAAS works just as well between airports. This allows the aircraft to fly directly from one airport to another, as opposed to following routes based on ground-based signals. This can cut route distances considerably in some cases, saving both time and fuel. In addition, because of its ability to provide information on the accuracy of each GPS satellite’s information, aircraft equipped with WAAS are permitted to fly at lower en-route altitudes than was possible with ground-based systems, which were often blocked by terrain of varying elevation. This enables pilots to safely fly at lower altitudes, not having to rely on ground-based systems. For unpressurized aircraft, this conserves oxygen and enhances safety.

The above benefits create not only convenience, but also have the potential to generate significant cost savings. The cost to provide the WAAS signal, serving all 5,400 public use airports, is just under US$50 million per year. In comparison, the current ground based systems such as the Instrument Landing System (ILS), installed at only 600 airports, cost US$82 million in annual maintenance. [citation needed]

Without ground navigation hardware to purchase, the total cost of publishing a runway’s WAAS
approach is approximately US$50,000; compared to the $1,000,000 to $1,500,000 cost to install an ILS radio system.[14]

Further savings can come from the nighttime closure of airport towers with a low volume of traffic. The FAA is reviewing 48 towers for such a potential reduction of services, which it estimates will save around US$100,000 per year at each tower, for a total annual savings of nearly US$5 million.[15]

Drawbacks and Limitations

For all its benefits, WAAS is not without drawbacks and critical limitations.

- The broadcasting satellites are geostationary, which causes them to be less than 10° above the horizon for locations north of 71.4° latitude. This means aircraft in areas of Alaska or northern Canada may have difficulty maintaining a lock on the WAAS signal.
- To calculate an ionospheric grid point’s delay, that point must be located between a satellite and a reference station. The low number of satellites and ground stations limit the number of points which can be calculated.
- Aircraft conducting WAAS approaches must possess certified GPS receivers, which are much more expensive than non-certified units. In 2006, Garmin’s least expensive certified receiver, the GNS 430W, had a suggested retail price of US$10,750.
- WAAS is not capable of the accuracies required for Category II or III ILS approaches. Thus, WAAS is not a sole-solution and either existing ILS equipment must be maintained or it must be replaced by new systems, such as the Local Area Augmentation System (LAAS).
- WAAS Lateral Precision Performance with Vertical guidance (LPV) approaches with 200-foot minimums will not be published for airports without medium intensity lighting, precision runway markings and a parallel taxiway. Smaller airports, which currently may not have these features, would have to upgrade their facilities or require pilots to use higher minimums.
Appendix D – NEAM Codes (Strings)

This appendix provides an explanation of what NEMA 0183 is and an attempt at the codes it creates. The NEMA 0183 is a standard by National Marine Electronics Association (http://www.nmea.org/) and to get the standard they charge $300. This price was much greater when NEMA 0183 first came out, but with the release of the newer NEMA 2000 the price has decreased. The Current price for NEMA 2000 is $4000 (http://www.nmea.org/store/index.asp?show=cprd&cid=7). The following description of NEMA 0183 is taken from: http://en.wikipedia.org/wiki/NMEA_0183

NMEA 0183 (or NMEA for short) is a combined electrical and data specification for communication between marine electronic devices such as echo sounder, sonars, anemometer (wind speed and direction), gyrocompass, autopilot, GPS receivers and many other types of instruments. It has been defined by, and is controlled by, the U.S.-based National Marine Electronics Association.

The NMEA 0183 standard uses a simple ASCII, serial communications protocol that defines how data is transmitted in a "sentence" from one "talker" to multiple "listeners" at a time. Through the use of intermediate expanders, a talker can have a unidirectional conversation with a nearly unlimited number of listeners, and using multiplexers, multiple sensors can talk to a single computer port. Third-party switches are available that can establish a primary and secondary talker, with automatic failover if the primary fails.

At the application layer, the standard also defines the contents of each sentence (message) type so that all listeners can parse messages accurately.

NEMA 0183 strings is the format of the data that the GPS module will output. The meaning of the strings has been extrapolated from known information by many individuals who did not want to pay for the standard. The information I used for this project came from: http://gpsd.berlios.de/NMEA.txt
# Appendix E – Bill of Materials

This appendix provides a Bill of Materials, a list of all the parts used in the final design. The information also includes the cost, name in the PCB, the package, the vendor, the price, and the vendor’s part number of every part.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Description</th>
<th>RefDes</th>
<th>Package</th>
<th>Vendor</th>
<th>Price</th>
<th>Mouser Part No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MISC_VIRTUAL, CRYSTAL_VIRTUAL</td>
<td>U5</td>
<td>Ultiboard\QUARZ_RQCH</td>
<td>Digikey</td>
<td>0.32</td>
<td>SER3205-ND</td>
</tr>
<tr>
<td>3</td>
<td>RESISTOR, 100kΩ</td>
<td>R1, R2, R3</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>0.1</td>
<td>71-TNPW0805100KDEEA</td>
</tr>
<tr>
<td>1</td>
<td>SCH_CAP_SYMS, POTENTIOMETER</td>
<td>X1</td>
<td>Ultiboard\POT2</td>
<td>Mouser</td>
<td>1.11</td>
<td>81-PV37W103C01B00</td>
</tr>
<tr>
<td>5</td>
<td>CAPACITOR, 10µF</td>
<td>C4, C11, C9, C7, C19</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>0.23</td>
<td>80-C0805C106M8P</td>
</tr>
<tr>
<td>1</td>
<td>INDUCTOR, 33µH</td>
<td>L1</td>
<td>Generic\1210</td>
<td>Mouser</td>
<td>0.63</td>
<td>807-1210R-333J</td>
</tr>
<tr>
<td>1</td>
<td>CAPACITOR, 22µF</td>
<td>C5</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>1.19</td>
<td>810-C2012X5R0J226M</td>
</tr>
<tr>
<td>1</td>
<td>CAPACITOR, 33nF</td>
<td>C6</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>0.47</td>
<td>80-C0805Y333K5R</td>
</tr>
<tr>
<td>1</td>
<td>CAPACITOR, 150pF</td>
<td>C10</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>0.21</td>
<td>80-C0805C151K5R</td>
</tr>
<tr>
<td>1</td>
<td>RESISTOR, 31.2kΩ</td>
<td>R4</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>0.56</td>
<td>71-TNPW08053122BR7S</td>
</tr>
<tr>
<td>1</td>
<td>RESISTOR, 1MΩ</td>
<td>R5</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>0.27</td>
<td>660-RN732ATTI004F50</td>
</tr>
<tr>
<td>1</td>
<td>RESISTOR, 499kΩ</td>
<td>R6</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>0.5</td>
<td>660-RN732ATTI4993B25</td>
</tr>
<tr>
<td>2</td>
<td>RESISTOR, 845kΩ</td>
<td>R7, R8</td>
<td>Generic\1206</td>
<td>Mouser</td>
<td>0.7</td>
<td>66-PFC12C845K-B</td>
</tr>
<tr>
<td>2</td>
<td>RESISTOR, 10Ω</td>
<td>R9, R10</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>0.7</td>
<td>66-PFC08LF10-B</td>
</tr>
<tr>
<td>6</td>
<td>CAPACITOR, 100nF</td>
<td>C12, C13, C8, C17, C18, C14</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>0.12</td>
<td>80-C0805X104K4R</td>
</tr>
<tr>
<td>1</td>
<td>DefLCD, LCD_DISPLAY_16x2</td>
<td>U6</td>
<td>Generic\HDR1X16</td>
<td>Sparkfun</td>
<td>14.95</td>
<td>LCD-09054</td>
</tr>
<tr>
<td>1</td>
<td>RESISTOR, 100Ω</td>
<td>R11</td>
<td>Generic\0805</td>
<td>Mouser</td>
<td>0.7</td>
<td>66-PFC08LF100-B</td>
</tr>
<tr>
<td>3</td>
<td>DefButton, PB_NO</td>
<td>U2, U7, U8</td>
<td>Generic\HDR1X3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DefTPS61016, TPS61016</td>
<td>U10</td>
<td>IPC-7351\MSOP-10</td>
<td>Mouser</td>
<td>3.37</td>
<td>595-TPS61016DGS</td>
</tr>
<tr>
<td>1</td>
<td>DefGPS_1, GPS_1</td>
<td>U11</td>
<td>Generic\HDR1X4</td>
<td>Sparkfun</td>
<td>59.95</td>
<td>GPS-09171</td>
</tr>
<tr>
<td>1</td>
<td>GPS antenna</td>
<td></td>
<td></td>
<td>Sparkfun</td>
<td>11.95</td>
<td>GPS-00177</td>
</tr>
<tr>
<td>1</td>
<td>MSP, MSP430F47196</td>
<td>U1</td>
<td>IPC-7351\LQFP-100</td>
<td>Digikey</td>
<td>17.71</td>
<td>296-24644-1-ND</td>
</tr>
<tr>
<td>1</td>
<td>DIODE, 1N4001</td>
<td>D1</td>
<td>IPC-7351\SOD-80</td>
<td>Mouser</td>
<td>3.78</td>
<td>610-CCLHM080</td>
</tr>
<tr>
<td>1</td>
<td>DefPOWER_HEADER_1,</td>
<td>U9</td>
<td>Generic\HDR1X2</td>
<td>ECE Shop</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

66
<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
<th>Part Description</th>
<th>Manufacturer</th>
<th>Model/Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>POWER_HEADER</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DefGPS_2, ON_OFF_SWITCH</td>
<td>1</td>
<td>U14 Generic\HDR1X2</td>
<td>ECE Shop</td>
<td></td>
</tr>
<tr>
<td>JTAG</td>
<td>1</td>
<td>U15 Generic\HDR2X7</td>
<td>ECE Shop</td>
<td></td>
</tr>
<tr>
<td>DefGPS_2, FT232RL</td>
<td>1</td>
<td>U17 IPC-7351\SSOP-28</td>
<td>Mouser</td>
<td>4.5 895-FT232RL</td>
</tr>
<tr>
<td>DefGPS_2, 0.5_AMP</td>
<td>1</td>
<td>U18 Generic\0603</td>
<td>Mouser</td>
<td>0.68 504-TR/0603FA500-R</td>
</tr>
<tr>
<td>DefGPS_2, USB_Connector</td>
<td>1</td>
<td>U13 USER\Mini_USB</td>
<td>Mouser</td>
<td>0.51 154-15410-E</td>
</tr>
<tr>
<td>ZENER, BZX84-C3V0</td>
<td>1</td>
<td>D2 Ultiboard\SOD-123</td>
<td>Mouser</td>
<td>0.05 78-MMSZ5225C-VGS18</td>
</tr>
<tr>
<td>RESISTOR, 15Ω</td>
<td>2</td>
<td>R12, R13 Generic\2512</td>
<td>Mouser</td>
<td>0.12 71-CRCW2512-15-E3</td>
</tr>
<tr>
<td>RESISTOR, 47kΩ</td>
<td>1</td>
<td>R14 Generic\0805</td>
<td>Mouser</td>
<td>0.5 660-RN732ATTD4702B25</td>
</tr>
<tr>
<td>Printed Circuit Board</td>
<td>1</td>
<td></td>
<td>APCircuits</td>
<td>$22</td>
</tr>
</tbody>
</table>
Appendix F – The Software

This Appendix provides the entire source code for the development software. This software is not complete; it has a few bugs and would need revisions to the available screens for the pilot to be useful in flight. It does however demonstrate the full capability of the device.

/*******************************************
* MQP                                   *
* coded By Nicholas Cappabianca         *
*                                          *
* ver 1.0                                *
 *******************************************/

//speed only 3 places and put knots
//no alt
//fg in the middle
//bearing only three places
//set l/d to 42
//dist to home
//bearing to home, dist., fg
//sp,alt,l/d

#include <io430.h>
#include <msp430x471x6.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <in430.h>

void swDelay(unsigned int max_cnt); //software delay
void dispString(int x, int chars[]); //display an array of characters to the screen
void nextLine(void); //move lcd cursor to second line first space
void clearLCD(void); //clear the LCD
void setToHome(void); //set cursor to home position
double toRadian(double x); //convert to radians
double min(double x, double y); //returns the min of 2 values
void fnlGld(void); //returns the final glide
double atod(char array[], int length); //converts a char array into a double

//int
SystemRestart[22]={0xA0,0xA1,0x00,0x0F,0x01,0x03,0xD6,0x07,0x0C,0x12,0x08,0x32,0x19,0xC4,0x09,0x70,0x30,0x64,0x00};
//int ConfigSerialPort[11] = {0xA0,0xA1,0x00,0x04,0x05,0x00,0x01,0x00};

//dummy lat and long for demo
/*
char dummyLat[9] = {0x34,0x32,0x31,0x39,0x2E,0x34,0x33,0x36,0x38};
char dummyLng[10] = {0x30,0x37,0x31,0x34,0x30,0x2E,0x39,0x32,0x35,0x36};
char **dummyPtr;
int j = 0;
*/
double d;

//strings needed for startup questions
int waitingFor[] = {0x57, 0x61, 0x69, 0x74, 0x69, 0x6E, 0x67, 0x20, 0x46, 0x6F, 0x72};
int sats[] = {0x53, 0x61, 0x74, 0x65, 0x6C, 0x6C, 0x69, 0x74, 0x65, 0x73, 0x20};
int isThisHome[] = {0x49, 0x73, 0x20, 0x54, 0x68, 0x69, 0x73, 0x20, 0x48, 0x6F, 0x6D, 0x65, 0x3F};
int whatIsYour[] = {0x57, 0x68, 0x61, 0x74, 0x20, 0x49, 0x73, 0x20, 0x59, 0x6F, 0x75, 0x72, 0x20};
int LDStr[] = {0x4C, 0x2F, 0x44, 0x3F};
int fnl[] = {0x46, 0x6E, 0x6C};
int glideAlt[] = {0x47, 0x6C, 0x6C, 0x69, 0x64, 0x65, 0x20, 0x41, 0x6C, 0x74, 0x65, 0x3F, 0x20};
int b[] = {0x42, 0x3A};
int sp[] = {0x20, 0x53, 0x50, 0x3A};
int s[] = {0x53};
int altStr[] = {0x41, 0x6C, 0x74};
int fg[] = {0x20, 0x46, 0x47};
int fg2[] = {0x20, 0x20, 0x20, 0x20, 0x46, 0x47, 0x3A};
int space[] = {0x20, 0x20, 0x20, 0x20, 0x20};

//variables needed
int state = 1; //global state machine current state
int Ustate = 0; //UART state machine current state
int counter = 0; //counter for storing the lat and long
int ld[2] = {0x34, 0x30}; //user l/d, default to 40
int ldNum = 40; //l/d in an int so i do not have to convert
int fgAlt[4] = {0x30, 0x35, 0x30, 0x30}; //user final glide alt
int fgAltNum = 500; //user fg alt as an int so i dont have to convert
int numSats[2] = {0x30, 0x30}; //array so i can use dispString with this var
int EW[1] = {0x57}; //array so i can use dispString with this var
int NS[1] = {0x4E}; //array so i can use dispString with this var
char lat[9] = {0x34,0x32,0x31,0x34,0x2E,0x33,0x32,0x35,0x37}; //holds the lat value
char lng[10] = {0x30,0x37,0x31,0x34,0x35,0x2E,0x37,0x30,0x34,0x33}; //holds the long value
const char *latPtr;
const char *lngPtr;
double homeLat = 1; // holds home lat
double homeLng = 1; // holds home long
int alt[7] = {0x30,0x30,0x30,0x30,0x30,0x30,0x30};
int bearing[5] = {0x30,0x30,0x30,0x2E,0x30};
int speed[5] = {0x30,0x30,0x30,0x2E,0x30};
int fgNum[5] = {0x20,0x30,0x30,0x30,0x30};
int time[10] = {0x30,0x30,0x3A,0x30,0x30,0x3A,0x30,0x30,0x30,0x30};

int main(void){

// setup clocks
WDTCTL = WDTPW + WDTHOLD;                  // Stop watchdog timer
FLL_CTL0 |= XCAP10PF;                     // Set load capacitance for xtal
SCFI0 |= FN_2;                            // x2 DCO, 4MHz nominal DCO
SCFQCTL = 74;                             // (74+1) x 32768 = 2.45Mhz

// setup buttons
P1DIR = 0x00;  // set to input direction
P1SEL = 0x00;  // set to I/O option
P1IES = 0xE0;  // set to interrupt on high to low
P1IE  = 0xE0;  // port is interruptable

// init uart
P2SEL |= BIT4 + BIT5;                     // P2.4,5 = USCI_A0 RXD/TXD
UCA0CTL0 = 0x00;
UCA0CTL1 |= UCSSEL_2;                     // CLK = SMI
UCA0BR0 = 0x0F;                           // 9600 - 255
UCA0BR1 = 0x00;
UCA0MCTL = 0x84;                           // Modulation
UCA0CTL1 &= ~UCSWRST;                     // Initialize USCI Ustate machine
IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt

// init display pins
P10DIR = 0xFF;  // set P10.3 to P10.0 to output
P10SEL = 0x00;  // I/O option
P9DIR = 0xFF;   // set P9.7 to P9.0 to output
P9SEL = 0x00;   // I/O option

// init display
P10OUT = 0x00;
P9OUT = 0x38; // 8 bit, 2 rows, 5x7 dots
swDelay(500);   // wait for disp to be ready
P10OUT = 0x01; //toggle enable pin high
swDelay(500); //wait
P10OUT = 0x00; //toggle enable pin low
P9OUT = 0x08; //display of, cursor off, no blink
swDelay(500); //wait
P10OUT = 0x01; //toggle enable pin high
swDelay(500); //wait
P10OUT = 0x00; //toggle enable pin low
P9OUT = 0x01; //clear display
swDelay(500); //wait
P10OUT = 0x01; //toggle enable pin high
swDelay(500); //wait
P10OUT = 0x00; //toggle enable pin low
swDelay(500); //wait
P9OUT = 0x0C; // display on, cursor off
swDelay(500); //wait
P10OUT = 0x01; //toggle enable pin high
swDelay(500); //wait
P10OUT = 0x00; //toggle enable pin low
swDelay(500); //wait
P9OUT = 0x06; //autoincrement, shift cursor
P10OUT = 0x01; //toggle enable pin high
swDelay(500); //wait
P10OUT = 0x00; //toggle enable pin low
swDelay(500); //wait
//display init done

//pointers
latPtr = &lat[0];
lngPtr = &lng[0];

_EINT(); //interrupts enabled
while(1){
    switch (state){
        case 0:
            //do i have 4 sats?
            setToHome();
            dispString(10, waitingFor);
            nextLine();
            dispString(10, sats);
            dispString(1, numSats);
            if (numSats[0] == 0x31 || numSats[1] >= 0x34){

clearLCD();
state = 1;
}
break;
case 1:
  //is this home?
  setToHome();
  dispString(12, isThisHome);
  break;
case 2:
  //what is your L/D?
  setToHome();
  dispString(12, whatIsYour);
  dispString(2, LDStr);
  nextLine();
  dispString(1, ld);
  break;
case 3:
  // what is your fnl glide alt?
  setToHome();
  dispString(12, whatIsYour);
  dispString(2, fnl);
  nextLine();
  dispString(10, glideAlt);
  dispString(3, fgAlt);
  break;
case 4:
  //screen 1 displays bearing, sped, number fo sats, alt, and final glide ratio
  /*
  for(j = 0;j<=8;j++){
    lat[j] = dummyLat[j];
  }
  for(j = 0;j<=9;j++){
    lng[j] = dummyLng[j];
  }*/
  setToHome();
  dispString(1, space);
  dispString(1, b);
  dispString(2, bearing);
  dispString(3, sp);
  dispString(2, speed);
  //dispString(0, s);
//dispString(0, numSats);
nextLine();
//dispString(2, altStr);
//dispString(4, alt);
fnlGld();
dispString(6, fg2);
dispString(4, fgNum);
break;
case 5:
    //screen 1 displays bearing, sped, number fo sats, alt, and final glide ratio
    setToHome();
dispString(1,b);
dispString(4, bearing);
dispString(3, sp);
dispString(4, speed);
    //dispString(0, s);
    //dispString(0, numSats);
    nextLine();
dispString(2, altStr);
dispString(4, alt);
fnlGld();
dispString(2, fg);
dispString(4, fgNum);
break;
case 6:
    setToHome();
dispString(3, space);
dispString(7, time);
    nextLine();
fnlGld();
dispString(6, fg2);
dispString(4, fgNum);
    }
    }
    
    /*
    //set ddram address to second line
    P10OUT = 0x00;//set RS to 0
    P9OUT =0xC0;//ddram set command to address 40(first char second line)
    swDelay(100);//wait
    P10OUT = 0x01;//toggle enable pin high
    swDelay(100);//wait
    P10OUT = 0x00;//toggle enable pin low
//display a e on the screen
P10OUT = 0x04; //RS set to high for RAM write
P9OUT = 0x65; //hx for letter A to screen
swDelay(100); //wait
P10OUT = 0x05; //toggle enable pin high
swDelay(100); //wait
P10OUT = 0x00; //toggle enable pin low
swDelay(100); //wait
*/
}

#pragma vector = USCIAB0RX_VECTOR
__interrupt void USCIAB0RX_ISR (void){
  switch(Ustate){
    case 0:
      if (UCA0RXBUF == 0x24){ //if rx is a '$'
        Ustate = 1;
      }else{
        Ustate = 0;
      }
    break;
    case 1:
      if (UCA0RXBUF == 0x47){ //if rx is a 'G'
        Ustate = 2;
      }else{
        Ustate = 0;
      }
    break;
    case 2:
      if (UCA0RXBUF == 0x50){ //if rx is a 'P'
        Ustate = 3;
      }else{
        Ustate = 0;
      }
    break;
    case 3:
      if (UCA0RXBUF == 0x47){ //if rx is a 'G'
        Ustate = 4;
      }else if(UCA0RXBUF == 0x56) { //if rx is a v
Ustate = 16;
} else {
    Ustate = 0;
}
break;
case 4:
    if (UCA0RXBUF == 0x47) { // if rx is a 'G'
        Ustate = 5;
    } else {
        Ustate = 0;
    }
break;
case 5:
    if (UCA0RXBUF == 0x41) { // if rx is a 'A'
        Ustate = 6;
    } else {
        Ustate = 0;
    }
break;
case 6:
    if (UCA0RXBUF == 0x2C) { // if rx is a ','
        Ustate = 7;
    } else {
        Ustate = 0;
    }
break;
case 7:
    if (UCA0RXBUF == 0x2C) { // if rx is a ','
        Ustate = 8;
        counter = 0;
    } else {
        Ustate = 7;
        if (counter == 2 || counter == 5) {
            time[counter] = 0x3A;
            counter++;
        }
        time[counter] = UCA0RXBUF;
        counter++;
    }
break;
case 8:
    if (UCA0RXBUF == 0x2C) { // if rx is a ','
Ustate = 9; //normally
    counter = 0;
} else{
    lat[counter] = UCA0RXBUF;
    counter++;
    Ustate = 8;
}
break;
case 9:
    if (UCA0RXBUF == 0x2C){ //if rx is a ','
        Ustate = 10;
        counter = 0;
    } else{
        EW[counter] = UCA0RXBUF;
        counter++;
        Ustate = 9;
    }
break;
case 10:
    if (UCA0RXBUF == 0x2C){ //if rx is a ','
        Ustate = 11; //for testing
        counter = 0;
    } else{
        lng[counter] = UCA0RXBUF;
        counter++;
        Ustate = 10;
    }
break;
case 11:
    if (UCA0RXBUF == 0x2C){ //if rx is a ','
        Ustate = 12;
        counter = 0;
    } else{
        NS[counter] = UCA0RXBUF;
        counter++;
        Ustate = 11;
    }
break;
case 12:
    if (UCA0RXBUF == 0x2C){ //if rx is a ','
        Ustate = 13;
    } else{
Ustate = 12; //if not stay here
}
break;
case 13:
if (UCA0RXBUF == 0x2C){ //if rx is a ','
    Ustate = 14;
    counter = 0;
} else{
    numSats[counter] = UCA0RXBUF;
    counter++;
    Ustate = 13;
}
case 14:
if (UCA0RXBUF == 0x2C){ //if rx is a ','
    Ustate = 15;
} else{
    Ustate = 14; //if not stay here
}
break;
case 15:
if (UCA0RXBUF == 0x2C){ //if rx is a ','
    Ustate = 0;
    counter = 0;
} else{
    alt[counter] = UCA0RXBUF;
    counter++;
    Ustate = 15;
}
break;
case 16:
if (UCA0RXBUF == 0x54){ //if rx is a 'T'
    Ustate = 17;
} else{
    Ustate = 0;
}
break;
case 17:
if (UCA0RXBUF == 0x47){ //if rx is a 'G'
    Ustate = 18;
} else{
Ustate = 0;
}
break;
case 18:
if (UCA0RXBUF == 0x2C){ //if rx is a ',
    Ustate = 18;
} 
else{
    Ustate = 0;
}
break;
case 19:
if (UCA0RXBUF == 0x2C){ //if rx is a ',
    Ustate = 20;
    counter = 0;
}else{
    bearing[counter] = UCA0RXBUF;
    counter++;
    Ustate = 19;
}
break;
case 20:
if (UCA0RXBUF == 0x2C){ //if rx is a ',
    Ustate = 21;
} 
else{
    Ustate = 20;
}
break;
case 21:
if (UCA0RXBUF == 0x2C){ //if rx is a ',
    Ustate = 22;
} 
else{
    Ustate = 21;
}
break;
case 22:
if (UCA0RXBUF == 0x2C){ //if rx is a ',
    Ustate = 23;
} 
else{
Ustate = 22;
}
break;
case 23:
if (UCA0RXBUF == 0x2C){ //if rx is a ','
    Ustate = 0;
    counter = 0;
} else {
    speed[counter] = UCA0RXBUF;
    counter++;
    Ustate = 23;
}
break;
}
/*
#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCIAB0TX_ISR (void){

}
*/
//port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void port1ISR(void){
    //0x80 = red button
    //0x40 = first white button
    //0x20 = second whit button
    //from right to left
    //lazy sw debounce
    swDelay(65535);
    swDelay(65535);

    switch (state){
    case 1:
        if(P1IFG == 0x80){
            homeLat = atod(lat, 9)/100;
            homeLng = atod(lng, 10)/100;
            if(EW[0] == 0x57){
                homeLng *= -1;
            }
            if(NS[0] == 0x53){
                homeLat *= -1;
            }
        }
    }
clearLCD();
state = 2;
}
break;
case 2:
    //increment or decrement the l/d input
    if (P1IFG == 0x20){
        ldNum++;
        ld[1] = ld[1] + 0x01;
        if (ld[1] >= 0x3A){
            ld[0] = ld[0] + 0x01;
            ld[1] = 0x30;
        }
    }
    if (P1IFG == 0x40){
        ldNum--;
        ld[1] = ld[1] - 0x01;
        if (ld[1] <= 0x2F){
            ld[0] = ld[0] - 0x01;
            ld[1] = 0x39;
        }
    }
    if (P1IFG == 0x80){
        state = 3;
        clearLCD();
    }
    if (ld[0] <= 0x2F || ld[0] >= 0x3A){
        ld[0] = 0x30;
    }
break;
case 3:
    //increment or decrement the final glide input
    if (P1IFG == 0x20){
        fgAltNum += 100;
        fgAlt[1] = fgAlt[1] + 0x01;
        if (fgAlt[1] == 0x3A){
            fgAlt[0] = fgAlt[0] + 0x01;
            fgAlt[1] = 0x30;
        }
    }
    if (fgAlt[1] == 0x3A){
        fgAlt[0] += 0x01;
    }
if (P1IFG == 0x40)
    fgAltNum -= 100;
    fgAlt[1] = fgAlt[1] - 0x01;
    if (fgAlt[1] <= 0x2F)
        fgAlt[0] = fgAlt[0] - 0x01;
        fgAlt[1] = 0x39;
    if (fgAlt[1] <= 0x2F)
        fgAlt[0] -= 0x01;
        fgAlt[1] = 0x39;
        fgAlt[2] = 0x39;
    if (P1IFG == 0x80)
        clearLCD();
        state = 4;
    if (fgAlt[0] <= 0x2F || fgAlt[0] >= 0x3A)
        fgAlt[0] = 0x30;
        fgAlt[1] = 0x30;
        fgAlt[2] = 0x30;
        fgAlt[3] = 0x30;
        fgAltNum = 0;
        break;
    case 4:
        clearLCD();
        state = 5;
        break;
    case 5:
        clearLCD();
        state = 6;
        break;
    case 6:
        clearLCD();
        state = 4;
        break;
P1IFG = 0x00; //reset interrupt flags

//software delay
void swDelay(unsigned int max_cnt){
    unsigned int cnt1=0,cnt2;
    while(cnt1 < max_cnt){
        cnt2=0;
        while(cnt2 < 65535)
            cnt2++;
        cnt1++;
    }
}

void dispString(int x, int chars[]){
    int i = 0;
    //for loop to display chars
    for (i = 0; i <= x; i++){
        //display a e on the screen
        P10OUT = 0x04; //RS set to high for RAM write
        P9OUT = chars[i]; //hx for letter A to screen
        swDelay(100); //wait
        P10OUT = 0x05; //toggle enable pin high
        swDelay(100); //wait
        P10OUT = 0x00; //toggle enable pin low
        swDelay(100); //wait
    }
}

void nextLine(void){
    //set ddram address to second line
    swDelay(100); //wait
    P10OUT = 0x00; //set RS to 0
    P9OUT = 0xC0; //ddram set command to address 40(first char second line)
    swDelay(100); //wait
    P10OUT = 0x01; //toggle enable pin high
    swDelay(100); //wait
    P10OUT = 0x00; //toggle enable pin low
}
void clearLCD(void){
    swDelay(500);  //wait
    P10OUT = 0x00;  //toggle enable pin low
    P9OUT  = 0x01;  //clear display
    swDelay(500);  //wait
    P10OUT = 0x01;  //toggle enable pin high
    swDelay(100);  //wait
    P10OUT = 0x00;  //toggle enable pin low
}

void setToHome(void){
    //set ddram address to first line
    P10OUT = 0x00;  //set RS to 0
    P9OUT  = 0x80;  //ddram set command to address 00 (first char fist line)
    swDelay(100);  //wait
    P10OUT = 0x01;  //toggle enable pin high
    swDelay(100);  //wait
    P10OUT = 0x00;  //toggle enable pin low
}

double toRadian(double x){
    return (3.141592653589793238462643/180) * x;
}

void fnlGld(void){
    double x = atod(lat, 9)/100;
    double y = atod(lng, 10)/100;
    if(EW[0] == 0x57){
        y *= -1;
    }
    if(NS[0] == 0x53){
        x *= -1;
    }
    double dLat, dLng;  //delta lat and long
    int r = 6371;
    int fglide = 0;
    double a, c;

    //haversine formula to get dist from two lat and longs
    dLat = toRadian(x - homeLat);
dLng = toRadian(y - homeLng);

\[ a = \sin\left(\frac{dLat}{2}\right) \cdot \sin\left(\frac{dLat}{2}\right) + \cos(\text{toRadian}(\text{homeLat})) \cdot \cos(\text{toRadian}(x)) \cdot \sin\left(\frac{dLng}{2}\right) \cdot \sin\left(\frac{dLng}{2}\right) \]

c = (2 \cdot \sin(\text{sqrt}(1-a), \text{sqrt}(a)))

d = (r \cdot c) \cdot 3280.8399

//d = d * 3280.8399

fglide = (\text{int})(d/ldNum) - fgAltNum;

//convert to ascii text to print on LCD
if (fglide >= 9999){
    fglide = 9999;
}
if (fglide <= -9999){
    fglide = -9999;
}

//get each digit
if(fglide <= -1){
    fgNum[0] = 0x2D;
}else{
    fgNum[0] = 0x2B;
}
r = fglide % 1000;
fgNum[1] = abs((fglide - r)/1000);
if(fgNum[1] != 0){
    if(r == 0){
        fglide = 0;
    }else{
        fglide = r;
    }
}
r = fglide % 100;
fgNum[2] = abs((fglide - r)/100);
if(fgNum[2] != 0){
    if(r == 0){
        fglide = 0;
    }else{
        fglide = r;
    }
}
r = fglide % 10;
fgNum[3] = abs((fglide - r)/10);
if(fgNum[3] != 0){
if(r == 0){
    fglide = 0;
} else {
    fglide = r;
}
}
fgNum[4] = abs(fglide);

// convert to ascii
for (r = 1; r <= 4; r++){
    if(fgNum[r] == 0){
        fgNum[r] = 0x30;
    }
    if(fgNum[r] == 1){
        fgNum[r] = 0x31;
    }
    if(fgNum[r] == 2){
        fgNum[r] = 0x32;
    }
    if(fgNum[r] == 3){
        fgNum[r] = 0x33;
    }
    if(fgNum[r] == 4){
        fgNum[r] = 0x34;
    }
    if(fgNum[r] == 5){
        fgNum[r] = 0x35;
    }
    if(fgNum[r] == 6){
        fgNum[r] = 0x36;
    }
    if(fgNum[r] == 7){
        fgNum[r] = 0x37;
    }
    if(fgNum[r] == 8){
        fgNum[r] = 0x38;
    }
    if(fgNum[r] == 9){
        fgNum[r] = 0x39;
    }
}
double min(double x, double y){
    if (x < y){
        return x;
    } else {
        return y;
    }
}

double atod(char array[], int length){
    int i = 0;
    double j = 10;
    double k = 1;
    int decimal = 0;
    double ans = 0;
    for (i = 0; i < length; i++){
        switch (array[i]){
            case 0x2E:
                j = 1;
                k = .1;
                decimal = 1;
                break;
            case 0x30:
                if (decimal == 0){
                    ans = ans * 10;
                }
                break;
            case 0x31:
                ans = ans * j + 1 * k;
                break;
            case 0x32:
                ans = ans * j + 2 * k;
                break;
            case 0x33:
                ans = ans * j + 3 * k;
                break;
            case 0x34:
                ans = ans * j + 4 * k;
                break;
            case 0x35:
                ans = ans * j + 5 * k;
        }
    }
    return ans;
}
break;
case 0x36:	ans = ans*j + 6*k;
break;
case 0x37:	ans = ans*j + 7*k;
break;
case 0x38:	ans = ans*j + 8*k;
break;
case 0x39:	ans = ans*j + 9*k;
break;
}
if(decimal == 2){
	k = k * .1;
}
if(decimal == 1){
	decimal = 2;
}
return ans;
Appendix G – How to Program the Device

This Appendix will show how to program the device. This includes instructions on how to setup the development environment, how to write a simple program, and how to download it to the device.

The development environment consists of a PC, a software development program, and a JTAG to connect to the device. There are two options for a software development program; Code Composer by TI and IAR Embedded Workbench for MSP430. Code Composer is a fully featured development program but is still in a “beta” phase, it is relatively new and still needs the kinks worked out, but it is free with no limitations. IAR Embedded Workbench for MSP430 is also a fully featured development program, it is older than Code Composer and works well, but it is not free. IAR offers a free version of IAR Embedded Workbench for MSP430 called Kickstart but it has a 4k code limitation, meaning it will not compile a program larger than 4kB. The software for the device found in Appendix F was written using IAR Kickstart and all the following directions will be for use with IAR Kickstart. To obtain a copy of IAR Kickstart follow this link http://supp.iar.com/Download/SW/?item=EW430-KS4 then fill out the forms and IAR will email you a link to a download page and a license key.

The JTAG is a device that will link the PC to the device. The JTAG used in the development for the program found in Appendix F is the MSP-FET430UIF, it can be purchased for $100 from TI at this link http://focus.ti.com/docs/toolsw/folders/print/msp-fet430uif.html. Many other companies make JTAG modules for the MSP430, IAR and Olimex for instance, all of these may work and are most likely cheaper but the MSP-FET430UIF is known to work and this device was designed to use it. A reference design from TI can be found here focus.ti.com/lit/an/slaa276a/slaa276a.pdf this will allow the creation of a USB JTAG.

Once a JTAG is obtained and IAR Kickstart is installed in IAR click Project -> Options. In the Options window, under the General Options tab, make sure the correct device is selected (for this device it is the MSP430F47196). Then under the Debugger tab make sure that the FET Debugger is chosen under the Drivers menu. Following that under the FET Debugger tab under the Connection menu select the Texas Instruments USB-IF. After this is completed the setup of the development environment is completed.

To create a program the user must be familiar with the C programming language, any higher forms of the C language; C++, etc. will only be marginally useful most of their commands will not be recognized. After understanding section 5.5 and skimming through Appendix F additional functions to the original software should not be difficult. An example program follows; it will display Hello World onto the device’s display.
**Hello World**
* coded By Nicholas Cappabianca
* 
* ver 1.0
*******************************************/

#include <msp430x471x6.h>
#include <stdlib.h>
#include <math.h>
#include <in430.h>

void swDelay(unsigned int max_cnt); //software delay
void dispString(int x, int chars[]); //display an array of characters to the screen

//variables needed
state = 0;
int HelloWorld[] = {0x48,0x65,0x7C,0x7C,0x6F,0x20,0x57,0x6F,0x72,0x7C,0x64};
int HelloWorld2[] = {0x48,0x45,0x4C,0x4C,0x4F,0x20,0x57,0x4F,0x52,0x4C,0x44};

int main(void){

    //setup clocks
    WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
    FLL_CTL0 |= XCAP10PF;                     // Set load capacitance for xtal
    SCFI0 |= FN_2;                            // x2 DCO, 4MHz nominal DCO
    SCFQCTL = 74;                             // (74+1) x 32768 = 2.45Mhz

    //setup buttons
    P1DIR = 0x00;  //set to input direction
    P1SEL = 0x00;  //set to I/O option
    P1IES = 0xE0;  //set to interrupt on high to low
    P1IE  = 0xE0;  //port is interruptible

    //init display pins
    P10DIR = 0xFF; //set P10.3 to P10.0 to output
    P10SEL = 0x00; //i/o option
    P9DIR = 0xFF;  //set P9.7 to P9.0 to output
    P9SEL = 0x00; //i/o option

    //init display
    P10OUT = 0x00;
P9OUT = 0x38; //8 bit, 2 rows, 5x7 dots
swDelay(500); //wait for disp to be ready
P10OUT = 0x01; //toggle enable pin high
swDelay(500); //wait
P10OUT = 0x00; //toggle enable pin low
P9OUT = 0x08; //display on, cursor off, no blink
swDelay(500); //wait
P10OUT = 0x01; //toggle enable pin high
swDelay(500); //wait
P10OUT = 0x00; //toggle enable pin low
P9OUT = 0x01; //clear display
swDelay(500); //wait
P10OUT = 0x01; //toggle enable pin high
swDelay(500); //wait
P10OUT = 0x00; //toggle enable pin low
P9OUT = 0x0C; //display on, cursor off
swDelay(500); //wait
P10OUT = 0x01; //toggle enable pin high
swDelay(500); //wait
P10OUT = 0x00; //toggle enable pin low
P9OUT = 0x06; //autoincrement, shift cursor
P10OUT = 0x01; //toggle enable pin high
swDelay(500); //wait
P10OUT = 0x00; //toggle enable pin low
swDelay(500); //wait
//display init done
_EINT(); //interrupts enabled

while(1){
    switch(state){
    case 0:
        dispString(10,HellowWorld); //display hello world
        break;
    case 1:
        dispString(10,HellowWorld2); //display hello world
    }
}
}
//port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void port1ISR(void){
  //0x80 = red button (enter)
  //0x40 = first white button (down)
  //0x20 = second whit button (up)
  //from right to left

  //lazy sw debounce
  swDelay(65535);
  swDelay(65535);

  if(P1IFG == 0x20){
    state = 0;
    }
  if(P1IFG == 0x40){
    state = 1;
    }

  P1IFG = 0x00; //reset interrupt flags
}

//software delay
void swDelay(unsigned int max_cnt){
  unsigned int cnt1=0,cnt2;
  while(cnt1 < max_cnt){
    cnt2=0;
    while(cnt2 < 65535)
      cnt2++;
    cnt1++;
  }
}

void dispString(int x, int chars[]){
  int i = 0;

  //for loop to display chars
  for (i = 0; i <= x; i++){
    //display a e on the screen
    P10OUT = 0x04;//RS set to high for RAM write
    P9OUT  = chars[i];//hx for letter A to screen
    swDelay(100);; //wait
P10OUT = 0x05; //toggle enable pin high
swDelay(100); //wait
P10OUT = 0x00; //toggle enable pin low
swDelay(100); //wait

Appendix H – LCD Character Pattern
This Appendix show the standard character pattern used for programming the LCD display. The characters need to be inputted into the program Upper bit first then Lower bit, i.e. the letter H is read Upper bits LHLL Lower Bits HLLL. This translates into 0x48 in the program.
### Standard character pattern

<table>
<thead>
<tr>
<th>Upper bit</th>
<th>Lower bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLLL</td>
<td>CG RAM</td>
</tr>
<tr>
<td>LLLH</td>
<td>1A0a9</td>
</tr>
<tr>
<td>LLLH</td>
<td>2BBr</td>
</tr>
<tr>
<td>LLLL</td>
<td>3C5cs</td>
</tr>
<tr>
<td>LHLL</td>
<td>4DTdt</td>
</tr>
<tr>
<td>LHLH</td>
<td>5EUeu</td>
</tr>
<tr>
<td>LHHL</td>
<td>6FVfv</td>
</tr>
<tr>
<td>LHHH</td>
<td>7GWgw</td>
</tr>
<tr>
<td>HLLL</td>
<td>8HXhx</td>
</tr>
<tr>
<td>HLLH</td>
<td>9IYiy</td>
</tr>
<tr>
<td>HLHL</td>
<td>@JZjz</td>
</tr>
<tr>
<td>HLLL</td>
<td>KIKK</td>
</tr>
<tr>
<td>HHLH</td>
<td>LLI1</td>
</tr>
<tr>
<td>HHHL</td>
<td>MNm</td>
</tr>
<tr>
<td>HHHH</td>
<td>?OLO+</td>
</tr>
</tbody>
</table>

- "0AP`F" - "5z0p"