Low-Power, Context-Aware Global Positioning System

Rashin Bolkameh, Chris Chadwick, Eric Hsu, Daniel Rolfe and Michael Stevens

***Abstract*— In our partnership with the Sandia National Laboratories in 2008-2009, building a low power GPS tracking system unit for the Sandia's based device, which not only had to be compatible with the other boards that were made by Sandia National Laboratories, it also had to be capable of capturing a short sample of GPS satellite signal and store the data into the serial flash memory, has been accomplished. The project has covered the hardware design, software design, board design, board fabrication and the testing through the development of a GPS tracking system unit.**

I. Introduction

The Global Positioning System (GPS) is a fully functional Global Navigation Satellite System (GNSS) [1]. The Global Positioning System is composed of twenty-four satellites 20,200 km (12,500 miles or 10,900 nautical miles) above the earth [1]. The satellites are spaced in orbit so that at any time a minimum of four satellites will be in view to users anywhere in the world [1]. The satellites continuously broadcast position of the satellite and time data to users throughout the world [1]. GPS has applications beyond navigation and location determination. GPS can be used for cartography, forestry, mineral exploration, wildlife habitation management, monitoring the movement of people and objects and bringing precise timing to the world [1].

This project involves the design of a positioning module to work in conjunction with existing wireless sensor network hardware. It also has developed the next generation of the location tracking system to be used in Sandia's products. The chip that has been designed for this project fits a low power GPS receiver to a standardized Sandia stack of boards from factor and log location.

The use of GPS tracking system to determine the position has become increasingly popular over the year. The GPS uses dozens of satellites that transmit the microwave signals and enable the receivers to determine their current location. The satellites continuously transmit the time, a precise orbit for the satellite sending the message, the general system health and the rough orbits of all the GPS satellites. Then the receivers use this information to calculate their current position.

In order to calculate the current position, the receiver must obtain a signal from at least four satellites. Four satellite signals are necessary to calculate the latitude, longitude, altitude, and to account for any time variance in the receiver, this is shown in figure 1.

.

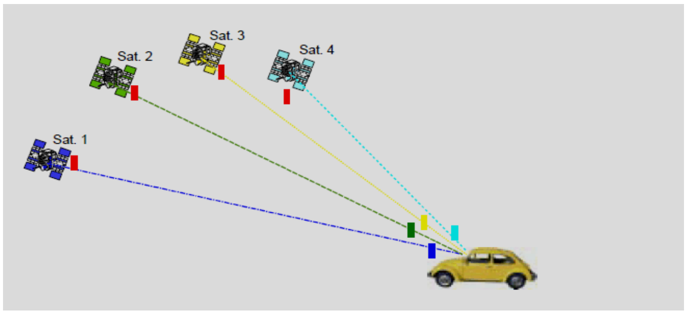

Fig. 1. Four satellite signals [2].

Sandia National Laboratories uses technology to do applications for the government. They want to keep track of where a piece of equipment is and not go to the places it shouldn't or be tampered with by someone other than the intended user.

Therefore, the board that has been designed for this project is going to keep track, where these products are, and also the path they have taken to get there.

Sandia uses a stack of boards where each board has its own functionality, but they also can communicate with the other boards in the stack as well. Some boards include: a power board that regulates the power for others in this stack of boards; a data storage board; a board for communication; and various sensor boards. In this project a sensor board has been designed that is compatible with Sandia's stack of boards.

In order of designing a product to be compatible with the Sandia's boards, there are several requirements that had to be met. The size requirement for the board in this project was 38 mm by 32 mm and a spacing of 5mm between boards.

This spacing allowed for the parts that are less than 1.5 mm to be placed on the bottom side of the board and all other parts that should be less than 3.5 mm to be placed on the top side of the board, this is shown in Figure2.

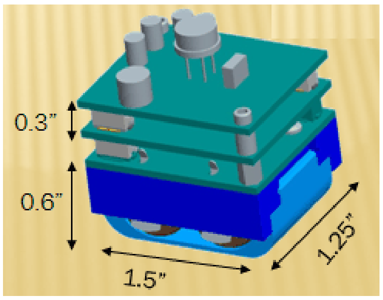

Fig. 2. Size requirements.

In addition to the size requirement, adding the low power consumption device, utilizing a GPS Front-End Module on the Sandia stack of boards, to capture and store that data into the serial flash memory (128 Kbits), and transfer data to the PC via serial has been accomplished. The device runs off two batteries and so the current that is drawn from the batteries should be minimized, so the batteries are not drained too quickly. Power cycling components that have a higher current draw is required to have lower average power consumption.

Most GPS receivers require a full signal from the satellites (about 38sec). In this project the device is able to calculate the position with less than one second of data. To be able to do that, during this project, several ways of making the power consumption to be significantly lower than a personal GPS receiver and also calculating the position in less time, has been discovered. One way to accomplish those will be by accessing a database of GPS information, knowing the exact time when the signal was sent, and another way is to use an accelerometer to only activate the receiver when the chip is moving.

In this project, both the hardware and the software has been designed for the module, and then also focused on minimizing the power consumption and ran several tests for the module to be able to figure out the results.

The hardware design includes, selecting appropriate components for a custom GPS solution; system-level design to interface the GPS and accelerometer; and printed circuit board (PCB) design, fabrication, and assembly.
For the hardware design, the size of the board, compatibility with the other boards on Sandia's stack of boards and required power were the main considerations in choosing the parts for developing a GPS tracking system unit. Most parts had to be chosen based on the requirements, but some components were specified by the Sandia for compatibility with their stack.

The software design includes, writing firmware to run on the microcontroller, writing software for positioning data post-processing, and writing software to be capable to store and retrieve the data.

Most traditional GPS calculates the position in about 38 seconds, the tasks of a conventional GPS receiver are as follows, and also it is shown in figure 3.

Firstly the GPS receiver searches for the weak satellite signals, then it measures their relative timing accurately to give the distance to the satellites [2].
In order to calculate where the receiver is, the receiver must first find out precisely where the satellites are in space. This "ephemeris" information is transmitted slowly by the satellites, repeated every 30s [2]. The need to demodulate this signal is the primary factor that slows down the response time of a GPS receiver [2]. Only after it has decoded the orbit information for (at least) four satellites does the GPS receiver calculate a position fix [2].
Subsequent position fixes when tracking can then be carried out quickly because the satellite orbit information is now known and can be reused, and also because it is easy to keep continuous track of the satellite signals once they have been found [2]. This system design works fine for navigation on a journey, where the user is prepared to wait a minute or so to start with, and then wants to track position continuously, using energy while doing so [2].

However this type of system is not suitable for the application users expect the device to be small and light to carry around everywhere, and demand a long battery life [2].

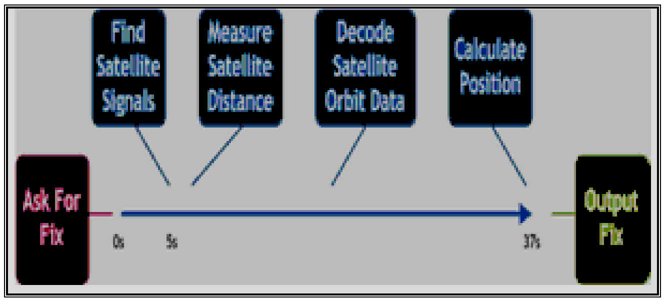

Fig. 3. Work done in a conventional GPS Receiver [2].

Geoate is an independent software, that has been used on the cameras these days, it provides the users an ability to find out their locations. Geotate technology instantly captures and assigns geotags to digital images; then the assigned RAW (unprocessed) location data is processed when the photographer uploads their digital images. It automatically adds a label location to the pictures as well [2].

Geotate technology and this design of GPS unit, they both have a lot of similar functionality. The similar tasks of the GPS receiver and the Geotate GPS receiver are as follows and it also shown in figure 4.
Can instantly capture outdoor location, and is ideally suitable for the users because of its instant response, and very low energy consumption [2].
Since for most of the traditional GPS receivers, takes them about 38 seconds to receive the signals, in both, this design of GPS system and Geotate technology, it takes the receiver to capture a short sample of the GPS satellite signal in less than 0.2 second and it will simply get captured And stored in the memory.

There is no delay or waiting for any processing task to be performed, therefore the user can carry on with what they were doing without getting interrupted. Later the signal samples can be sent out to the PC by using the serial [2].
The system then processes the signal sample to calculate the location where the samples were captured. As a result the device itself acts instantly, can be small and lightweight, and can have a battery life of several weeks [2].

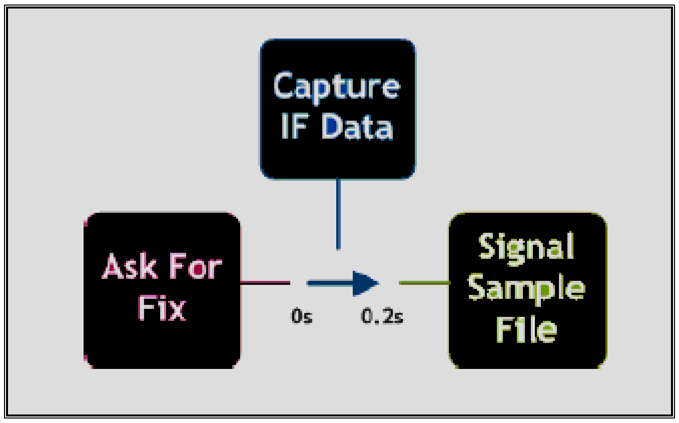

Fig. 4. Work done in a Capture and Process [2].

This report will discuss what has been accomplished, how it was accomplished and suggestions for the next team that will be working on the Sandia National Laboratories clinic in future.

This paper is broken into five sections. Part 1, discusses about the “Hardware Design”, Part 2, discusses about the “Board Design”, Part 3, discusses about the “Programming the Utah GPS stack”, Part 4, discusses about the “Testing and Results” and Part 5, will be the conclusion.

II. Hardware Design

In designing the project it became apparent that there was an order to the design process. This consisted of series of steps. First identify the problems and finding solutions. Second find components to satisfy the solutions that were discovered. Third read the date sheets for the components selected to better understand the components. Forth figure out how all of the components will work to together. Fifth if any problems are found in between any of these steps return to the first step.

The first step is identifying the problem. The problems that were given by Sandia National Laboratory are first the ability to capture raw GPS data and store that date. This is so the data can be processed at a later date. Next, the ability to be able to remove the data collected from the board to the PC, so the data can be processed. Also, the board must be able to detect movement so that the board will not take data in the same location over and over again. Also, the board must be designed for low power. This is import because the board runs on two batteries and must run for weeks without new batteries. Lastly, the board must conform to the standers of the Sandia stack.

One of the first components that were looked for was a GPS front end. There were 3 GPS front end modules that considered they were made by SiGe, Rakon, and Maxim. The SiGe chip is a GPS front end module that outputs raw GPS data at a rate of 16 Mbis/s. This chip was under consideration because there is a test sample available. The test sample is a USB drive that allowed the collection of the raw GPS data on a computer. This data then can be processed with software that will be discussed later. The other two modules Rakon and Maxim chip were also under consideration because there is different software available to process the raw GPS data.

There was two software looked at for processing the data. They were a matlab based software and a software used by a company called Geotate. The matlab software is a software that came with a book “A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach (Applied and Numerical Harmonic Analysis).”[3] The matlab software works by processing the data using a numerical technique. One of the biggest draw backs in that the matlab software it requires 1 GB of data to process. The other software was developed by a company called Geotate. This software is interesting because of the fact that it only required 128kbits of data to process the coordinate. It’s able to do this because it would use a time stamp from when the coordinates were taken and retrieve the rest of the signal from a data base and then process that data. This is appealing do to the fact of low power consideration.

Another major component that was debated over was the flash memory. There were 3 different types of flash memory that were looked at nand flash, microSD, and serial nor flash. Nand flash is a common kind of flash memory found in many modern devices such as, jump drives, mp3 players, and cell phones. The advantage of this type of memory was its large capacity. The size that was under consideration was 1 to 2GB. The down side is the fact that the data must be formatted in page setup to be able to write to it. This would require the data to be formatted by the micro processer before it was written to the flash. The next type of memory that was considered was microSD. The advantages to it was also it size, 2GB was the size for consideration. MicroSD is also a type of nand flash so it would also require writing in pages. Also, there are two protocols for writing to microSD they are SPI or SD. SD protocols is an industry protocols and to get the specifics of you must pay for them. The other mode is SPI is a standard protocol that is supported by many micro processers. The last type of memory that was looked at is serial nor flash. The advantage to the serial nor flash is that the data need not need to be formatted. The disadvantage to this type of memory is that it is smaller in size the size looked at was 128Mbit.

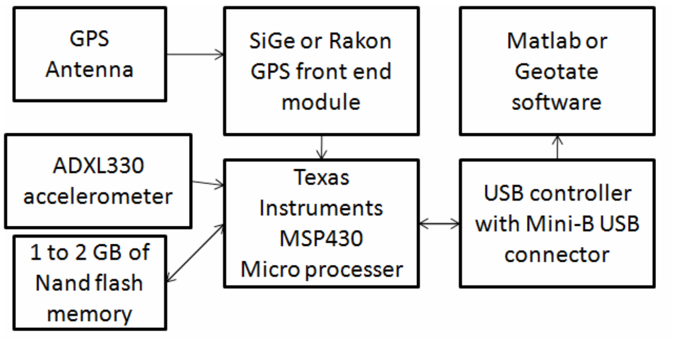
GPS Antenna
SiGe or Rakon GPS front end module
Matlab or Geotate software
ADXL330 accelerometer
Texas Instruments MSP430 Micro processer
USB controller with Mini-B USB connector
1 to 2 GB of Nand flash memory

Fig. 5. Revision 1 of the board.

After these considerations it leads to the first revision to the board which can be seen in Fig. 5. The components that have yet to be discussed are the GPS antenna, accelerometer, micro controller, USB controller, and mini-B USB connector.
The GPS antenna is an antenna that is tuned to 1.575 GHz which is the stander GPS frequency. The size of the antenna is 7 by 2 mm.

The accelerometer is the ADXL330. This is a three axis accelerometer meaning it can detect acceleration in any direction. The accelerometer is used so if the board is stationary power and memory space aren’t wasted by taking samples of same coordinate.

The micro controller is in Texas Instruments MSP430 family. It’s an 8 MHz low power micro processer. It was chosen for the fact that Sandia had a lot of experience with this processer and other boards in the stack.

The USB controller is the ISP1760. This is a USB 2 supporting USB controller. The connector is just a stander mini-B type USB connector.

The first revision of the board as seen in Figure 1, this shows everything feeding thought the micro controller.

In designing of the board there was some problems discovered. The USB controller we had chosen was not right for the application. It was designed for use in a USB hub. Also using USB would require us to write drivers for the computer for the PC to recognize the board, and the USB would consume a lot of power. This could be avoided by only powering up the controller only when it connected to the PC and powered by the PC. But for these reason it was decided to go a different rough and not use the USB.

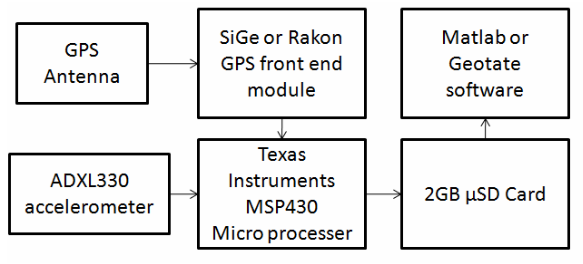
GPS Antenna
SiGe or Rakon GPS front end module
Matlab or Geotate software
ADXL330 accelerometer
Texas Instruments MSP430 Micro processer
2GB µSD Card

Fig. 6. Revision 2 of the board.

In revision 2 of the board, as seen in Fig. 6, there were a few changes manly the addition of the USB controller and 2GB Nand flash memory. To fill the gap left by these items it was decided to use a 2GB microSD card. This would allow for transfer of the data to the card and when the data need to be retrieved the card could be remove and place directly into the PC were the data could be removed, and processed by the software.

Unfortunately there were also problems with this design had problems as well. First were problem with the processing software. The matlab software was unsuccessful a processing with less than 1GB of data. This would only allow us to capture 2 samples which are useless. Next was problem with Geotate software. After contacting Geotate discussing motives we were unable to obtain the Geotate software for our use. This was because it would require a lot of work on their end and they were mainly interested in producing commercially available produces. After this and discussing the situation it was decider that focusing on the processing would be removed and looked at if there was still time. So we proceeded by just focusing on collecting 128kbit sample know it is possible to process with that much data.

The second problem was with the microSD card. The SiGe chip output data at 16 Mbit/s but the micro controller is only 8 MHz. This caused a problem because that data need to be formatted before it was written to the flash. The solution was discovered in Revision 3 as seen in Fig. 7.

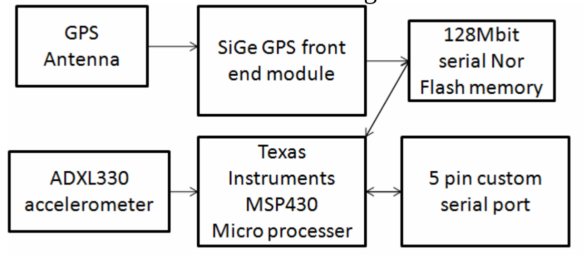
GPS Antenna
SiGe GPS front end module
128Mbit serial Nor Flash memory
ADXL330 accelerometer
Texas Instruments MSP430 Micro processer
5 pin custom serial port

Fig. 7. Revision 3 of the board.

Revision 3 is the final revision of the board. The solution to the problem was to use Serial Nor Flash in place of the microSD card. This would work by multiplexing the signals from the SiGe GPS module and output pins from the micro controller. This would allow for the micro controller to set up the Flash memory and then write directly from the SiGe to the Flash. It would also allow for the data to be formatted by the micro controller to be output to the PC. To output the data to the PC a 5 pin custom serial port would be used. This would allow for transfer of the data to the PC though a hyper terminal.

III. Board Design

A. *Using Cadsoft Eagle*

The board schematic and layout were designed using Cadsoft Eagle. The first step of designing a printed circuit board (PCB) with Eagle was to make any parts that were not already contained in an Eagle library. Adding a part to a library consisted of making a symbol to be used in the schematic and a footprint to be used in the layout. After all parts were available in Eagle, the schematic was created. The schematic was checked using Eagle's electrical rule check. Then a layout was generated from the completed schematic. Parts were placed in their desired location on the board and the traces were constructed. This was done with the help of Eagle's design rule check and autorouter. After the layout was complete, gerber files were made using Eagle's CAM processor. The gerber files were checked with Viewplot and then sent to Sierra Proto Express for fabrication. After the boards were fabricated, they were sent to PCB Solutions to be assembled.

B. *Creating parts in Eagle*

Before the board could be designed, a model in Eagle was needed for every chip that was used on the board. The parts that were not already contained in Eagle had to be made. First, a symbol was made. The symbol is what is used in the schematic. To make the symbol, the desired shape of the part was drawn. The shape could be anything that was desired, this was usually a square. Then a pin had to be added for every pin on the chip. The pins were numbered and named according to the datasheet. The direction of the pins was also set, e.g. input, output, power, ground, etc. In addition to a symbol, a footprint had to be made. A footprint is the physical layout of the chip. This had to be made to the exact size specifications of the chip given in the datasheet. There are three major layers included in the footprint which are silkscreen, copper, and solder mask. The solder mask layer indicates where there should be a lack of solder mask so that the part can be soldered to the board. The silkscreen is printed on the board and can be used to label the chips or anything else that is desired. With a completed symbol and footprint, the two were connected. That way Eagle knows which pins on the symbol correspond to which pins on the layout.

C. *Creating a schematic in Eagle*

With a model for all the parts used, a schematic was created. All of the parts were added to the schematic and then they were connected appropriately using nets. Labels were added to important nets. Most of the sub-schematics were made from looking at the datasheets and hooking up the chips as recommended. For all chips, decoupling capacitors were used. These are capacitors that are placed close to the power supply pins to help regulate the supply voltage and prevent voltage spikes. When the schematic was complete, Eagle’s electrical rule check was run. It checks that all pins are connected and it checks for close but unconnected nets. After the layout is made, it will also check that the layout and schematic are consistent. From the schematic, Eagle can generate a partslist, which is a list of all of the parts in the schematic and their labels. The partslist was used to help make a bill of materials, which was used to order all of the parts.

D. Generating a layout in Eagle

After the schematic was totally finished, a layout was generated from it. When the layout is first created, all of the parts are connected with airwires. First the board was sized to the appropriate size for the Sandia stack. Then all of the parts were placed within the board. Some parts were placed on the top and some were placed on the bottom. The rules for the design rule check were also set. The rules include things like minimum spacing between copper and drill holes. They also include minimum sizes for copper traces and drill holes. The minimum requirements of the company that was used to fabricate the board (Sierra Proto Express) could be used to set the rules. The actual rules were set a little bit more lenient than Sierra Proto’s minimum requirements to help prevent broken traces or unintentionally connected traces. The rules that are set in the design rule check are also used for Eagle’s autorouter. So that when the traces are routed, they will be done so without breaking any of the specified rules in the design rule check.

E. Completing the layout

The autorouter was used to route as many of the traces as possible after all of the parts were placed in their desired locations. Routing the traces turns the airwires into actual copper traces. The autorouter has four levels of optimization that were also set. Any traces that the autorouter did not route were manually routed. The board was created using four layers of copper, the top, bottom, and two middle layers. The two middles layers were supply layers used for ground and power. In Eagle supply layers are drawn inverted, meaning that they are drawn where copper should not be placed. After the layout was complete a centroid file was generated from Eagle. A centroid file is a spreadsheet describing exactly where all of the parts are located. It tells the part name, what layer the part is located on, the x and y coordinates of the part, and the rotational orientation of the part. The centroid file was very helpful for having the boards assembled. The complete layout of the top of the board can be seen in Fig. 8 and the bottom can be seen in Fig. 9.

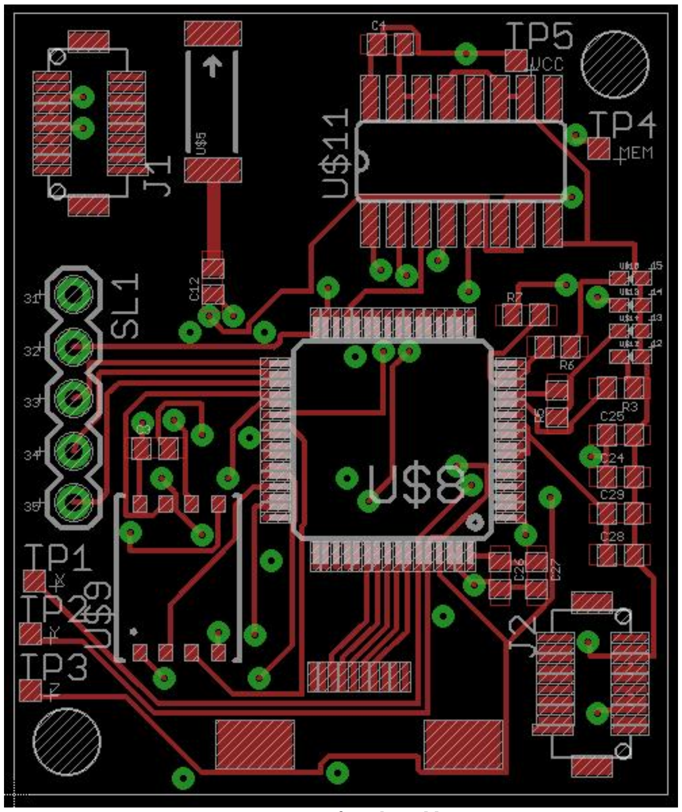

Fig. 8. Layout of top board layer.

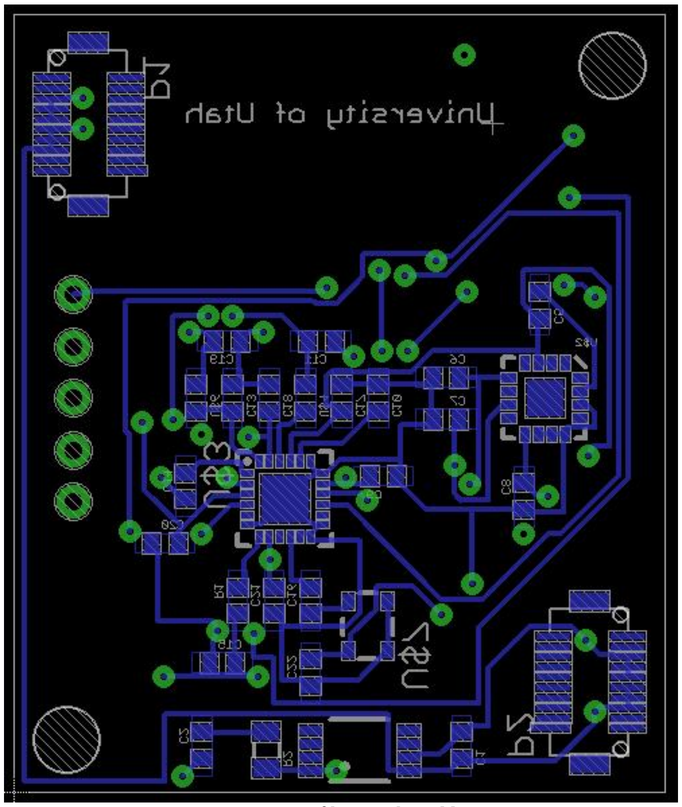

Fig. 9. Layout of bottom board layer.

F. Creating and viewing gerber files

Once the layout was completed, gerber files were generated by Eagle’s CAM processor. Gerber files are text files describing each layer. One gerber file was created for each layer including top copper, top silkscreen, top solder mask, the two middle layers, bottom copper, bottom silkscreen, bottom solder mask, and a drill file describing where all the holes were to be drilled. Because the gerber files are what were sent to the fabrication company, it was a good idea to double check them. Viewplot is a free gerber file viewing program that was used to view the gerber files. The files were loaded and viewed and looked just like the layout in Eagle. The layout as seen in Viewplot can be seen in Fig. 10.

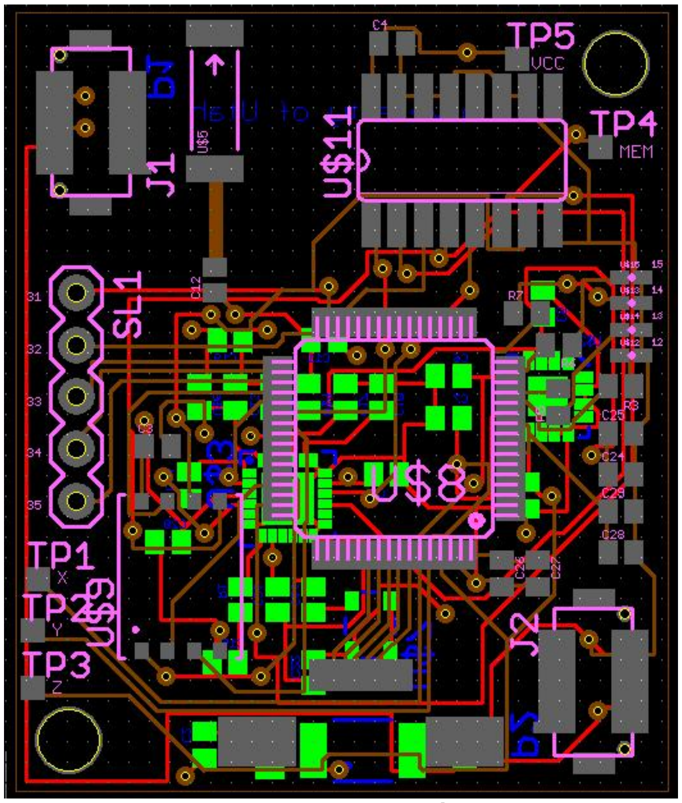
Fig. 10. Layout as seen in Viewplot.

G. *Fabricating the board*

After everything was complete, the gerber files were sent to Sierra Proto Express. The fabricated boards were sent back to us within a few days. While the boards were being fabricated, a bill of materials was made. The bill of materials was made by adding Digi-Key part numbers to the partslist that was previously generated by Eagle. Using the bill of materials, all of the parts were ordered. Once the boards and all of the parts were in hand, they were sent to PCB Solutions to have the boards assembled. Along with the parts and boards, the bill of materials and the centroid file were needed for assembly. After the boards were assembled, four complete boards had been created.

IV. Programming

A. *Picking a development environment*

Texas Instruments has provided two different development suites for development on the TI MSP430 line of micro controllers. The first development environment is Code Composer Essentials and the second is IAR Embedded Workbench Kickstart. The software for this project was developed in IAR Kickstart because IAR kickstart was the first development suite to work on any of the available sample code.

IAR kickstart is similar to other development environments such as Visual Studio or Eclipse. There are some minor differences to IAR Kickstart since it is made for developing on micro controllers. Some notable differences are; The MSP430 micro controller currently being used must be selected in the project options, IAR Kickstart has more than one debugging mode, and IAR kickstart allows the programmer to simulate program interrupts. Selecting the micro controller for the current project involves going into the project properties and picking the model from the available list. The two available debugging modes are FET Debugger and Simulator. The main difference between FET Debugger and Simulator modes is the FET Debugger mode downloads the program to a micro controller to run it. The Simulator mode simulates some of the actions the micro controller would normally do. Some of the available options do not work in both debugging modes. For example, the TAR register does not count in Simulation mode. The TAR is supposed to continually count and is normally used in the timers built into the MSP430 line of micro controllers. IAR Kickstart also provides the ability to simulate program interrupts. This is useful in testing to make sure that the interrupts are setup properly. Simulating interrupts only works in the Simulator debug mode. Another notable option in IAR Kickstart is the choice between C, C++ and Assembly for programming languages. This project was developed in C.

B. *Code Development*

For this project to work some custom software would need to be developed for the MSP430F169 micro controller. The software needed to be able to accurately capture and store GPS samples that would be transferred off system to be processed later. Below is a simple flow diagram of control flow in the software. Specifics about each of the programs parts will be explained later.

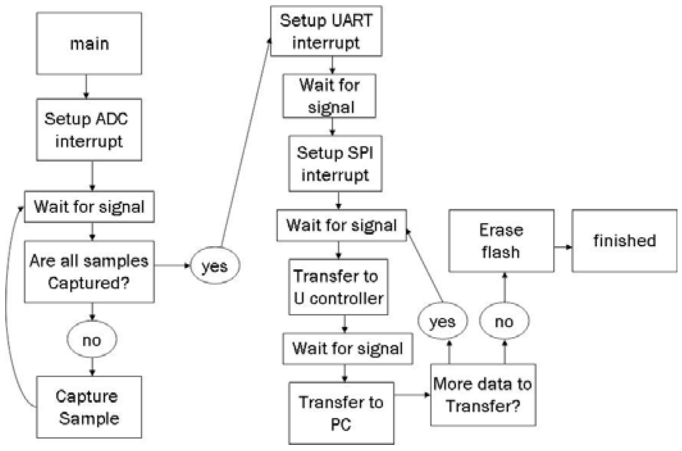

Fig. 11. Control flow of the custom program for the GPS board control flow begins in main.

Software development for this project can be split into five main sections; creating a timer used to accurately wait inside the program, getting GPS samples from the SiGe GPS front end to flash memory, transferring sub GPS samples from flash memory to the micro controller, transferring sub GPS samples from the micro controller to the computer to be processed, and making sure all of the interrupts inside the program are properly setup and working. Program execution begins in main. Interrupts are setup for the analog to digital converter. The program will then wait for a signal from the accelerometer to find out when the system begins moving. Once the signal from the accelerometer is received the program will check to see if all GPS samples have been captured. The amount of samples needed is setup during the beginning of the program. If more samples are still needed a sample will be gathered, then the program will loop back up to waiting for the signal from the accelerometer. Samples are continually acquired after the micro controller gets signal from the accelerometer until samples are no longer needed. At this point the micro controller will setup the UART interrupt and wait for a signal from a computer. Once the signal from the computer is received the UART interrupt will setup the SPI interrupt. This means the SPI interrupt is nested inside the UART interrupt. The SPI interrupt then requests data from the serial flash. When the data is ready the micro controller will receive a signal, the data will be stored on the micro controller. When the micro controller memory is full the UART interrupt will take over. Items the SPI interrupt has stored into memory will be loaded into the UART transmit register which is controlled by the UART interrupt. Data on the micro controller is continually sent to the computer until the micro controller has sent its entire memory contents. The program then checks to see if there is more data in the serial flash to be transferred. If there is more data the SPI interrupt will take over to gather more data, if not the flash memory is erased and the program ends. Now that the overall control flow has been covered each sub part of the program will be covered in more detail.

C. *The need and design of a timer*

There are multiple points within the program where an accurate stall or sleep is needed to allow time for events to happen. One such event is when the GPS board begins moving. If a GPS sample is taken right away there will only be limited distance between GPS samples. To account for this fault the timer can be used to create a stall in the program. Using this method there will be at least a minimum time period between GPS samples. The program as written will wait ten minutes between samples. Below is a flowchart depicting the general flow of a timer.

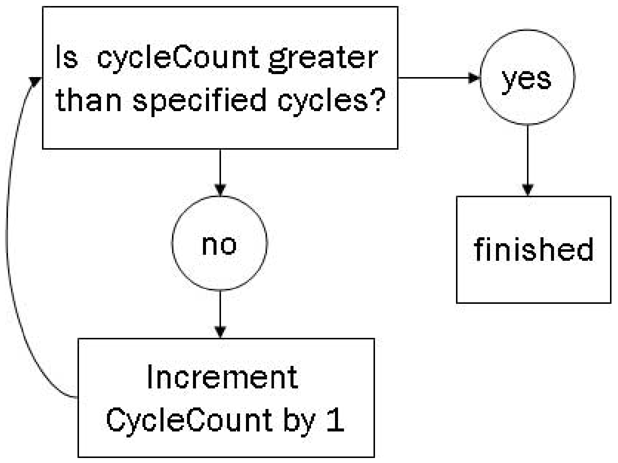
Is cycleCount greater than specified cycles?
yes
finished
no
Increment CycleCount by 1

Fig. 12. Timer code control flow.

In this diagram program execution begins in the comparison between cycleCount and cycles. Cycles is initialized earlier in the program, and is set based on how long the timer should wait. CycleCount is initialized to zero. Using Figure 12 we can see that if CycleCount is less than the specified cycles CycleCount will be incremented by one. This will happen continually until CycleCount is greater than the amount of specified cycles. In this case the program will have waited long enough and program execution will continue. Although this covers the basic idea behind the timer on the micro controller there are some minor details that should be covered. The timer on this micro controller needs to be sourced from one of the clocks. The two options available for this project are the main clock (MCLK) and the sub main clock (SMCLK). SMCLK runs in higher power saving modes than MCLK so this project uses SMCLK. Incrementing cycleCount is actually controlled by a special purpose register called TAR. TAR can be set to count up, down, up then down, or continually. This method has it counting up only. There is also a divider of up to eight than can be applied to the source clock, this means TAR will only increment once per eight clock ticks. Since TAR is only sixteen bits wide, and SMCLK runs at 1MHz so there is a limit to how long the timer can wait. To get around that limit the wait function is defined once and stalls for a short period of time. If the stall needs to be longer the program will loop, calling the wait as many times as needed to wait the required amount of time. The timer is not depicted in Figure 1 for the program flow because many of these portions of the program will need to call wait, for differing lengths of time to ensure all components are ready.

D. *Capturing a GPS sample*

For the GPS board to work as specified it needs to be able to gather and store GPS samples of 128KB. Since the SiGe front end GPS gathers data at 16.12Mb per second the sample capture should only last about 200ms. The figure below depicts control flow for capturing GPS samples.

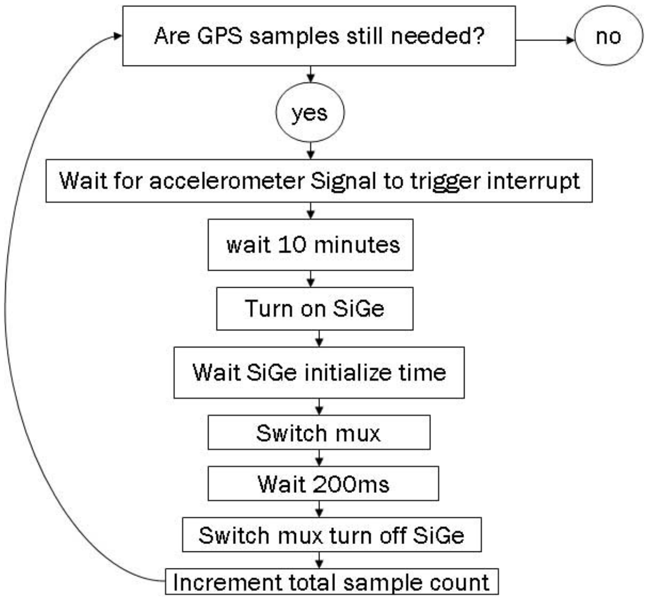
Are GPS samples still needed?
no
yes
Wait for accelerometer Signal to trigger interrupt
wait 10 minutes
Turn on SiGe
Wait SiGe initialize time
Switch mux
Wait 200ms
Switch mux turn off SiGe
Increment total sample count

Fig. 13. GPS sample acquisition control flow.

Program execution in the GPS sample acquisition code begins with a check to see if more samples are still needed. A global value keeps track of the amount of samples wanted. This value can be used to make sure the memory on system does not overflow since each sample is 128KB. If more samples are needed the program will wait for a signal from the accelerometer. Once the signal triggers the interrupt the system must moving in one of the three axis directions. The program waits 10 minutes to avoid taking multiple GPS samples in the same area. The SiGe is turned on and the program waits until the SiGe has properly initialized. The mux that controls data flow to the serial flash is then set to allow data flow from the SiGe to the memory. Then wait for 200ms to capture GPS data of size 128KB. Program flow turns off the SiGe and returns the mux to the default position connecting the micro controller to the memory. The total acquired sample count is incremented by one. Then the flow returns to the total sample check. Once all of the samples have been gathered the program will exit this portion. The figure below demonstrates how much of the overall program has been built at this point.

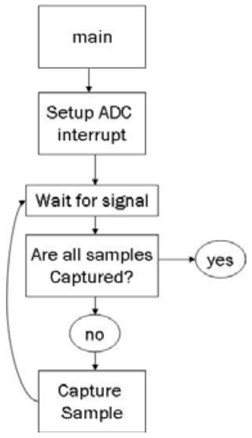

Fig. 14. Program flow so far.

E. *Setup and use of the SPI interrupt*

Ideally once all of the GPS samples have been captured the program would be able to transfer information to the micro controller until it was full. Then the information could be transmitted to the computer to be processed. These steps will be explained separately as in the ideal case, and then explained as things appear in the code. Once all of the GPS samples have been complete data transfer to the computer can begin. The micro controller communicates to the serial flash using SPI protocol. To be able to transfer the data the SPI interrupt needs to be setup on the micro controller. The code sets the micro controller is set to the SPI master on 3-wire SPI communication. After this SPI interrupts are enabled and information is requested from the serial flash. Once the information is sent to the micro controller it will enter this interrupt function that captures and stores the information.

F. *Setup and use for the UART interrupt*

Again ideally the micro controller will have data stored on it that needs to be transmitted to the computer. Since the micro controller will be communicating with the computer using UART protocol the code must setup the interrupt for the UART communication. Program execution now does the setup for the UART interrupt. Setup for the UART interrupt involves setting the character length to 8 bits, synchronizing the serial clock to SMCLK, and setting the baud rate for serial communication. Once these steps are complete the micro controller goes into low power mode to wait for something to trigger the UART interrupt. In this case there are two portions of code to run, one when data is ready to be sent, and the other when data is received. When the data is ready to be retrieved a computer will be hooked up to the GPS board and it will send a character z through hyper terminal. This step is important to be sure no data is lost during this transmission. The data stored on the micro controller will be loaded into the transmit register and sent to the computer. The data retrieve step from the serial flash using SPI protocol and the transmit to computer step using UART protocol will be repeated until all of the data has been sent to the computer. The figure below shows program execution load to micro controller and transmit to computer steps in the code as it actually happens. Minor differences between the ideal case and the code will be explained while walking through this program execution.

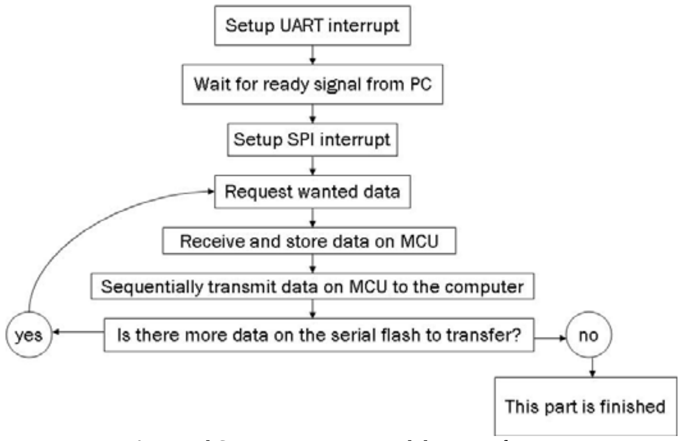
Setup UART interrupt
Wait for ready signal from PC
Setup SPI interrupt
Request wanted data
Receive and store data on MCU
Sequentially transmit data on MCU to the computer
yes
Is there more data on the serial flash to transfer?
no
This part is finished

Fig. 15. UART and SPI interrupt setup and data transfer to computer.

G. *Tying the UART and SPI interrupts together*

Software execution for these two steps actually starts with setting up for the UART interrupt. Nothing else happens until signal is sent letting the software know a computer is hooked up waiting for the data on the flash. This signal is a ‘z’ through hyper terminal. Once this signal is received the UART interrupt does the setup for the SPI interrupt. This is creating nested interrupts, which can be done according to the user’s guide for the micro controller. At this point the micro controller will request data from the flash. Data is sent to the micro controller which stores it. When the memory on the micro controller is full the data will be loaded into the UART transmit register which causes the UART interrupt to take over program execution. Data is continually loaded into the UART transmit register and sent to the computer until the micro controller has sent its memory contents. The program will then check to see if any other data is still on the serial flash waiting to be sent. If there is data the micro controller will request it and the SPI interrupt will take over. Otherwise this portion of the program is over. The figure below shows the software development progress so far.

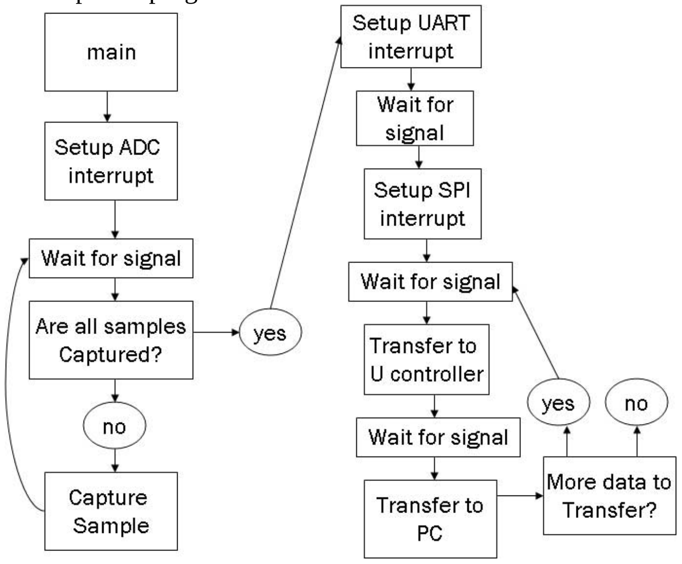

Fig. 16 Software development progress up to SPI and UART interrupts.

Comparing figure 16 to figure 11 it is easy to tell that most of the software development for this project has been completed. The only major component left to do at this point is to make sure the flash memory gets erased during each use. Due to a limitation in the flash memory chosen for this project the memory needs to be erased or it will not properly store data. The limitation has to do with how the memory is designed. This particular flash memory does not store the information it gets by doing an exact copy into memory. Instead this memory relies on the fact that it should contain all ones at the beginning of the program, and will do bit flipping for any bit that should be a zero. The limitation in this design is the memory cannot flip a bit from a zero to a one. This limitation means if this memory is not erased after use and needs to store a one in a location that currently stores a zero this flash memory will not accurately report the data. Erasing this flash memory sets all of the bits stored to ones. As long as this memory is erased before each use this system should not have any problems. To deal with this right after all of the data is transferred from the serial flash the contents of the flash will be erased. To erase the serial flash the software sets the write enable bit on the pin for the serial flash, then waits until all the data has been overwritten.

V. Testing and Results

A. *Manufactured Board Review*

Once we received our boards back with all the designed parts solder on, we identified all the components on the board. First part of the testing was to verify that all the components are where we had designed them in Eagle. Figure 17 shows the bottom side of the board and figure 18 shows the top side of the board.

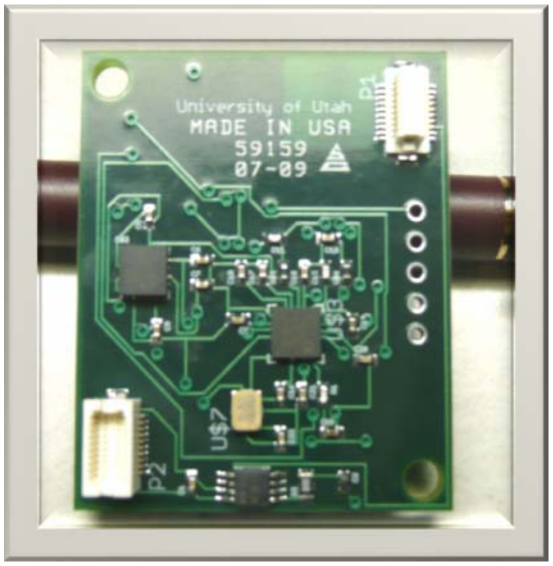

Fig. 17. Bottom side of our manufactured board.

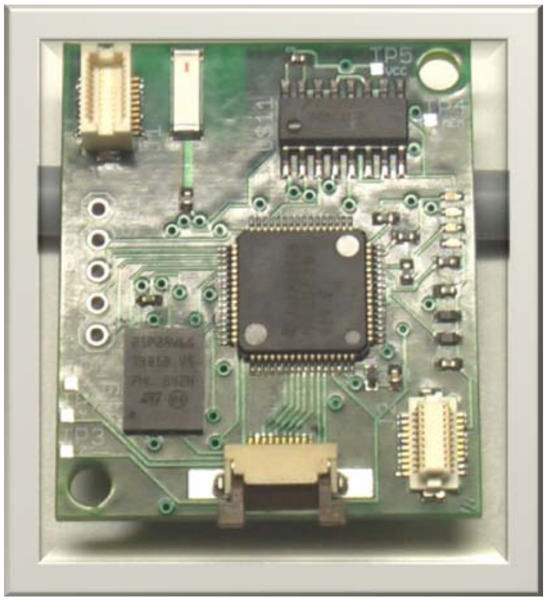

Fig. 18. Top side of our manufactured board.

Due to many errors on our part during the manufacturing of the board, the boards came back to us very late. This cut into our testing and debugging phase that shortened our timeline.

B. Problems Encountered

During the testing phase, we encounter three major problems that potentially stall our project but we had overcome two of these and in the end we stall on the last major problem.

1) The first problem that encounter was when we tried to plug our board into a Sandia Test Platform board. The Sandia Test Platform board had the connectors that provided power and communication via the two custom made sockets. When we plug the board into the Test Platform, there was a huge current draw which indicated there was a short in the design. After few hours of testing and measuring connection, we isolated to the lower right custom socket looking from the top as shown in figure 19. The custom socket was wired up incorrectly which resulted in crossing the power and creating a short on the board.

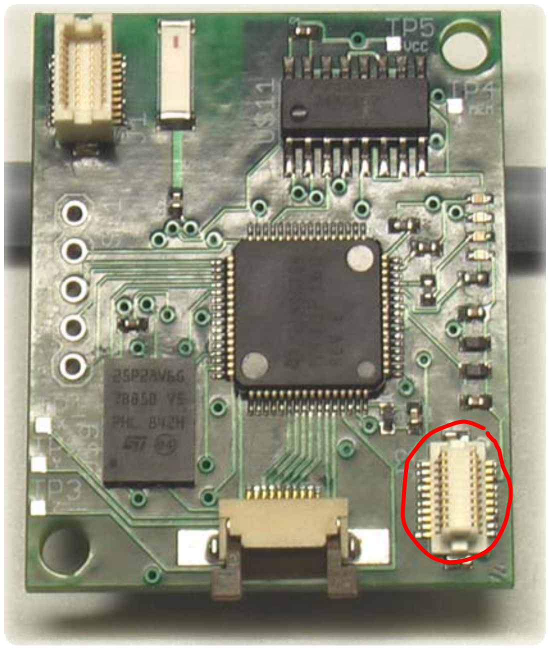

Fig. 19. Lower right socket.

2) The second problem that we encounter during testing was an irregular voltage when the supply voltage socket problem was fixed. With an input voltage of greater than 3.3V DC, the voltage regular was outputting 1.66V DC. At first, we thought that the voltage regulator was an incorrect part, not working properly or wired up improperly. After few days of testing and isolating the load after the voltage regulator, we narrowed down the problem to the micro controller. Initially there was around 200mA of current being drawn which resulted in voltage regulator drop from 3.3V to 1.66V. We tried to force a 3.3V DC in the output of a voltage regulator but resulted in a huge current draw which leads to our assumption it was not the voltage regulator. After painstakingly isolating each load by de-soldering several tiny components we isolated the problem to a specific component.

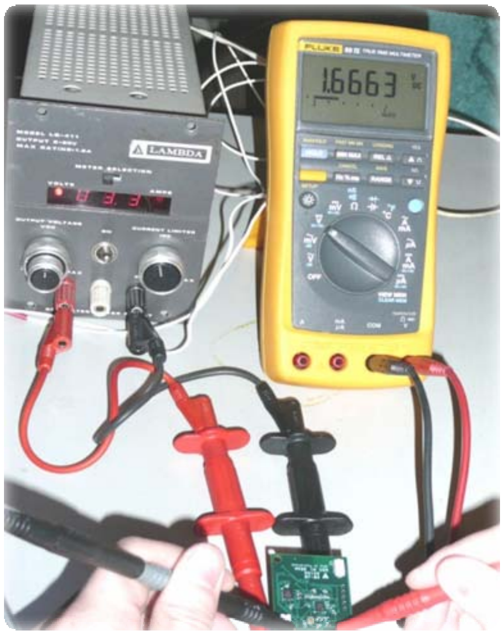

Fig. 20. Voltage regulator output.

3) The third problem that we encounter which stalls the rest of the project was the issue of micro processor not being able to be flashed. Without the ability to flash the micro processor, we would not be able to test our programming code or make the device do what we wanted it to function.

Once these problems were identified, we took steps to come up with solutions. Most of these problems were not as big and complicated as the last one, which involved trying to flash the micro processor.

C. Fixes and Solutions

For the first problem, the fix was easy, either we wired up the connection correctly or not use that socket in the current setup by de-soldering it. Without manufacturing more boards which takes a while, we de-solder the lower right connector which allows us to power the board up with un-regulated power. We verified that the input of voltage regulator was getting the power and voltage it required to function.

The fix for second problem was a bit challenging than before Since the output of the voltage regulator was 1.66V instead of 3.3V that was what we designed for, the voltage was too low to power up the devices on the board. At first we thought it was the voltage regulator, but eventually it was narrow down to the micro processor. It seems that components that needed to supply voltage to the micro processor were wired up incorrectly causing the micro processor to draw more current than it required. By de-soldering pin 64 (Avcc) on the micro processor and removing the associated capacitors, the problem was fixed. This resulting procedure made it possible for the voltage regulator to output 3.3V in the output.

The final problem still does not have a fix as the preparation of this report. So far we connected the JTAG up with the IAR Software that allowed the MSP430F169 to be programmed. The IAR Software from TI would not detect our micro processor indicating that the micro processor is not connected or powered up correctly. We verified that the voltage regulator is power up correctly, JTAG connections are wired up properly, and we took special precautions in regarding with electrostatic discharge. Even with all these checks, all manufactured board will not flash correctly.

D. More Troubleshooting

In the following weeks to come we are prepared to do more testing and fixes in the next revision of the board. We planned to de-solder MSP430F169 from the manufactured board and test it on a TI development kit. We also plan to check the schematic for the TI development kit and Sandia base stack to compare how they connected the MSP430 micro processor. Since both of those were used to test the flashing of micro processor as a baseline.

E. Planned Application

Once the captured RAW GPS data is transferred to a Personal Computer, we can then use either software like Geotate or our own developed MATLAB to process them. Once these software outputs the longitude, latitude, and the altitude of the raw GPS data, we plan to collect them into a .CSV file. A comma separated value (.CSV) can them be used to input into a program (either written by our group or a similar program online) and then outputs a .KMZ file.

Keyhole Markup Language (KML) is an XML-based language schema for expressing geographic annotation and visualization on existing or future Web-based, two-dimensional maps and three-dimensional Earth browsers. KML was developed for use with Google Earth, which was originally named Keyhole Earth Viewer. We can then open these KML file with Google Earth to view all the processed GPS coordinates in a visually interesting way.

VI. Recommendations for Next Team

The problems listed in testing and results have been isolated and fixed on the schematic and layout. The connectors (J2 and P2) were corrected. We have added 0 ohm resistor to each load to make the debugging components much easier. The following resistors are all 0 ohms: R2, R4, R8, R9, and R10 as labeled in the schematic. The programming of JTAG has been solved by connecting the RST line from the microcontroller to power through a resistor and a capacitance to ground. I find that if the testing platform of the Sandia stack device sometimes does not work and best way is to power it using tapped out lines into the Voltage Regulator. I would recommend reading the included documentations of all the components for our board if the team continues using the selected components. If at all possible, it is recommended that the team assemble a complete prototype of the board to test it completely before manufacturing another board. It is also recommended that the transmission line from the antenna be looked at as it might be necessary to move the Sige chip to the top side of the board. Another recommendation is the following team should consider adding a push button to the design and layout. This push button may simplify the code for the micro controller immensely if used to signal when a computer is connected to the device and ready to receive data. The simplification would come from the possibility of avoiding the use of nested interrupts. Also note that the code included went through minimal testing and verification. Expect to spend a fair amount of time changing and updating the program. The programming portion of this paper outlines a general idea on how the program could be developed, but this may not be the best approach. Keep in mind the given code wasn't loaded onto the board and tested. If nothing else read the included user's guide, it was very helpful for explaining how to use the built in hardware. This program relies heavily on nested interrupts, test this early to make sure both interrupts will actually trigger because if they do not a program redesign will be needed.

VII. Conclusion

We have laid the ground work in regard with the design decisions and board layout for the next clinic team. This is a very challenging project and we recommend getting an early start on it. We would like to thank Steven Paradise and Adrian Wong from Sandia National Laboratories for all of their help and the opportunity to work on this challenging project. We would also like to thank Professor Cameron Charles for all his help and guidance on this difficult project.

References

[1] http://geography.about.com/od/geographictechnology/a/gps.htm

[2] http://geotate.com/en/technology

[3] K. Borre, D. M. Akos, N Bertelsen, P. Rinder, and S. H. Jensen, *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach (Applied and* ***Numerical Harmonic Analysis).*** Boston, MA: Birkhäuser.