Project Number: KAC-0701

Multi-Client Embedded Telemetry System[1]

A Major Qualifying Project Report

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Matthew A. Babina

Ryan T. Moniz

Michel A. Sangillo Jr.

Date: October 11, 2007

Professor Kevin A. Clements Ph.D., Major Advisor

[1] This work was sponsored by the Air Force Contract ESC/XPK. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Abstract

The Multi-Client Embedded Telemetry System (MCETS) is an ultra-low-power prototype data acquisition system developed in collaboration with MIT Lincoln Laboratory for use across a wide range of telemetry applications. Capable of collecting both atmospheric and kinematic data, the MCETS incorporates a network of small modular clients that stream data to a server in real-time. This project is concerned with all aspects of the system, including defining the system's functionality, designing the client hardware, developing firmware, and writing server-control software.

Executive Summary

The Multi-Client Embedded Telemetry System (MCETS) is a newly designed prototype data acquisition system developed for MIT Lincoln Laboratory to aid in rapid prototyping of telemetry modules used for varying mission areas. In order to increase their data acquisition capability, The Laboratory desires a more flexible system that can acquire telemetry data – position, velocity, and acceleration – from several modular clients at a raw data rate of ten Hertz, and for this data to be accumulated by a data analysis server. Among other specifications, The Laboratory requires each client in the system to operate on batteries for at least ten minutes, consume less than five watts of power, and weight less than one kilogram. Accordingly, the MCETS is a custom-designed system that exceeds virtually all of these requirements, providing a telemetry system capable of acquiring Cartesian and Geodetic position, Cartesian and Geodetic velocity, acceleration, temperature, pressure, angular rate, and magnetic field strength.

The basic concept of the MCETS is for numerous modular clients to remain in a low-power standby mode until they are individually queried for data acquisition by the MCETS server. Once needed, the server initiates communications with selected clients using unique electronic identification numbers. After the communication link is opened, the server requests data at a specified data acquisition rate between one and one-hundred Hertz. The server also indicates the length of data acquisition ranging from one to 65,535 seconds (eighteen hours, twelve minutes, and fifteen seconds), and the particular telemetry data to acquire (e.g., one or more of temperature, pressure, acceleration, angular rate, magnetic field strength, Cartesian position, Geodetic position, Cartesian velocity, Geodetic velocity, and/or GPS receiver time). Ultimately, this flexibility and functionality is accomplished from three major subsystems: the MCETS hardware, firmware, and software.

Each MCETS client incorporates a four-layer stack of printed circuit boards, including a sensor, microcontroller development, GPS OEM receiver, and power supply board. The sensor board (top layer) is a custom-designed board that houses the analog sensors (temperature, pressure, acceleration, angular rate, and magnetic field strength), their associated electrical components, an 802.11b embedded wireless module for Internet Protocol-based communications, and the necessary connections to interface with the other three layers. The second and third layer – the microcontroller development and GPS OEM receiver board – are commercially available boards that incorporate a Texas Instruments© MSP430 microcontroller and a GPS processor, respectively. Lastly, the fourth layer (bottom layer) is a custom-designed power supply that provides 3.3-volt, 5.0-volt, and 11.1-volt sources for the three other boards, which employs high efficiency switching voltage regulators and an 11.1-volt lithium ion battery.

The hardware for the MCETS clients is controlled via firmware written in the assembly programming language for the Texas Instruments© MSP430 microcontroller. The firmware executes three main procedures, a client initialization, data acquisition, and data transmission procedure. The client initialization procedure configures the client for proper operation; the data acquisition procedure determines what data the server requested and then collects this data; and the data transmission procedure formats this acquired data and transmits it to the embedded wireless module for wireless communications with the MCETS server. Once the MCETS server receives this streaming data, a MATLAB-based graphical user interface parses, converts, and stores the acquired telemetry data for future in-depth data analysis.

The final MCETS products is a functional prototype data acquisition system that exceeds virtually all of MIT Lincoln Laboratory's system requirements. Specifically, the system has a variable data rate between one and one-hundred Hertz, improving upon the required minimum rate of ten Hertz. Furthermore, the MCETS consumes a maximum of 2.70 watts, allowing it to run on a 4.40

ampere-hour battery for seven to eight hours, weighs approximately one kilogram, and measures 2.25 x 3.45 x 4.00 inches. Overall, the most important result stemming from the MCETS is that it proves the concept of a low power and cost effective data acquisition system that employs multiple modules is both feasible and practical for the data analysis requirements of MIT Lincoln Laboratory.

Table of Contents

List of Figures

List of Tables

1. Introduction

MIT Lincoln Laboratory, a Federally Funded Research and Development Center managed by the Massachusetts Institute of Technology, is interested in developing and advancing a Multi-Client Embedded Telemetry System (MCETS) to assist in the rapid acquisition and transmission of atmospheric and kinematic data. Inherently, these systems are in distant and inaccessible environments that are too difficult, dangerous, and otherwise expensive to analyze.[1] Being entirely self-sufficient from all systems and environments it observes, a MCETS provides the real-time data link between objects in motion and a relatively central, stationary command center where measured and collected data can be processed, instantaneously viewed, and stored for future mission analysis.

The MCETS project was inspired by the growth of the United States' Ballistic Missile Defense System (BMDS), which is "one of the most complex and challenging missions of the United States' Department of Defense (DoD)".[2] Ultimately, the BMDS, a "collection of Elements and components that are integrated to achieve the best possible performance against a full range of potential [ballistic missile] threats",[3] desires a reliable, low-cost, and disposable test asset – such as the MCETS – that can support both current and future missile-defense missions. In providing this support, the MCETS could assist the DoD in accomplishing two of its seven strategic BMDS goals, including the completion of "fielding, verification, and transition of the initial BMDS capability",[4] as well as executing an "increasingly integrated and complex test program to build confidence in system support".[5]

As a result, by providing remote data acquisition capability independent of Elements and components integrated in the BMDS, the MCETS could bridge the gap between physical missions and ground-based researchers and mission analysts. Such a system of multiple wireless data acquisition modules, who seamlessly communicate with a single base station, could provide a cost-effective solution necessary for obtaining the versatile atmospheric and kinematic data needed during various flight tests. By doing so, another layer of test assets can be integrated with current methodologies to build a better, more flexible and adaptable BMDS test program.

Notes

1. "Glossary of Telemetry, Technology & Technical Terms", Texas A&M University, June 2003, <http://www.tamug.edu/labb/Technology/Glossary.htm> (26 August 2007).

2. "Global Ballistic Missile Defense", The United States Missile Defense Agency, n.d., <http://www.mda.mil/mdalink/pdf/bmdsbook.pdf> (15 September 2007).

3. Ibid.

4. Ibid.

5. Ibid.

2. System Requirements

Although the full potential and benefits of a multi-client telemetry system are not fully known, the MCETS sponsor exhibits interest in specific electrical, physical, and economical system-characteristics that are unique compared to those of similar, though more expensive, commercial off-the-shelf (COTS) telemetry units. Foremost, The Laboratory requires the MCETS to be comprised of multiple relatively low-cost modules that wirelessly communicate with a central base station. In this type of setup, several MCETS sensor modules (i.e., clients) stream live data back to a base station (i.e., server) for processing and analysis.

As well as the single-server/multiple-client setup, the sponsor also requires that the prototype system have a minimum data rate of ten Hertz. Throughout this 0.10-second reacquisition rate, The Laboratory deems it necessary to have an extremely precise position accuracy of at least one-meter using the United States' Global Positioning System (GPS). In addition, the MCETS must have a ten-minute operation time, during which the total power consumption is less than five watts. Finally, each client module in the system should have a mass of around one kilogram and a total volume of approximately one-eighth of a cubic meter (19.5 x 19.5 x 19.5 inches).

While sensors beyond a GPS receiver are not listed as requirements, it is understood that the system should provide basic atmospheric and kinematic data acquisition. With that being said, the higher the number of onboard sensors present on each MCETS module, the more flexible and expandable the system will be for applications beyond those specific to ballistic missile defense systems. Moreover, it should also be noted that specific environmental conditions including, but not limited to, extreme operating temperature, high relative humidity, and immense system vibration and shock, are not given, nor will be fully taken into account in the design and prototype of the MCETS. However, the transition from this proof-of-concept system to an actual standalone system should be nearly seamless. In fact, it is expected that most care will have to be taken in sensor output compensation, filtering, and recalibration, not in actual sensor replacement and system redesign.

Given these system requirements, the ultimate goal of this electrical and computer engineering design project is to propose, prototype, and begin testing a telemetry system with multiple wireless sensor modules that communicate with a single graphical user interface-based (GUI-based) server for MIT Lincoln Laboratory. Following the completion of this Major Qualifying Project (MQP), a successful proof-of-concept system that meets and/or exceeds the specifications given by The Laboratory will ultimately demonstrate the feasibility of a MCETS and its significant importance to the testing and analysis objectives of future flight-test campaigns.

3. The Wireless Instrumentation and Telemetry System

The current concept of a multi-client telemetry system stems from a previously designed proof-of-concept system, formerly under the name Wireless Instrumentation and Telemetry System (WITS), designed by MIT Lincoln Laboratory engineer Omar Moussa. The main concept in the WITS is to have multiple data acquisition modules (i.e., clients) with the ability to both locally store and wirelessly transmit information to a designated server. This server, a laptop for prototyping and proof-of-concept purposes, is capable of processing and storing multiple data streams, though does not have any GUI for viewing this acquired data live.

In terms of data acquisition capability, the WITS only includes two sensors – a tri-axial analog accelerometer and a GPS original equipment manufacturer (OEM) receiver. The accelerometer chosen for the design is the Colibrys© Si-Flex SF3000L (Figure 3.1: WITS Tri-Axial Analog Accelerometer), which is capable of measuring up to three times the acceleration of gravity on Earth (i.e., ±3.0 g) along the *x*-, *y*-, and *z*-axis Cartesian coordinate directions. Additionally, this sensor has at least forty-six decibels (dB) of cross-axis rejection from orthogonal axes, a sensitivity of 1.2 volts per g, and a maximum quiescent supply current of 30.0 milliamperes (mA). However, since its input voltage is between 6.0 and 15.0 volts, the accelerometer consumes a maximum of 180.0 to 450.0 milliwatts (mW) of power, a relatively high amount for a portable telemetry system.

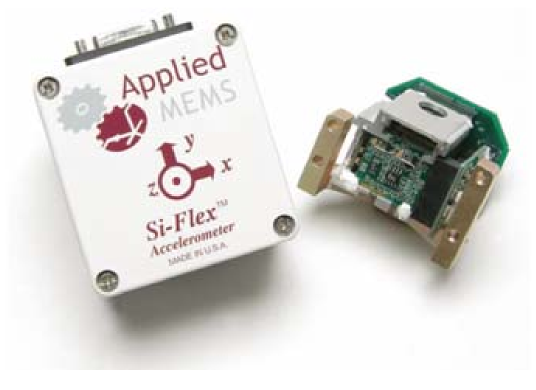


Figure 3.1: WITS Tri-Axial Analog Accelerometer
(Source: Si-Flex SF3000L Low-Noise Analog 3g Accelerometer)[1]

The GPS receiver selected for the WITS is the Javad© JNS100 (Figure 3.2: WITS GPS OEM Receiver), a sophisticated unit capable of tracking up to fifty different satellites and producing a raw data output rate up to one-hundred Hertz. Additionally, the receiver can accurately generate kinematic data at almost unlimited altitudes and velocities with 10.0-centimeter (cm) code phase and 0.1-millimeter (mm) carrier phase position accuracy. With its onboard voltage regulator, the GPS receiver accepts an unregulated voltage between 6.5 and 40.0 volts, and returns data streams on four high-speed RS-232 serial ports. Thus, with all these features available to the WITS, the system is able to incorporate and utilize a highly accurate and fast raw data GPS solution.

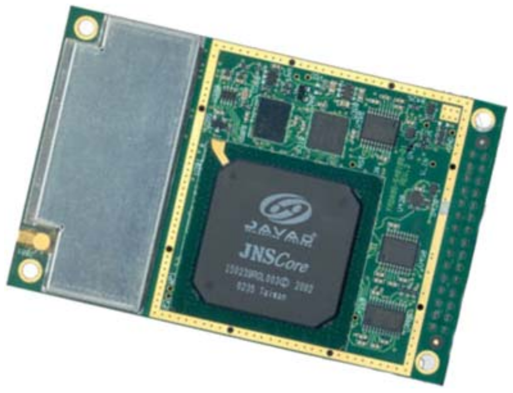


Figure 3.2: WITS GPS OEM Receiver
(Source: *Javad Navigation Systems JNS100*)[2]

In order to process the acquired data from the tri-axial accelerometer and the GPS receiver, the WITS uses a stack of PC-104 compliant boards (a size standard for computer boards) that consists of a motherboard, an input/output (I/O) peripheral board, and a PCMCIA interface socket for a wireless internet card. The motherboard incorporates an onboard Intel© 300 Megahertz (MHz) microprocessor and 512 megabytes (MB) of random access memory (RAM). Since the board is virtually a personal computer using an x86 microprocessor, the system requires both an operating system and hard drive to operate. As a result, each WITS client utilizes a four-gigabyte (GB) solid-state Flash drive and runs the Microsoft© Windows 2000 operating system. (Microsoft© Windows was chosen over tighter operating systems like Linux because, at the time of development, the drivers for the WITS's encrypted wireless network card only support Windows 2000.)

The I/O peripheral board in the three-layer PC-104 stack consists of two twelve-bit analog-to-digital converters (ADC) and a twelve-bit digital-to-analog converter (DAC). The tri-axial analog accelerometer is fed directly into one of the ADCs on the I/O board, which is programmed and controlled in the Windows operating system environment. The GPS receiver, however, utilizes an RS-232 port directly on the motherboard, and does not interface with the PC-104 I/O board. Finally, the PCMCIA socket allows for the attachment of a wireless internet card to transmit the GPS and accelerometer data to a networked server.

Notes

1. "Si-Flex SF3000l Low-Noise Analog 3g Accelerometer", Colibrys, n.d., <http://www.colibrys.com/files/e/pdf/inertial/data_sheet_siflex3000L.pdf> (6 October 2007).

2. "Javad Navigation Systems JNS100", Javad, n.d., <http://javad.com/jns/index.html?/jns/support/manuals.html> (6 October 2007).

4. The Multi-Client Embedded Telemetry System

In terms of functionality, the WITS adequately accomplishes its objectives, permitting multiple-module data acquisition and remote collection by a single server. Unfortunately, there is a significant gap between the WITS and a standalone system practical for in-flight data acquisition and analysis. The foremost problem with the system is its impractical and unacceptable power requirements. Unfortunately, in order to operate and meet the required specifications, the WITS consumes upwards of twenty watts of power, with the PC-104 motherboard and 300 MHz Intel© Celeron processor alone consuming nine to ten watts. As a result, individual system modules are virtually unsuitable to run on batteries for any significant operating time, while maintaining a small enough package (in terms of physical size and weight) necessary for onboard ballistic missile applications. Unfortunately, these pitfalls – power consumption, size, and weight – all stem from the issue that the WITS has a lot of unnecessary hardware-based overhead.

Consequently, the next-generation Wireless Instrumentation and Telemetry System contracted by MIT Lincoln Laboratory, hereby known as the Multi-Client Embedded Telemetry System, must be a complete overhaul of the current system. The most significant change that needs to be made is to use a microcontroller instead of the currently implemented Intel© Celeron microprocessor and PC-104 motherboard. By transforming from power-hungry microprocessors that (for this application) unnecessarily run at 300 MHz to ultra-low-power microcontrollers (sub-milliwatt) that can run at clock speeds in the low MHz to low kilohertz (kHz) range, the power consumption per module can be lowered by approximately nine or ten watts (virtually the entire power consumption of the currently-implemented microprocessor).

Moreover, by replacing the microprocessor with a microcontroller, the superfluous Microsoft© Windows 2000 operating system and four GB external solid-state Flash drive can be replaced with tighter application-specific embedded software and onboard microcontroller Flash memory. Additionally, the other unnecessary hardware components, including the PC-104 twelve-bit ADC and DAC board, as well as the parallel and serial communication ports on the PC-104 motherboard, can be substituted with peripherals directly on the microcontroller. And, with all of this overhead removed, ample space – in terms of power, size, and weight – remains for additional sensors not incorporated in the original WITS, including environmental analysis sensors (e.g., temperature and pressure sensors) and even a tri-axial analog gyroscope capable of measuring angular rate.

In addition to this hardware overhaul, the newly proposed MCETS builds upon the multiple-client/single-server concept, as well as the server software first developed in the WITS. Although the WITS only had one-way communications from each client module to the server, the MCETS will incorporate two-way communications via an embedded wireless module. With the ability of two-way communications, a redesigned MATLAB-based server can query particular clients already in flight based upon unique module identification numbers. This feature allows the server to ask individual clients for particular telemetry data (e.g., position, acceleration, temperature, pressure, etc.), at a requested variable data rate and length of time. Ultimately, the capability of communicating from the server to selectable clients, and back from the clients to the server, gives ground-based researchers and analysts much more flexibility and control than with the current WITS.

To summarize the proposed MCETS and to illustrate the drastic changes made from the first-generation WITS, Figure 4.1: Proposed MCETS Client Architecture depicts the overall system-level architecture for each data acquisition module in the system. The analog temperature, pressure, and tri-axial accelerometer and gyroscope sensors provide the detailed atmospheric and inertial measurements that unfortunately cannot be provided by the more accurate, RS-232-based GPS receiver. In contrast to the one inertial sensor on the WITS, these four analog sensors are tied directly into the microcontroller via a multi-channel onboard ADC. Additionally, the redesigned MCETS uses a wireless module that interfaces directly with the microcontroller via serial communications, and is capable of generating the necessary signals to communicate back and forth over an Internet Protocol-based wireless network.

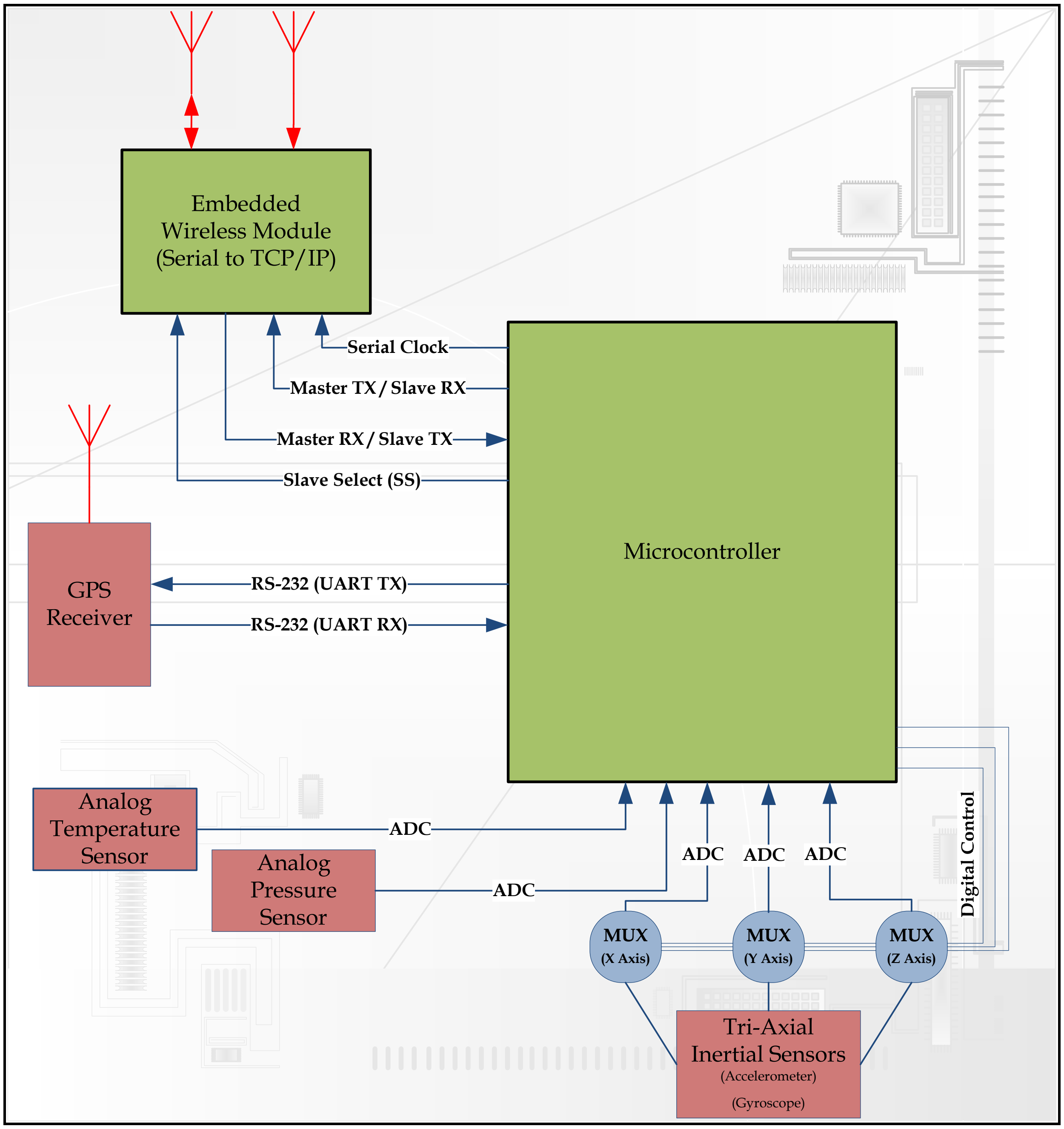


Figure 4.1: Proposed MCETS Client Architecture

On a higher system level, Figure 4.2: Proposed MCETS System Board Layout illustrates the MCETS client architecture, layer for layer. The top layer – the sensor board – is a custom-designed printed circuit board (PCB) that holds the analog sensors and the wireless module. This board has parallel communications with both the OEM GPS receiver (layer three) and the microcontroller development board (layer two), and thus acts as a gateway between the microcontroller, the GPS receiver, the analog sensors, and the wireless module. The sensor board also provides the necessary voltage lines emanating from the custom-designed power supply board (layer four) for the GPS receiver, microcontroller, and other onboard components. As a result, the sensor board is the most complex and critical aspect in the MCETS, in which the entire design relies on its full functionality.

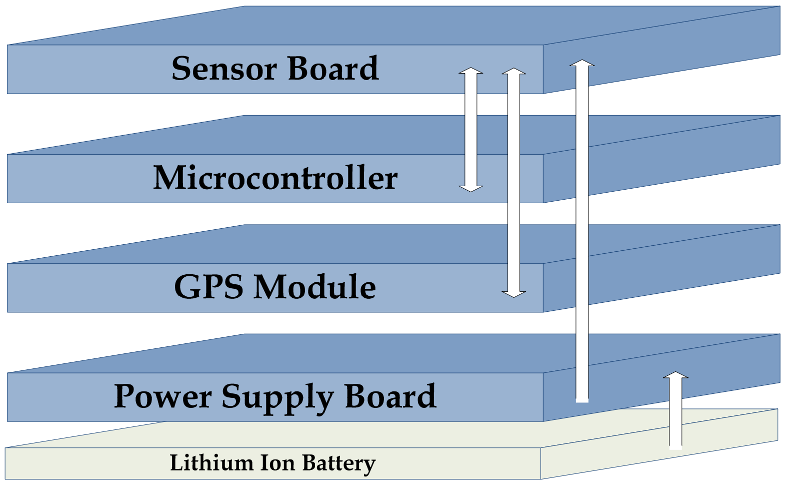


Figure 4.2: Proposed MCETS System Board Layout

With this next-generation telemetry system (Figure 4.1 and Figure 4.2), the MCETS is projected to surpass all requirements provided by The Laboratory. With respect to power requirements, it is predicted the system will consume between 2.0 and 3.0 watts of power, meeting the design requirement of at most 5.0 watts of power consumption. In comparison, the proposed MCETS consumes 12.5 of the energy of the WITS (2.5 watts compared to 20.0 watts), making battery operation more than feasible. With such a low power demand, the MCETS is more than capable of running for ten minutes, and, depending on the particular battery employed, an operating time of five or six hours is more than possible.

Furthermore, based on the anticipated size of components, the total volume of each MCETS module is projected to be around 0.00051 cubic meters (2.25 x 3.45 x 4.00 inches), far less than the 0.125 cubic meters stated in the design requirements. With respect to mass, it is hard to make an accurate estimation, however even with a battery each module should weigh around one kilogram as specified by MIT Lincoln Laboratory. Finally, regardless of the transformation from a microprocessor to a microcontroller, the new design allows for a variable data rate of one to one-hundred Hertz, which can be dynamically controlled by the MATLAB-based server, providing more control than the required minimum rate of ten Hertz. As a result, the proposed MCETS is a significant improvement over the first-generation telemetry system, WITS, and the design will help propel a relatively low-cost telemetry system from the proof-of-concept stage closer to a standalone system practical for in-flight ballistic missile data acquisition and analysis.

5. Technical Background

As illustrated in Figure 4.1, the proposed telemetry system needed to move the first-generation WITS closer to a more-practical standalone system draws upon various Electrical Engineering interfaces, protocols, and standards. One of the foremost and most accurate onboard kinematic sensors, the GPS receiver, employs the United States' Global Navigation Satellite System (GNSS), and is connected to each client microcontroller using a RS-232 cable and a Universal Asynchronous Receiver-Transmitter (UART). Similarly, the physical wireless communication link between client microcontrollers and the central MATLAB server utilizes the Serial Peripheral Interface (SPI) bus and an 802.11 wireless module. Onboard this wireless module, the use of the Transmission Control Protocol (TCP) and the Internet Protocol (IP), layers within the Internet Protocol Suite, provide the transport and network links between the client microcontrollers and the MCETS MATLAB server. Accordingly, each of these interfaces, protocols, and standards are investigated in-depth in the following subsections.

5.1. The Global Positioning System

The United States' GNSS, the Global Positioning System, is an infrastructure of twenty-four satellites set into medium-Earth orbit[1] (Figure 5.1: GPS Satellite Orbits) that permit extremely accurate object tracking and timing, including *x*, *y*, and *z*-axis position, latitude and longitude coordinates, altitude, velocity, and Coordinated Universal Time (UTC time). Originally developed by the DoD in the early 1970s and now free for both civilian and government use, the system utilizes exceptionally accurate atomic clocks that are onboard each of the twenty-four satellites.[2] With these highly accurate clocks implemented in the space segment of the GPS, the user segment of the system can advantageously use low-cost handheld receivers with far less accurate clocks that are continuously compensated for by satellite signals. For this reason, the GPS has become one of the most important systems in the world for both the United States' military and civilians across the world. Today, GPS is found in navigation systems used to coordinate military troop movement and supply shipment, civilian surveying and navigation, cellular phone networks, and even earthquake and tectonic plate measuring systems.

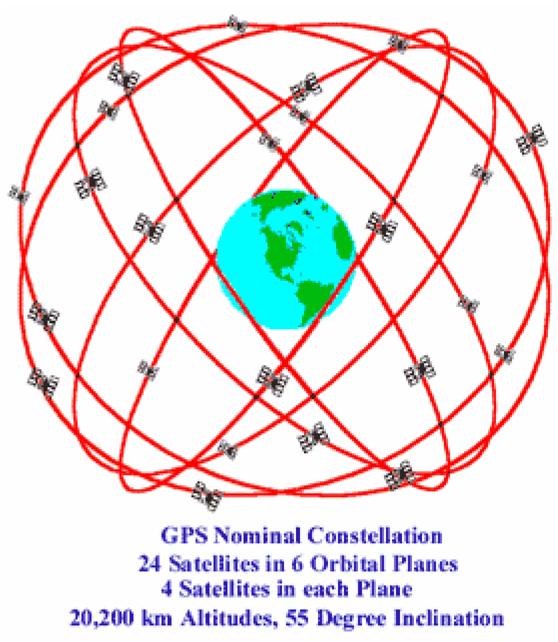

GPS Nominal Constellation
24 Satellites in 6 Orbital Planes
4 Satellites in each Plane
20,200 km Altitudes, 55 Degree Inclination

Figure 5.1: GPS Satellite Orbits
(Source: *The Navy & Satellites: Global Positioning System (GPS)*)[3]

The currently implemented GPS algorithm used in civilian GPS receivers to produce accurate timing and kinematic measurements takes advantage of the signal propagation delay from orbiting satellites. By measuring the delay from when a satellite's signal is transmitted to when it is received, the speed of light *c* can be used to calculate the distance between the GPS receiver and the satellite (i.e., $d = c \cdot \Delta t$). In order to measure this delay, a satellite creates a unique Pseudo Random Noise sequence PRN_S (Figure 5.2: Typical PRN_S Sequence Transmitted by a GPS Satellite), which is modulated by a high-frequency carrier wave and transmitted by the satellite to the GPS receiver. Once the GPS receiver receives the modulated PRN_S signal, it creates its own Pseudo Random Noise sequence PRN_R based on a predefined seed number unique to the transmitting satellite's atomic clock. This PRN_R sequence is then shifted in time to match the incoming PRN_S, and the propagation delay (i.e., time shift) is measured and used to calculate the relative distance between the satellite and the receiver.

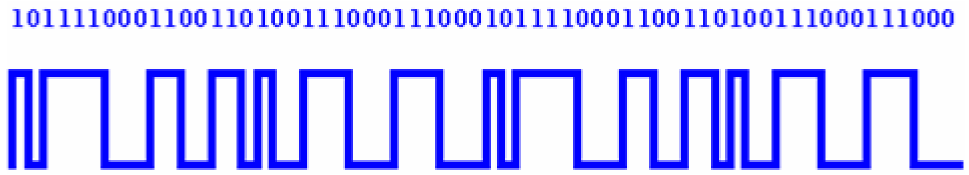


Figure 5.2: Typical PRN_S Sequence Transmitted by a GPS Satellite
(Source: *Global Positioning Systems*)[4]

As illustrated in Figure 5.3: Single-Satellite Line of Position, once a receiver knows its relative distance from one of the orbiting GPS satellites, it can infer that it is anywhere equidistant from the satellite on a line of position around the Earth (and into space). Following the connection to another satellite (Figure 5.4: Double-Satellite Line of Position), it can reduce the infinite number of possible locations calculated from the one satellite down to only two possibilities on the Earth. Similarly, a connection to a third satellite reduces the two possible receiver locations to one possible location in a two-dimensional space (i.e., latitude and longitude), and a fourth satellite to one possible location in a three-dimensional space (i.e., latitude, longitude, and altitude). By connecting to more than four satellites, a GPS receiver can calculate its three-dimensional location with a much higher degree of accuracy, a calculated value described by Geometric Dilution of Precision (GDOP).

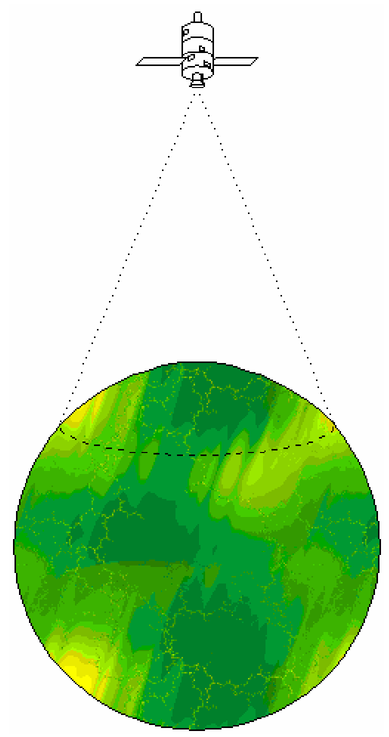


Figure 5.3: Single-Satellite Line of Position
(Source: *Navigation for Weapons*)[5]

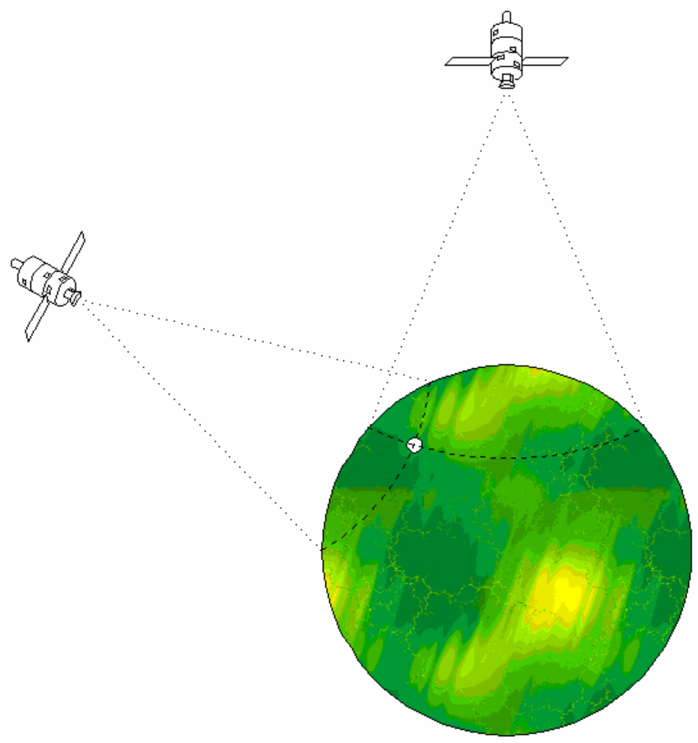


Figure 5.4: Double-Satellite Line of Position
(Source: *Navigation for Weapons*)[6]

The ideal scenario for any GPS receiver is to be connected to GPS satellites that are fully geometrically spread out in three-space, which in terms of three satellite connections is a 120-degree spread. Consequently, as illustrated in Table 5.1: GPS Dilution of Precision Values13F, when satellites are close together in space, their geometry is unfavorably weak and the GDOP value is high. On the other hand, when satellites are farther apart, their geometry is more favorable for accurate kinematic calculations and the GDOP value is low. Additionally, other accuracy parameters such as Horizontal Dilution of Precision (HDOP) for latitude and longitude accuracy, Vertical Dilution of Precision (VDOP) for altitude accuracy, Position Dilution of Precision (PDOP) for three-dimensional position accuracy, and Time Dilution of Precision (TDOP) for time accuracy, provide an insight into how reliable a GPS receiver's measurements really are.

DOP Value	Rating	Description
1	Ideal	This is the highest possible confidence level to be used for applications demanding the highest possible precision at all times.
2 – 3	Excellent	At this confidence level, positional measurements are considered accurate enough to meet all but the most sensitive applications.
4 – 6	Good	Represents a level that marks the minimum appropriate for making business decisions. Positional measurements could be used to make reliable in-route navigation suggestions.
7 – 8	Moderate	Positional measurements could be used for calculations, but the fix quality could still be improved. A more open view of the sky is recommended
9 – 20	Fair	Represents a low confidence level. Positional measurements should be discarded or used only to indicate a very rough estimate of the current location
21 – 50	Poor	Measurements are inaccurate by as much as fifty yards and should be discarded.

Table 5.1: GPS Dilution of Precision Values[7]

5.2. Parallel and Serial Communication Methods

The two most standard methods for transmitting digital information between one electrical component (e.g., a GPS receiver) and another (e.g., a microcontroller) are parallel and serial communication systems. In a parallel communication system (Figure 5.5: Parallel Communication Systems), a driver places *n* bits of data onto *n* different communication channels (e.g., wires), in which all *n* bits of information are transmitted at the same time.[8] Following the delays in the channels, all *n* bits are ideally received at exactly the same time, rendering the information ready for processing. Conversely, in a serial communication system (Figure 5.6: Serial Communication Systems), data is first converted into a serial stream of *n* bits in a process known as serialization.[9] Thereafter, a driver places the *n*-bit stream onto a single communication channel, in which each bit is sequentially transmitted. Then, following the single delay in the channel, a receiver reads the serial data and de-serializes it back into parallel form for processing.

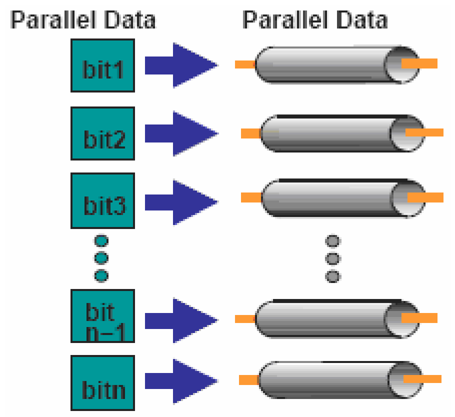


Figure 5.5: Parallel Communication Systems
(Source: *Comparing Bus Solutions*)[10]

As one can imagine, the de-serialization process in a serial communication system results in a lot of overhead compared to a similar parallel system. This overhead translates into larger time delays that accumulate due to individual encoding delays (time from when data is ready for transmission to it is actually transmitted) and decoding delays (time from when data is received to when it can actually be processed).[11] Furthermore, in order to achieve the same throughput, data must be transmitted *n* times faster in a serial system than in an equivalent parallel system. Consequently, this higher data rate causes larger signal bandwidths, ultimately increasing the cost and complexity of the channel and hindering the maximum distance between communicating devices.

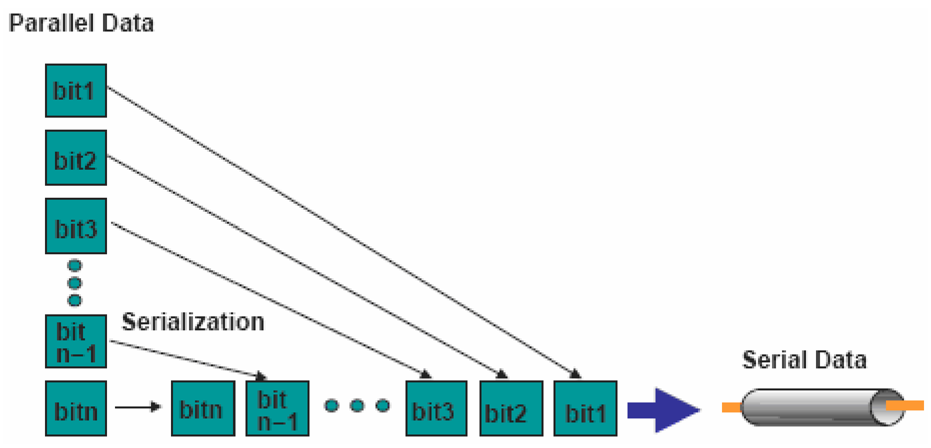


Figure 5.6: Serial Communication Systems
(Source: *Comparing Bus Solutions*)[12]

However, due to the nature of the system, serial buses require far fewer conductors than equivalent parallel buses, resulting in much smaller and less expensive systems. Moreover, since only one communication channel is needed to transmit data, serial communication systems do not have any line-to-line time skewing that can become an issue in parallel data transmission.[13] This also forces the power loss per bit of information down, as the more rapid serial signal is transmitted over less paths of resistance (only one channel compared to *n* channels) for a shorter duration of time. In other words, the higher data rate of serial communication systems result in shorter bit durations compared to parallel communications systems, ultimately resulting in less power consumption. Therefore, for embedded systems that require small packages and consume minimal power, the choice method of communications is via serial bus systems that uses only one channel per direction of information flow.

5.3. Universal Asynchronous Receivers-Transmitters

In order to interface a serial communication system with a component (e.g., a microcontroller) that naturally processes data in its parallel form, a Universal Asynchronous Receiver-Transmitter (UART) is needed to convert any parallel data to be transmitted into a serial bit stream, as well as any received serial data back into parallel form.[14] The basic functionality of these integrated circuits (ICs) is to convert back and forth between the two interfaces using digital logic. In a full-duplex system (i.e., a system that has the ability to simultaneously transmit and receive data), UARTs typically consist of a clock generator, input and output shift registers that actually perform the serial-to-parallel and parallel-to-serial conversion, read/write and transmit/receive control logic, and transmitter and receiver buffers that are used to temporarily store data before it is converted into serial and parallel forms, respectively.

UART devices, in contrast to more versatile Universal Synchronous/Asynchronous Receiver-Transmitter (USART) devices, exclusively communicate asynchronously, in which timing parameters are recovered from designated start and stop bits that are automatically embedded in the data stream. Since these characters provide framing for transmitted messages, UARTs are self-synchronizing, thus allowing data to be transferred at any given time (opposed to synchronous devices, which must constantly transmit data to maintain synchronization). When a string of binary data is written to its transmit buffer, a UART automatically appends a start bit to the beginning of the message. This bit is of the opposite polarity to the data-lines idle state (i.e., a logical '0' bit), which alerts a connected UART that data transmission has begun.

After the transmission of a start bit, five to eight bits of data are serially sent across the communication channel, least-significant bit (LSB) first, followed by an optional parity bit automatically generated by the UART.[15] Following these six to ten bits used for synchronization, data, and error checking, a stop bit, which is either one, one and a half, or two bits long, indicates the completion of a message. In actuality, the stop bit is really a minimum amount of time the signal line must be held high in the data-lines idle state before another start bit pulls the line low to initiate a new frame of data.

The optional parity bit, which follows the five to eight data bits, is as an error-checking bit used to determine error in binary data transmission. When even parity is selected, the transmitter automatically adds an extra bit (either a logical '0' or '1') to make the transmitted data packet have an even number of logical '1's. On the contrary, odd parity adds an extra bit to make the transmitted data packet have an odd number of logical '1's. Often, odd parity is more reliable than even parity because it assures that there is always at least one logical transition in the frame, allowing the UART to resynchronize itself. Unfortunately, a parity bit can only detect one bit of error and does not permit error correction like other more-sophisticated channel codes (e.g., the Hamming Code).

Before two communicating UART devices will work, they must agree on the number of data bits per frame, whether or not to add an error-checking odd or even parity bit, the number of stop bits, and most importantly, the Baud (i.e., symbol) rate. Since there is data overhead in asynchronous communications, this Baud rate does not equal the actual information throughput of the UART, which is instead between fifty-five and eighty percent of the actual Baud rate. This predefined rate, whose conventional values are listed in bits per second (bps) in Table 5.2: Standard UART Baud Rates, along with the number of data and parity bits, determines the period of a frame. Since the UART uses this information as a way to determine when certain bits should be received, the local clock must have a frequency drift of less than ten percent to assure correct bit sampling (ideally in the center of each received bit).[16]

Range	Baud Rate (bps)	Range	Baud Rate (bps)	Range	Baud Rate (bps)
Hecto-	100	Kilo-	1,200	Mega-	1,382,400
			2,400		
			4,800		
			14,400		
			19,200		1,843,200
			28,800		
			38,400		
	300		57,600		
			76,800		2,764,800
			115,200		
			230,400		
			460,800		
			921,600		

Table 5.2: Standard UART Baud Rates

5.4. The RS-232 Standard

In nearly all cases, UART devices are connected to drivers and other logic circuitry that are able to generate RS-232 compliant signals from lower-voltage, ground-based Complementary Metal–Oxide–Semiconductor (CMOS) and Transistor-Transistor Logic (TTL) signals. In 1962, the Electronic Industries Alliance (EIA) defined a standard for serial binary data signals connecting a host system (Data Terminal Equipment (DTE), such as a microcontroller) and a peripheral system (Data Circuit-Terminating Equipment (DCE), such as a GPS receiver).[17] With the purpose of ensuring complete compatibility between DTEs and DCEs, the RS-232 standard defines electrical signal characteristics (e.g., voltage levels, signaling rate, timing and slew-rates, maximum stray capacitance, line impedance, and cable length), mechanical interface characteristics, and functions for each circuit in the interface connector.

Though RS-232 is defined for both asynchronous and synchronous communication systems, the standard is almost exclusively used for asynchronous systems employing UART devices. It is here, at the UART transmitter and receiver hardware level, where the actual framing of characters (i.e., the number of start, data, parity, and stop bits used, as well as their logic values) is agreed upon. Furthermore, the standard does not define any methods of error detection or means of data compression.[18] Thus, the RS-232 standard is used solely as a way to ensure compatibility when transmitting data over a communication channel, not as a communication protocol that defines how data is formatted and interpreted.

Since RS-232 was defined prior to TTL, the standard defines valid voltage signals as ±5 volts to ±15 volts, instead of the easier to implement TTL voltage levels at +5 volts and ground. For a logical '1', historically referred to as a mark,[19] the voltage signal level is negative, and for a logical '0', often referred to as a space,[20] the voltage is positive. Typically, signal voltage levels of ±5, ±10, ±12, and ±15 volts are used depending on the power supplies available to the transmitter drivers. Moreover, RS-232 receiver drivers are sensitive to as low as ±3 volts, providing a minimum two-volt noise margin between communicating transmitters and receivers.

In order to reduce the chance of crosstalk between adjacent parallel channels, the RS-232 specification limits the maximum slew rate at the driver output to thirty volts per microsecond, and the maximum data rate to 19.2 kilobits per second (kbps).[21] Regardless of this specification, many "RS-232 compliant" devices operate at data rates in excess of 115.2 kbps and as high as 1.5 megabits per second (Mbps),[22] however they are in violation of the

RS-232 standard. A summary of these electrical specifications and others are listed in Table 5.3: RS-232 Electrical Specifications.

Electrical Parameter	Specification
Number of Devices per Line	1 Driver and 1 Receiver
Communication Mode	Full-Duplex
Maximum Cable Length	50 feet at 19.2 kbps
Maximum Data Rate	19.2 kbps
Maximum Driver Output Voltage	± 25 V
Loaded Driver Output Signal Level	±5 to ±15 V
Driver Load Impedance	3 kΩ to 7 kΩ
Maximum Slew Rate	30 V per μs
Output Current	500 mA
Receiver Input Voltage Range	±15 V
Receiver Input Sensitivity	±3 V
Receiver Input Resistance	3 kΩ to 7 kΩ

Table 5.3: RS-232 Electrical Specifications[23] [24]

In terms of mechanical interface characteristics, the standard recommends, but does not make mandatory, a D-subminiature twenty-five pin connector.[25] However, a vast majority of RS-232 compliant drivers use a smaller, D-subminiature nine pin connector, as illustrated in Figure 5.7: Standard RS-232 DB-9 Connector. As summarized in Table 5.4: RS-232 DB-9 Signal and Pin Assignments (DTE Viewpoint), for a nine-pin RS-232 cable, there are two circuits (pins two and three) for simultaneously transmitting and receiving data (permits full duplex communications), a common signal ground (pin five), two handshaking circuits (pins seven and eight), and three miscellaneous circuits (pins one, four, and nine) historically used with modem devices. For this reason, users today can asynchronously communicate with a minimal three-wire RS-232 connection, using only the ground, transmit, and receive circuits (pins five, two, and three, respectively), or if needed, with the handshaking circuits request-to-send and clear-to-send (pins seven and eight, respectively), for a five-wire connection.[26]

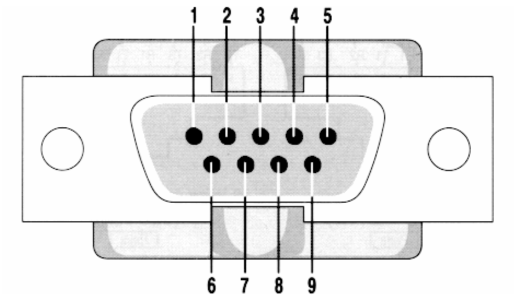


Figure 5.7: Standard RS-232 DB-9 Connector
(Source: *RS232 Tutorial on Data Interface and Cables*)[27]

Pin	Signal Type	Direction	Signal Function
1	Data Carrier Detect (DCD)	Input	Cleared by DCE when a remote connection is established
2	Received Data (RxD)	Output	Data transmitted from DCE to DTE
3	Transmitted Data (TxD)	Input	Data transmitted from DTE to DCE.
4	Data Terminal Ready (DTR)	Output	Cleared by DTE to indicate that it is ready to be connected
5	Signal Ground (G)	–	Common ground signal for the DTE and DCE
6	Data Set Ready (DSR)	Input	Cleared by DCE to indicate an active modem connection
7	Request-to-Send (RTS)	Output	Cleared by DTE to prepare DCE to receive data
8	Clear-to-Send (CTS)	Input	Cleared by DCE to accept RTS signal
9	Ring Indicator (RI)	Input	Cleared by DCE when a telephone ring is detected

Table 5.4: RS-232 DB-9 Signal and Pin Assignments (DTE Viewpoint)[28] [29]

5.5. The Serial Peripheral Interface Bus

The Serial Peripheral Interface (SPI) bus is a four-wire master/slave synchronized communications interface, originally defined by the Motorola Corporation©, which transmits binary data streams between microcontrollers or microprocessors and peripheral devices.[30] Although the interface is not an internationally- or industry-defined standard like RS-232, the SPI has become a premier method for serially communicating with multiple devices that do not need confirmation of data reception.[31] Today, the interface has become standard in devices such as ADCs, DACs, temperature and pressure sensors, LCD and USB controllers, and even EEPROM and Flash memories.[32] As a result, due to the SPI's support for full duplex communications, as well as its requirement of only four electrical traces, it has been chosen for the communication link between the microcontroller and the embedded wireless module on each of the MCETS clients.

As illustrated in Figure 5.8: Single-Master, Single-Slave SPI Implementation, the SPI only requires four traces between a master (e.g., a microcontroller) and a single slave (e.g., a peripheral device). The trace that provides the synchronization between the master and the slave is the serial clock signal SCLK. Typically ranging in frequencies from one to seventy MHz,[33] the SCLK is always driven by (i.e., the output of) the master device. As a result, the communication link is independent of any discrepancies in crystal aging and tolerance imperfections between communicating devices, a problem inherent to asynchronous interfaces such as UART. Furthermore, with only one clock source present in the system, the data rate can easily be changed by the master without any reprogramming of the slave, as is necessary in UART devices.

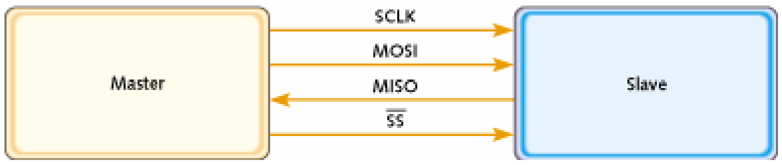


Figure 5.8: Single-Master, Single-Slave SPI Implementation
(Source: *Introduction to Serial Peripheral Interface*)[34]

The two data lines used in the SPI – MOSI and MISO – permit the exchange of binary data to and from the master device. The MOSI (master output, slave input) signal, also referred to as SIMO (slave input, master output) and SDI (serial data in) for slave devices, allows the transmission of data from the master to the slave. Similarly, the MISO (master input, slave output) signal, also referred to as SOMI (slave output, master input) and SDO (serial data out) for slave devices, allows the transmission of data from the slave to the master. Additionally, an active-low slave-select (SS) or chip-select (CS) pin activates the slave for communications, similar to start and stop bits and data framing in UART devices. This pin however, can be permanently fixed to ground in a single-slave system as long as the slave permits this type of operation. Some slaves require the falling edge (i.e., the high-to-low transition) of the SS signal to initiate an action, in which the SS grounding method does not work.[35]

From Figure 5.9: Single-Master, Multiple-Slave SPI Implementation, the extension to a multiple-slave system is made possible by using different SS signals driven by general-purpose output pins provided by the master. In this setup, the clock and data lines are shared between slaves; accordingly, every peripheral connected to the serial bus in a multi-slave system needs its own SS trace.[36] When not selected (i.e., when the SS pin is held high), the corresponding slave becomes unselected, in which its MISO trace switches to a high impedance output, thus appearing disconnected from the bus.[37] Though it is possible to use only one SS trace through the implementation of slave daisy chaining, in general, the number of slaves in a SPI setup is only limited by the number of general-purpose output pins available from the master, opposed to only one in a UART setup.

In order to setup a single-master, single-slave SPI system like the one needed on each of the MCETS clients, two different parameters – clock polarity (CPOL) and clock phase (CPHA) – need to be configured and matched between the master and the slave. The purpose of these parameters is to define the edges of the SCLK on which data bits are latched (i.e., read) and changed. As illustrated in Figure 5.10: SPI Timing Diagram, the base value for SCLK is low when the clock polarity CPOL = 0 and high when CPOL = 1. Then, if the clock phase CPHA = 0, data is latched on the first clock edge and changed on the second. Similarly, if the clock phase CPHA = 1, data is latched on the second clock edge and changed on the first.

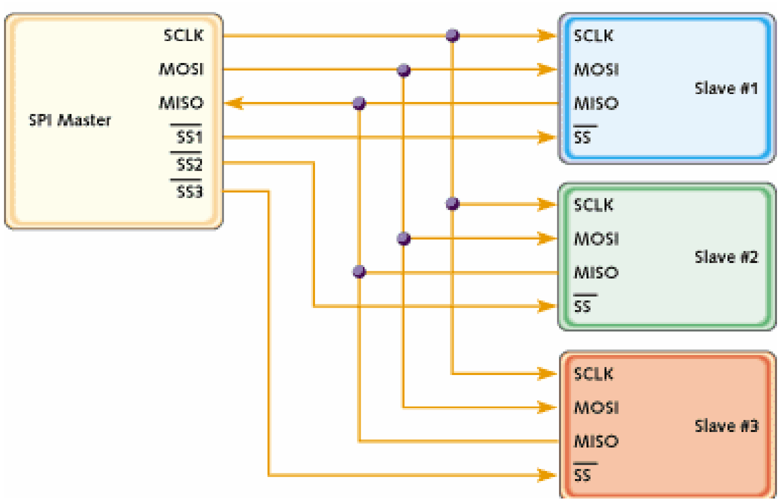


Figure 5.9: Single-Master, Multiple-Slave SPI Implementation
(Source: *Introduction to Serial Peripheral Interface*)[38]

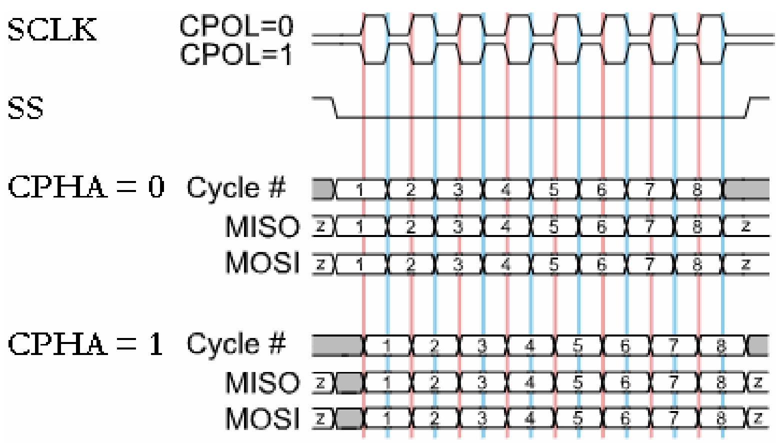


Figure 5.10: SPI Timing Diagram
(Source: *Serial Peripheral Interface Bus*)[39]

Thus, with both CPOL and CPHA having two possible states (low and high), there are four unique SPI configurations, all of which are incompatible with the other three operation modes. By convention, CPOL is considered the most significant bit (MSB) and CPHA the least significant bit (LSB). Therefore, from Figure 5.10 and Table 5.5: SPI Configuration Modes, in mode zero, data first latches on the rising clock edge (low-to-high clock transition) and then changes on the falling clock edge, and in mode one, data first changes on the rising clock edge and then latches on the falling clock edge. Similarly, in mode two, data first latches on the falling clock edge and changes on the rising clock edge, and in SPI configuration mode three, data first changes on the falling clock edge and then latches on the rising clock edge.

Mode	CPOL	CPHA	Event 1	Event 2
0	0	0	Data Latches on Rising Edge	Data Changes on Falling Edge
1	0	1	Data Changes on Rising Edge	Data Latches on Falling Edge
2	1	0	Data Latches on Falling Edge	Data Changes on Rising Edge
3	1	1	Data Changes on Falling Edge	Data Latches on Rising Edge

Table 5.5: SPI Configuration Modes

Following the CPOL and CPHA setup, once the SS pin is pulled low, data transmission between the master and the selected slave begins. During each synchronized clock cycle, a full-duplex data transmission occurs regardless of if two-way communications is desired. During this clock cycle, the master device transmits one bit of data on the MOSI trace and the slave receives it, and simultaneously, the slave transmits one bit of data on the MISO trace and the master receives it.[40] After data is serially transferred between the devices, the bits are shifted and stored into register buffers by a USART, making the data internally available for parallel processing.[41] Thus, since data is both transmitted, received, and ready for processing in one clock cycle, the data throughput of a single-master, single-slave SPI system is equal to (and therefore limited by) the SCLK provided by the master device, where a clock frequency in Hertz results an equal data rate in bits per second.

5.6. The Internet Protocol Suite

The Internet Protocol Suite, also known as TCP/IP after two of the most important protocols in the suite, TCP and IP, is a set of dozens of protocols that fully define a flexible method for communicating information over a network between a source host and a destination host. Since the suite has so many protocols, it is often separated by functionality into four-, five-, or seven-layer models. As illustrated in Figure 5.11: Five-Layer Internet Protocol Model, the most common model for the suite is a five layer stack consisting of a Physical, Data Link, Network, Transport, and Application layer. Communications is then supported between these layers (Figure 5.12: Layer-to-Layer Communications in the Internet Protocol Suite) by moving data up or down one layer, depending on whether information is being received or transmitted, respectively.

	Layer	**Function**
5	**Application**	Encapsulates data for all high-level purposes
4	**Transport**	Error and flow control
3	**Network**	Moves packets from source to destination
2	**Data Link**	Moves packets from host to host
1	**Physical**	Encodes and transmits data over a medium

Figure 5.11: Five-Layer Internet Protocol Model

The bottom layer of the Internet Protocol stack is the lowest level of the suite, responsible for the modulation and transmission of raw binary data. In this layer, data is encoded bit for bit into an electrical signal suitable for transmission through a communication channel. For this reason, the physical layer, which includes common protocols such as V.92 telephone modems, Bluetooth, 100BASE-TX, 1000BASE-T, Wi-Fi, the Institute of Electrical and Electronics Engineers (IEEE) 1294 Firewire, and DSL,[42] is the backbone of network communications, providing the physical communication link between hosts. For this unique reason, the physical layer is the only shared layer between a source and destination host, and the only layer that directly communicates with itself.

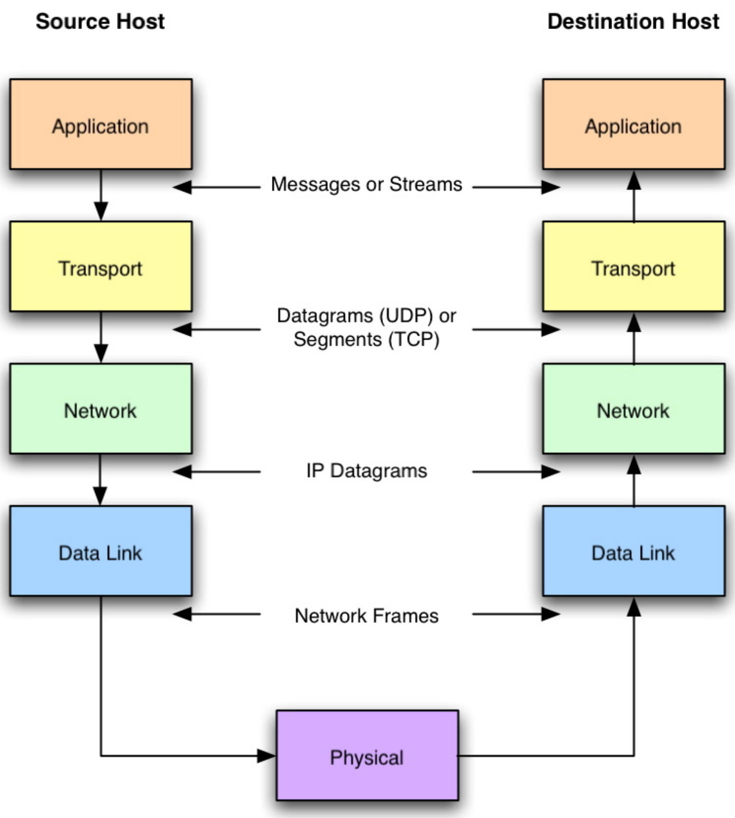


Figure 5.12: Layer-to-Layer Communications in the Internet Protocol Suite
(Source: *Novell Open Enterprise Server*)[43]

The data link layer, which includes the Point-to-Point Protocol (PPP), Ethernet, and IEEE 802.11 protocols,[44] is somewhat related and dependent upon the physical layer, and performs final data packaging before passing it to the physical layer for transmission. As a whole, the data link layer provides the final encapsulation of data to ensure that information arrives to intended devices properly. This functionality, however, can be visualized as consisting of two sub-layers, the Media Access Control (MAC) layer and the Logical Link Control (LLC) layer.

The MAC sub-layer of the data link layer defines procedures for multiple network devices to share a single communication channel. Often, several network devices physically transmit data through a single medium, an Ethernet cable for instance, in which the MAC sub-layer guarantees that there are no conflicts in data transfer. Accordingly, each device on a network has a specific hardware or MAC address that the data link layer uses to package and send data through the physical layer.[45] In order to effectively communicate with the network layer (the third layer in the five-layer model), the LLC sub-layer multiplexes and de-multiplexes information from the MAC sub-layer. This interfacing scheme ultimately allows more network layer technologies to work smoothly with the MAC sub-layer.

The third layer in the five-layer Internet Protocol Suite model is the first truly abstract layer, which defines how different interconnected networks operate. In comparison to the data link layer, the network layer governs the connection of devices regardless of the network in which they reside; the data link layer, however, governs the physical connection of devices on particular networks. Common examples of network layer protocols include the Internet protocols IPv4 and IPv6, Internetwork Packet Exchange (IPX), and the Datagram Delivery Protocol (DDP).

Specifically, the network layer performs a few important functions in the transfer of data across a network, including addressing and switching. In terms of addressing, network layer protocols define addressing standards completely independent of hardware for every machine on every network, compared to data link protocols that limitedly define addressing standards on a single network. Additionally, the network layer is responsible for moving data across internetworks by receiving data from numerous sources and sending that data to its proper final destination. As illustrated in Table 5.6: Internet Protocol (IPv4) Header, all of this is accomplished by encapsulating data received from the transport layer (the second layer in the five-layer model) into a packet with a standardized (typically 192-bit) header.[46] Furthermore, in order to facilitate the needs of data link protocols that have limits on the size of a packet, the network layer governs the division and reassembly of data packets.

Bits						
0 – 31	Version	Internet Header Length	Type of Service	Total Length		
32 – 63	Identification			Flags	Fragment Offset	
64 – 97	Time to Live		Protocol	Header Checksum		
96 – 127	Source IP Address					
128 – 159	Destination IP Address					
160 – 192	Options and Padding					

Table 5.6: Internet Protocol (IPv4) Header[47]

Like the network layer, the transport layer is an abstract, conceptual layer having no direct relationship with hardware devices. It separates itself from the network layer and the layers below it in the sense that it is not concerned with getting data from one physical location to another. Rather, the transport layer protocols govern the transmission of data from one application process to another. Accordingly, the transport layer protocols perform much of the same functions for application-to-application transmission of data, as the network layer protocols perform for device-to-device communications. Common examples of transport layer protocols include the User Datagram Protocol (UDP), Transmission Control Protocol (TCP), Datagram Congestion Control Protocol (DCCP), and the Stream Control Transmission Protocol (SCTP).[48]

Similar to how the network and data link layers address data for network and internetwork communications, the transport layer addresses data to specific software applications. Additionally, the layer also multiplexes and de-multiplexes data streams from many different software applications, therefore guaranteeing that layers below the transport layer only see one stream of data, regardless of the number of communicating applications on a device. Finally and most importantly, the transport layer is responsible for establishing, maintaining, managing, and terminating the connection between two devices, as well as controlling the data rate and providing procedures to ensure reliable transmission of data. All of this is accomplished with a standardized (typically 192-bit) header (Table 5.7: Transmission Control Protocol (TCP) Header55F), similar to the IPv4 header in the network layer.

Bits					
0 – 31	Source Port				Destination Port
32 – 63	Sequence Number				
64 – 97	Acknowledgement Number				
96 – 127	Offset	Reserved	ECN	Control Bits	Window
128 – 159	Checksum				Urgent Pointer
160 – 192	Options and Padding				

Table 5.7: Transmission Control Protocol (TCP) Header[49]

The top of the Internet Protocol stack is the application layer, which is responsible for translating data in an application-specific format to one that is compatible with the transport layer. The most well known application-layer protocols include Secure Shell (SSH), File Transfer Protocol (FTP), HyperText Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), Post Office Protocol (POP), Internet Message Access Protocol (IMAP), and Simple Object Access Protocol (SOAP).[50] Most commonly, these application protocols compress and decompress data, as well as provide an encryption and decryption method to improve the security of information as it propagates through the other four Internet Protocol layers. The application layer also handles the translation of information from different operating systems and software applications, so that programs can seamlessly communicate data regardless of their particular formats.

Overall, the five-layer Internet Protocol Suite is the accumulation of a complex interconnected system used to provide reliable communications between source and destination hosts. Fortunately, there are commercially available embedded modules that permit communications between microcontrollers and an internet-like network using the five-layer stack illustrated in Figure 5.11. These modules are capable of taking serial data from a SPI bus (e.g., an application layer) and interfacing it with a TCP transport layer (which adds the header in Table 5.7), an internet-employed IPv4 network layer (which adds the header in Table 5.6), an IEEE 802.11 data link layer (Table 5.8: IEEE 802.11 Data Link Layer Specifications57F), and finally an over-the-air Wi-Fi physical layer. Ultimately, this layered system allows data to be streamed from the MCETS client microcontrollers and over a local area network to a MATLAB-based server.

Protocol	Release Date	Frequency Band	Data Rate		Maximum Outdoor Range
			Typical	Maximum	
Legacy	1997	2.4 – 2.5 GHz	0.9 Mps	2 Mbps	~100 Meters
802.11b	1999	2.4 – 2.5 GHz	4.3 Mbps	11 Mbps	~140 Meters
802.11g	2003	2.4 – 2.5 GHz	19 Mbps	54 Mbps	~140 Meters
802.11a	1999	5.15 – 5.25 GHz 5.25 – 5.35 GHz 5.49 – 5.725 GHz 5.725 – 5.85 GHz	23 Mbps	54 Mbps	~120 Meters
802.11n	~ 2008 (Draft 2.0)	2.4 GHz and/or 5 GHz	74 Mbps	248 Mbps	~ 250 Meters
802.11y	~ 2008 (Draft 4.0)	3.65 – 3.7 GHz	23 Mbps	54 Mbps	~ 5000 Meters

Table 5.8: IEEE 802.11 Data Link Layer Specifications[51]

Notes

1. Casey L. Larijani, GPS For Everyone, (New York: American Interface Corporation, 1998), 3-5.
2. Casey L. Larijani, GPS For Everyone, 274.
3. "The Navy & Satellites: Global Positioning System (GPS)", The United States Navy, n.d., <http://www.onr.navy.mil/Focus/spacesciences/satellites/gps.htm> (9 September 2007).
4. D.J. (Dave) Sauchyn, "Global Positioning Systems" The University of Regina, n.d., <http://uregina.ca/~sauchyn/geog411/global_positioning_systems.html> (9 September 2007).
5. "Navigation for Weapons", Federation of American Scientists, n.d., <http://www.fas.org/man/dod-101/navy/docs/es310/GPS/GPS.htm> (9 September 2007).
6. Ibid.
7. Jon Person, "Mastering GPS Programming: Part Two", GeoFrameworks, n.d., <http://www.geoframeworks.com/Articles/WritingApps2_3.aspx> (6 October 2007).
8. Georg Becke, "Comparing Bus Solutions", Texas Instrument, February 2004, <http://focus.ti.com/lit/an/slla067a/slla067a.pdf> (5 September 2007).
9. Ibid.
10. Ibid.
11. Ibid.
12. Ibid.
13. Ibid.
14. Frank Durda, "The UART: What it is and how it works", 13 January 1996, <http://www.freebsd.org/doc/en_US.ISO8859-1/articles/serial-uart/index.html#UART> (5 September 2007).
15. Frank Durda, "The UART: What it is and how it works".
16. Frank Durda, "The UART: What it is and how it works".
17. "Fundamentals of RS-232 Serial Communications", Maxim IC, 29 March 2001, <http://pdfserv.maxim-ic.com/en/an/AN83.pdf> (6 September 2007).
18. Ibid.
19. Ibid.
20. Ibid.
21. Ibid.
22. Ibid.
23. Ibid.
24. "RS232 Tutorial on Data Interface and Cables", ARC Electronics, n.d., <http://www.arcelect.com/rs232.htm> (7 September 2007).
25. "RS-232 Serial Interface Pinout", 25 June 2006, <http://pinouts.ru/SerialPorts/RS232_pinout.shtml> (5 September 2007).
26. Wikipedia Contributors, "RS-232", Wikipedia, The Free Encyclopedia, 4 September 2006 <http://en.wikipedia.org/w/index.php?title=RS-232&oldid=155696305> (7 September 2007).

27. “RS232 Tutorial on Data Interface and Cables”, ARC Electronics.

28. Ibid.

29. Wikipedia Contributors, “RS-232”, Wikipedia, The Free Encyclopedia.

30. “Serial Buses Information Page: SPI”, n.d., <http://www.epanorama.net/links/ serialbus.html#spi> (3 September 2007).

31. Kalinsky, David, and Roee Kalinsky, “Introduction to Serial Peripheral Interface”, Embedded Systems Design, 1 February 2002, <http://embedded.com/columns/beginerscorner/9900483?printable=true> (2 September 2007).

32. Martin Schwerdtfeger, “SPI – Serial Peripheral Interface”, June 2000, <http://www.mct.net/faq/ spi.html> (2 September 2007).

33. Wikipedia Contributors, “Serial Peripheral Interface”, Wikipedia, The Free Encyclopedia, 6 September 2007, <http://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&oldid=156061198> (3 September 2007).

34. Kalinsky, David, and Roee Kalinsky, “Introduction to Serial Peripheral Interface”, Embedded Systems Design.

35. Wikipedia Contributors, “Serial Peripheral Interface”, Wikipedia, The Free Encyclopedia.

36. “Serial Buses Information Page: SPI”.

37. Martin Schwerdtfeger, “SPI – Serial Peripheral Interface”.

38. Kalinsky, David, and Roee Kalinsky, “Introduction to Serial Peripheral Interface”, Embedded Systems Design.

39. Ibid.

40. Ibid.

41. Martin Schwerdtfeger, “SPI – Serial Peripheral Interface”.

42. Wikipedia Contributors, “Physical Layer”, Wikipedia, The Free Encyclopedia, 6 September 2007, <http://en.wikipedia.org/w/index.php?title=Physical_layer&oldid=156014135> (8 September 2007).

43. “Novell Open Enterprise Server”, Novell, 1 June 2005, <http://www.novell.com/documentation/oes/ pdfdoc/tcpipenu/tcpipenu.pdf> (9 September 2007).

44. Wikipedia Contributors, “Data Link Layer”, Wikipedia, The Free Encyclopedia, 16 August 2007, <http://en.wikipedia.org/w/index.php?title=Data_link_layer&oldid=151664494> (8 September 2007).

45. Charles M. Kozierok, “Data Link Layer (Layer 2)”, The TCP/IP Guide, 20 September 2005, <http:// www.tcpipguide.com/free/t_DataLinkLayerLayer2.htm> 11 September 2007.

46. “IP, Internet Protocol”, Network Sorcery, n.d., <http://www.networksorcery.com/enp/protocol/ ip.htm> (10 September 2007).

47. Ibid.

48. Wikipedia Contributors, “Transport Layer”, Wikipedia, The Free Encyclopedia, 7 September 2007, <http://en.wikipedia.org/w/index.php?title=Transport_layer&oldid=156366144> (8 September 2007).

49. "TCP, Transmission Control Protocol", Network Sorcery, n.d., <http://www.networksorcery.com/enp/protocol/tcp.htm> (10 September 2007).

50. Wikipedia Contributors, "Application Layer", Wikipedia, The Free Encyclopedia, 3 September 2007, <http://en.wikipedia.org/w/index.php?title=Application_layer&oldid=155347969> (9 September 2007).

51. Wikipedia Contributors, "IEEE 802.11", Wikipedia, The Free Encyclopedia, 8 September 2007, <http://en.wikipedia.org/w/index.php?title=IEEE_802.11&oldid=156466248> (9 September 2007).

6. MCETS Client Hardware Design

Having a proposed system that is capable of meeting the contract design requirements, as well as the technical background knowledge needed to execute the design successfully, the next step in the advancement of the MCETS is to design and physically prototype the four-layer client modules. Overall, the hardware design consists of five major phases, which together will progress the proposed design to a tangible system ready for testing and implementation. In order, the five phases of the hardware design, which are discussed in the following subsections, are

1. Selecting an appropriate microcontroller development board;
2. Selecting an analog temperature, pressure, and tri-axial inertial sensor, as well as an RS-232 compliant GPS receiver, and interfacing them with one another and the microcontroller development board;
3. Selecting and interfacing an 802.11 embedded wireless module that uses the SPI bus;
4. Designing a power supply board that can provide adequate power to the sensor, microcontroller development, and GPS receiver boards; and
5. Designing and assembling a custom PCB for the sensor and power supply boards.

6.1. Olimex© MSP430-P1611 Development Board

Due to the rapid advancement of digital- and computer-based products, there are literally thousands of different low-cost microcontrollers capable of providing the sensor data processing needed in the MCETS. However, none stands out in terms of flexibility, ultra-low-power consumption, and the number of peripheral devices as the Texas Instruments© MSP430 Series. This series of microcontrollers is capable of executing as many as eight million instructions per second (MIPS) using a master clock frequency of eight MHz and a supply voltage between 1.8 and 3.6 volts. Additionally, these controllers can operate in any of five different software selectable power-saving modes that manipulate the status of the central processing unit (CPU) and three internal clock signals. And, with the ability to wake up from any one of these standby modes in less than six microseconds, MCETS clients can maximize their time in low power modes while only using the high-frequency active mode for the data acquisition and transmission processes.

The MSP430 series of microcontrollers uses a von-Neumann style architecture, in which there is a common sixteen-bit memory address and data bus. In this type of setup, all program memory, interrupt vectors, data memory, and peripheral devices share a common bus structure into the CPU. Being a sixteen-bit reduced instruction set computer (RISC), the CPU contains sixteen fully addressable, single-cycle registers able to store sixteen bits of data. Combining these ultra-fast memories with twenty-seven basic instructions and seven different addressing modes, the MSP430 facilitates maximizing both processing and code efficiency. Ultimately, these features permit tight code capable of acquiring, packaging, and transmitting data from the MCETS sensors at a relatively high throughput.

Within the MSP430 series there are around 130 different microcontrollers, each featuring a different arrangement of memory types, memory sizes, and communication and peripheral devices. As illustrated in Table 6.1: The Texas Instruments© MSP430 Microcontroller Series, there are a number of devices suitable for the MCETS; however, the microcontroller that stands out the most for this application is the MSP430-F1611 microcontroller. This Flash-memory based controller encompasses 48 kilobytes (KB) of Flash memory and 10 KB of RAM, primarily used for code and program data, respectively. Since the purpose of the MCETS it acquire a fairly large amount of binary data, the MSP430-F1611 with its significant 10 KB of RAM is appropriate for the data acquisition and temporary data storage needs of each client module.

Part Number	Flash	ROM	RAM	ADC	I/O	Integrated Peripherals	Interface
MSP430-FG4619	120 KB	-	4 KB	12-bit SAR	80	12-bit DAC (2) Operational Amplifier (3) Analog Comparator (1) DMA Controller (1) Hardware Multiplier (1) Watchdog Timer (1) 16-bit Timer (2) 8-bit Timer (2) LCD Segments (160)	USART (1)
MSP430-F447	32 KB	-	1 KB	12-bit SAR	48	Analog Comparator (1) Hardware Multiplier (1) Watchdog Timer (1) 16-bit Timer (2) 8-bit Timer (2) LCD Segments (160)	USART (2)
MSP430-F1611	**48 KB**	**-**	**10 KB**	**12-bit SAR**	**48**	**12-bit DAC (2)** **Analog Comparator (1)** **DMA Controller (1)** **Hardware Multiplier (1)** **Watchdog Timer (1)** **16-bit Timer (2)**	**USART (2)**
MSP430-CG4618	-	116 KB	8 KB	12-bit SAR	80	Analog Comparator (1) 12-bit DAC (2) DMA Controller (1) Operational Amplifier (3) Watchdog Timer (1) 16-bit Timer (2) LCD Segments (160)	USART (1)
MSP430-F1612	55 KB	-	5 KB	12-bit SAR	48	12-bit DAC (2) Analog Comparator (1) DMA Controller (1) Hardware Multiplier (1) Watchdog Timer (1) 16-bit Timer (2)	USART (2)
MSP430-F133	8 KB	-	256 B	12-bit SAR	48	Analog Comparator (1) Watchdog Timer (1) 16-bit Timer (2)	USART (1)

Table 6.1: The Texas Instruments© MSP430 Microcontroller Series[1]

Additionally, the MSP430-F1611 has an eight-channel, twelve-bit successive-approximation-register (SAR) ADC, capable of sampling input voltages up to the supply voltage (1.8 to 3.6 volts) at a rate of 200 kilosamples per second (ksps). A three-channel direct memory accesses (DMA) controller also allows data to be transferred from one address to another, without CPU intervention. This helps increase the throughput of other peripheral modules (including the UART, SPI bus, and ADC), while reducing the number of executed CPU instructions and the overall system power consumption. Furthermore, the microcontroller incorporates two USART peripheral devices with independent receive and transmit interrupts vectors. The two devices support serial communications, both asynchronously in UART mode and synchronously in SPI mode. Finally, the MSP430-F1611 has two sixteen-bit timers (Timer A and Timer B) with extensive interrupt capability, as well as forty-eight general-purpose I/O pins that can individually be read or written to.

Having selected a microcontroller appropriate for each of the MCETS client modules, the next step is to choose a development board that includes both the Texas Instruments© MSP430-F1611 microcontroller and an RS-232 driver capable of interfacing with the UART peripheral on the microcontroller and the GPS receiver. The leading developer of MSP430 development boards is Olimex©, which conveniently sells an MSP430-P1611 development board that has both a MSP430-F1611 microcontroller and a DB-9 female RS-232 port and driver. This driver, which interfaces the RS-232 signal from the GPS receiver with the UART peripheral on the microcontroller, is guaranteed to transmit and receive at data rates up to 350 kbps while maintaining the RS-232 output voltage levels defined in the standard. Running on 3.3 volts, the driver draws a maximum current of 1.0 mA, for a total power consumption of 3.3 mW.

The development board also has an onboard power supply jack for alternating current (AC) and direct current (DC) voltage sources between 4.5 and 6.0 volts. As illustrated in the datasheet in Appendix A.1: Olimex© MSP430-P1611 Development Board Datasheet, following a bridge rectifier, the incoming supply voltage is filtered and steppeddown using a linear voltage regulator. Nevertheless, due to the unreasonable efficiency of linear voltage regulators for power sensitive systems like the MCETS, the development board will be externally powered at 3.3 volts from the power supply board through the development board's sixty-pin extension header. (For the full details of how the board is powered, as well as how it is interfaced with the sensor board, see Appendix B.1: Miscellaneous Header and Connector Pin Connections, and Appendix C.1: Miscellaneous Header and Connector Circuit Schematic.)

Additional features that make the MSP430-P1611 development board perfect for each MCETS client are its small 100-by-80 millimeter (mm) board size and JTAG male connector for Flash programming and system debugging. The board also incorporates a light emitting diode (LED) to illustrate that the microcontroller has a sufficient supply voltage, a reset button to restart the microcontroller, and a programmable button and LED for miscellaneous use. Finally, the development board has a standard 32.768 kHz low-frequency crystal oscillator, as well as a crystal socket and capacitor solder pads for an additional high-frequency oscillator. As a result, each MCETS microcontroller board incorporates an added 6.0 MHz high-frequency crystal oscillator, used to clock the processing-intensive data acquisition and transmission instructions.

6.2. Maxim© DS600U Analog-Output Temperature Sensor

The temperature sensor selected for each MCETS client, the Maxim© DS600U, is truly a unique sensor with numerous appealing features for a low power, embedded system. Foremost, this analog-output temperature sensor provides an extremely accurate factory-calibrated temperature measurement through an exposed thermal conducting pad, all within a three-by-five millimeter (mm) surface mount package. Specifically, the accuracy is ±0.5°C over the temperature range -20°C to 100°C (±0.9°F over the range -4°F to 212°F), and ±0.75°C over the extended ranges -20°C to -40°C and 100°C to 125°C (±1.35°F over the ranges -4°F to -40°F and 212°F to 257°F). This accuracy is valid over the device's entire operating voltage range of 2.7 to 5.5 volts, making it very appealing for microcontroller based systems that run on low voltages around 3.3 volts.

As illustrated in Figure 6.1: Temperature versus Output Voltage, the DS600U eight-pin temperature sensor outputs a voltage proportional to the temperature on the thermal conducting pad, where

$$T = \frac{v_{out} - 0.509}{6.45 \times 10^{-3}} \quad [°C], \tag{6.1}$$

Since the output voltage range is between 251 and 1,315 millivolts (mV), corresponding to the two extreme temperatures -40°C and 125°C (125°F and 257°F), respectively, the output of the temperature sensor (pin four) is directly tied into the high-impedance ADC input of the microcontroller (pin fifty-eight (P6.7/A7) on the sixty pin microcontroller header). (For the full details of how this sensor interfaces with the MCETS PCB boards, see Appendix B.2: Temperature Sensor Pin Connections and Appendix C.2: Temperature Sensor Circuit Schematic.)

The final two features that make the DS600U a perfect analog sensor for the MCETS are its very low power requirements and its thermometer shutdown feature. In terms of maximum supply current, the device draws a maximum of 140 µA, which in terms of power when using a 3.3 volt supply, is a mere 462 microwatts (µW). Additionally, the temperature sensor incorporates an active-high shutdown feature that turns the thermal sensor off when the shutdown pin (pin six) is pulled high by the microcontroller. In this mode of operation, the supply current drops from 140 µA to 2.5 µA, resulting in a maximum power dissipation of 8.25 μW. As a result, when the temperature sensor is not in use, the microcontroller can put it into low power mode, saving more than 450 μW of power. (For additional electrical and mechanical specifications, as well as absolute maximum ratings, see Appendix A.2: Maxim© DS600U Analog-Output Temperature Sensor Datasheet.)

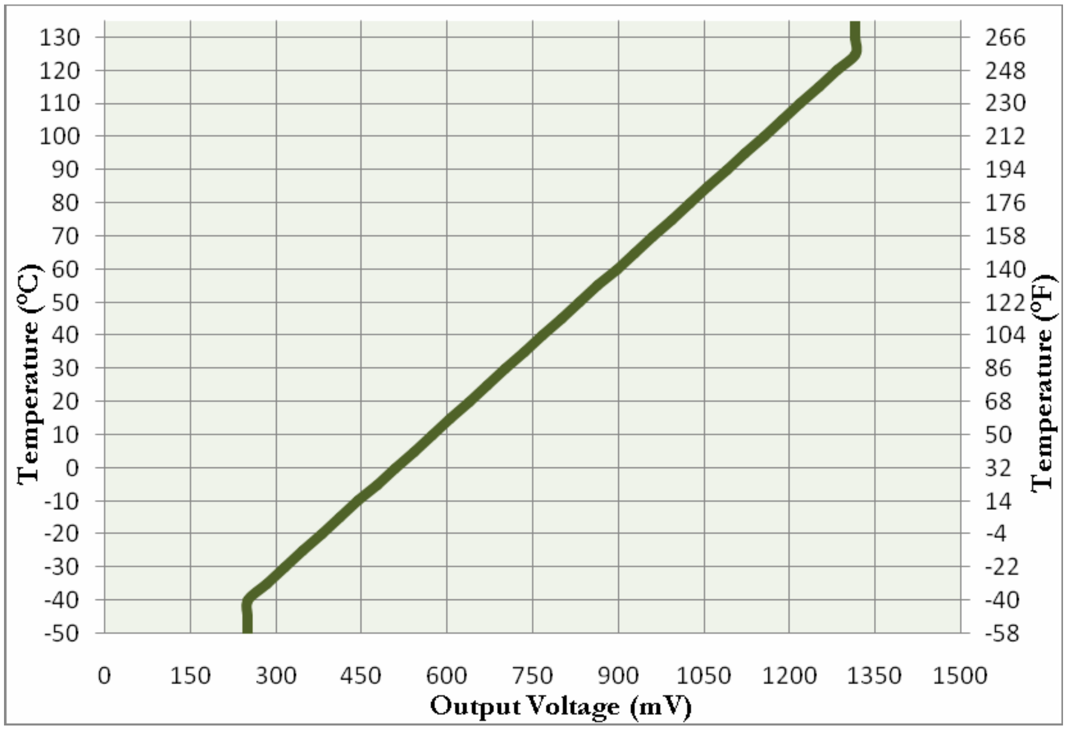


Figure 6.1: Temperature versus Output Voltage

6.3. Motorola© MPXA4250A6U Pressure Sensor

Unlike temperature sensors, it is very difficult to find a relatively small, accurate analog pressure sensor that is factory-calibrated, temperature compensated, and able to measure absolute pressure (i.e., pressure with respect to a sealed vacuum). However, Motorola© manufactures a ten-by-eighteen mm surface mount pressure sensor that is calibrated and temperature compensated from -40°C to 125°C (-40°F to 257°F), adhering to the exact temperature range of the Maxim© DS600U temperature sensor. With an accuracy of approximately ±0.5 pounds per square inch (psi) in the temperature range 0°C to 85°C (32°F to 185°F), the Motorola© MPXA4250A6U is capable of measuring absolute pressure from 2.9 to 36.3 psi. Outside of this range, between 0°C and -40°C (32°F to -40°F) and 85°C and 125°C (185°F to 257°F), the accuracy decreases linearly from ±0.5 psi to approximately ±1.5 psi as illustrated in Figure 6.2: Pressure Sensor Accuracy versus Temperature.

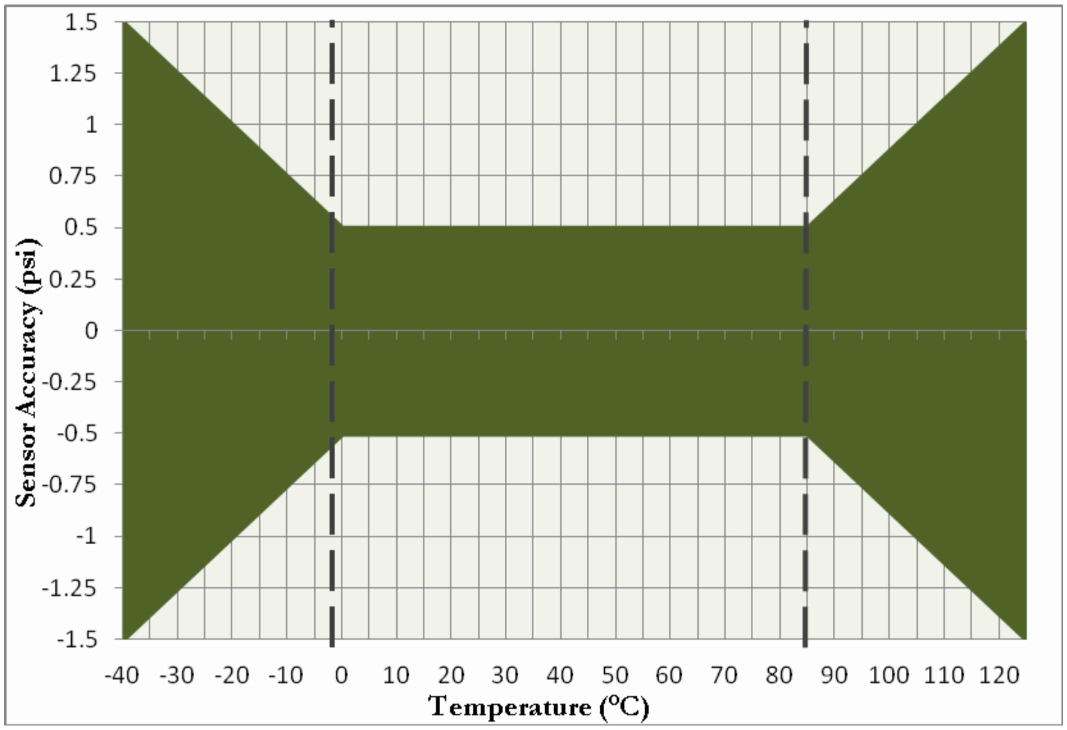


Figure 6.2: Pressure Sensor Accuracy versus Temperature

The supply voltage and current requirements of the MPXA4250A6U, however, greatly exceed those of the DS600U temperature sensor, requiring a steady voltage of around 5.0 volts and a maximum supply current of 10.0 milliamperes (mA). Thus, this particular pressure sensor can consume as much as 50.0 mW of power, all while using a non-microcontroller compatible voltage of 5.0 volts. As a result, each MCETS client now requires two separate voltage lines – 3.3 volts and 5.0 volts – capable of efficiently delivering

450 μW and 50.0 mW of power, respectively. Nonetheless, due to the pressure sensors relatively good accuracy and small size, these electrical shortcomings are considered more than adequate tradeoffs for the corresponding pressure-sensing performance gain. (For supplementary specifications, as well as how the pressure sensor is interfaced with the rest of the MCETS, see Appendix A.3: Motorola© MPXA4250A6U Pressure Sensor Datasheet, and Appendix B.3: Pressure Sensor Pin Connections, respectively.)

Since the MPXA4250A6U pressure sensor produces a ratiometric output voltage that is dependent upon the 5.0 volt input voltage, the sensor can produce an output voltage between 0.2 and 4.8 volts. From Figure 6.3: Pressure versus Output Voltage, this output voltage relates to an applied absolute environmental pressure, where with a sensitivity of 138 mV per psi,

$$P = 0.145038\left(\frac{250v_{out}}{v_s} + 10\right) = 0.145038(50v_{out} + 10) \quad [\text{psi}]. \tag{6.2}$$

Therefore, unlike the temperature sensor, the pressure sensor cannot directly interface with the ADC on the microcontroller, which can only accept voltages up to the 3.3-volt supply.

As illustrated in Figure 6.4: Single-Supply Op-Amp Attenuator Circuit, in order to overcome this incompatibility a voltage divider is needed to reduce the pressure sensor output voltage from 4.8 volts to at most 3.3 volts. Unfortunately, the traditional and more efficient method of reducing a voltage using an inverting operational amplifier (op-amp) attenuator circuit is impractical because of the need for a negative supply voltage. As a result, 0.1-percent tolerance 30 kiloohm (kΩ) and 20 kΩ resistors are used to reduce the pressure sensor output voltage to an acceptable ADC input voltage, where

$$v_{ADCIN} = v_{POUT}\left(\frac{R_2}{R_1 + R_2}\right) = v_{POUT}\left(\frac{30k}{20k + 30k}\right) = 0.6 \cdot v_{POUT} \tag{6.3}$$

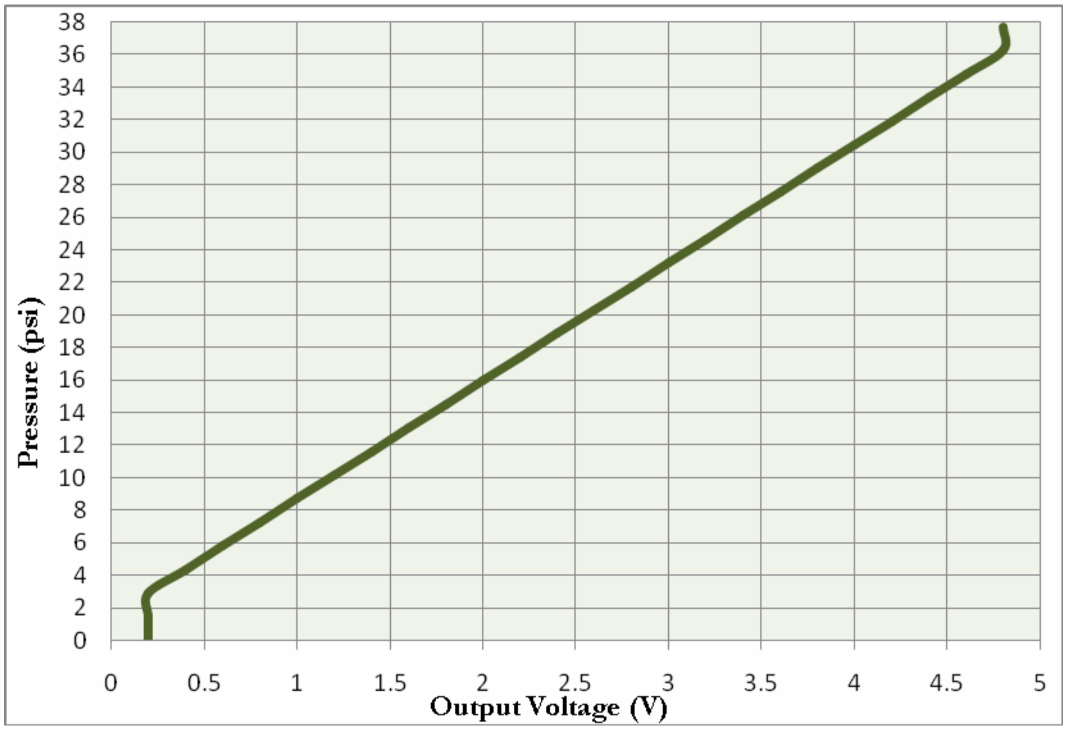


Figure 6.3: Pressure versus Output Voltage

To ensure minimal distortion and that enough current can be supplied by the pressure sensor to the resistors (the MPX4250A6U can only source around 100 µA), an op-amp buffer circuit is used to isolate the sensor from the voltage divider network. With this circuit configuration, a maximum of 100 µA is drawn from the 5.0-volt supply rail, not the pressure sensor, when the output voltage of the sensor is 5.0 volts. Therefore at maximum pressure and a supply voltage of 5.0 volts, the op-amp attenuator circuit will dissipate 1150 µW from the actual op-amp IC and 500 µW from the two series resistors. Combining this with the power dissipation of the actual pressure senor (50 mW), the entire pressure sensor configuration (Appendix C.3: Pressure Sensor Circuit Schematic) consumes a maximum of 51.65 mW.

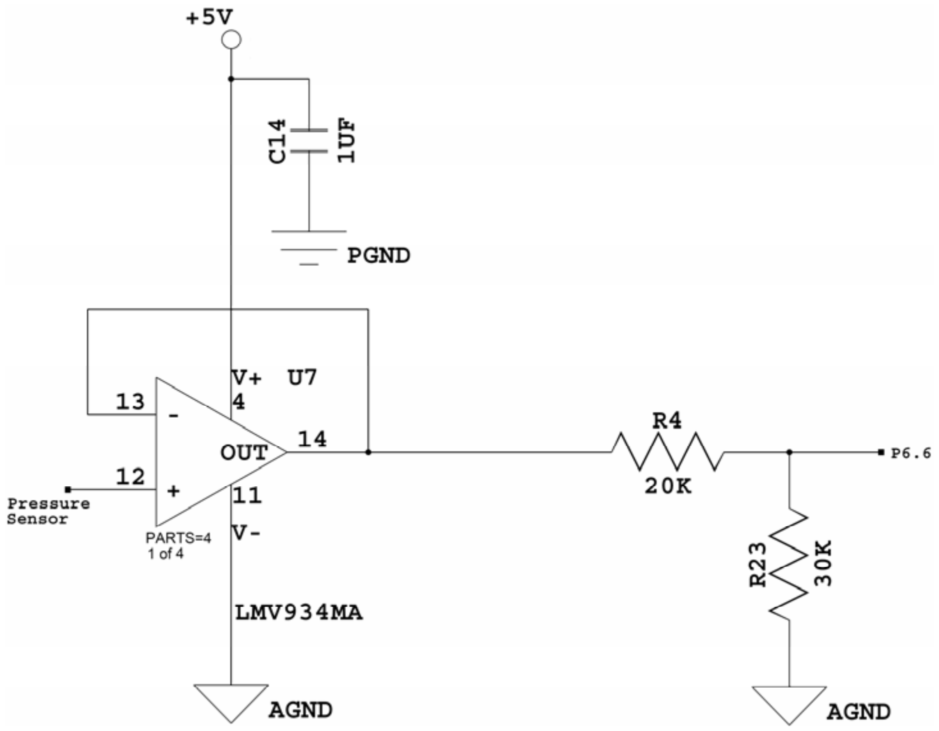


Figure 6.4: Single-Supply Op-Amp Attenuator Circuit

6.4. MemSense© MAG10-1200S050 Tri-Axial Analog Inertial Sensor

The most distinctive and frankly astonishing sensor on the MCETS sensor board is the MemSense© MAG10-1200S050 tri-axial analog inertial sensor. Claimed by MemSense© to be the world's smallest analog inertial measurement unit, the MAG10 incorporates a tri-axial accelerometer, gyroscope, magnetometer, and internal temperature sensor in a 0.70 x 0.70 x 0.40-inch surface mount forty-four pin package that weighs a mere five grams. In such a small package, the sensor can measure acceleration, angular rate (rotation), magnetic field strength, and temperature (for compensation techniques) about three orthogonal axes. Furthermore, the device only draws a maximum of 35 mA at a supply voltage of 5.0 volts, for a total power consumption of only 175 mW. In comparison to the tri-axial analog accelerometer on the WITS, the MAG10 consumes slightly less power (175 mW compared to 180 mW (at the lowest supply voltage of 6.0 volts)), however the MAG10 incorporates tri-axial gyroscopes, magnetometers, and internal temperature sensors.

Because of the 5.0-volt supply, all four inertial sensors – the accelerometer, gyroscope, magnetometer, and internal temperature sensor – have output voltages centered at 2.50 volts. This output voltage corresponds to an acceleration of 0.00 g, an angular rate of 0.00°/s, a magnetic field strength of 0.00 gauss, and an internal temperature of 25°C, respectively. Stemming off of this center output voltage, the accelerometer has a sensitivity of 200 mV/g, the gyroscope 1.25 mV/(°/s), the magnetometer 1.00 V/gauss, and the internal temperature sensor 8.4 mV/°C. Accordingly, the relationship of each output voltage to acceleration, angular rate, magnetic field strength, and internal temperature is

$$a_{x,y,z} = \frac{v_{out} - 2.50}{0.200} = 5v_{out} - 12.5 \quad [g], \tag{6.4}$$

$$\omega_{x,y,z} = \frac{v_{out} - 2.50}{0.00125} = 800v_{out} - 2000 \quad [°/s], \tag{6.5}$$

$$B_{x,y,z} = v_{out} - 2.50 \quad [gauss], \tag{6.6}$$

and

$$T_{x,y,z} = \frac{v_{out} - 2.29}{0.0084} \quad [°C], \tag{6.7}$$

respectively.

Due to the wide range of the inertial sensors on the MAG10 – ±10 g for acceleration, ±1200°/s for angular rate, and ±1.90 gauss for magnetic field strength – the output voltage swing of each sensor exceeds the maximum microcontroller ADC input voltage of 3.3 volts. As illustrated in Figure 6.5: Acceleration versus Output Voltage, each axis on the accelerometer can produce an output voltage between 0.5 and 4.5 volts. Similarly, the gyroscope can produce an output between 1.0 and 4.0 volts (Figure 6.6: Angular Rate versus Output Voltage), the magnetometer between 0.6 and 4.4 volts (Figure 6.7: Magnetic Field Strength versus Output Voltage), and the internal temperature sensor between 1.9 and 3.1 volts (Figure 6.8: Internal Temperature versus Output Voltage). Therefore, an op-amp buffer and a voltage divider identical to the one depicted in Figure 6.4 is needed to interface each axis output with the ADC on the MSP430 microcontroller. (Though the tri-axial internal temperature sensor is within the voltage requirements of the ADC (0.0 to 3.3 volts), in order to isolate the potential 5.0-volt source and prevent any possible damage to the microcontroller, an op-amp buffer and voltage divider are also used to step down its voltage.) (For additional specifications on the MemSense© tri-axial analog inertial sensor, see Appendix A.4: MemSense© MAG10-1200S050 Tri-Axial Analog Inertial Sensor Datasheet).

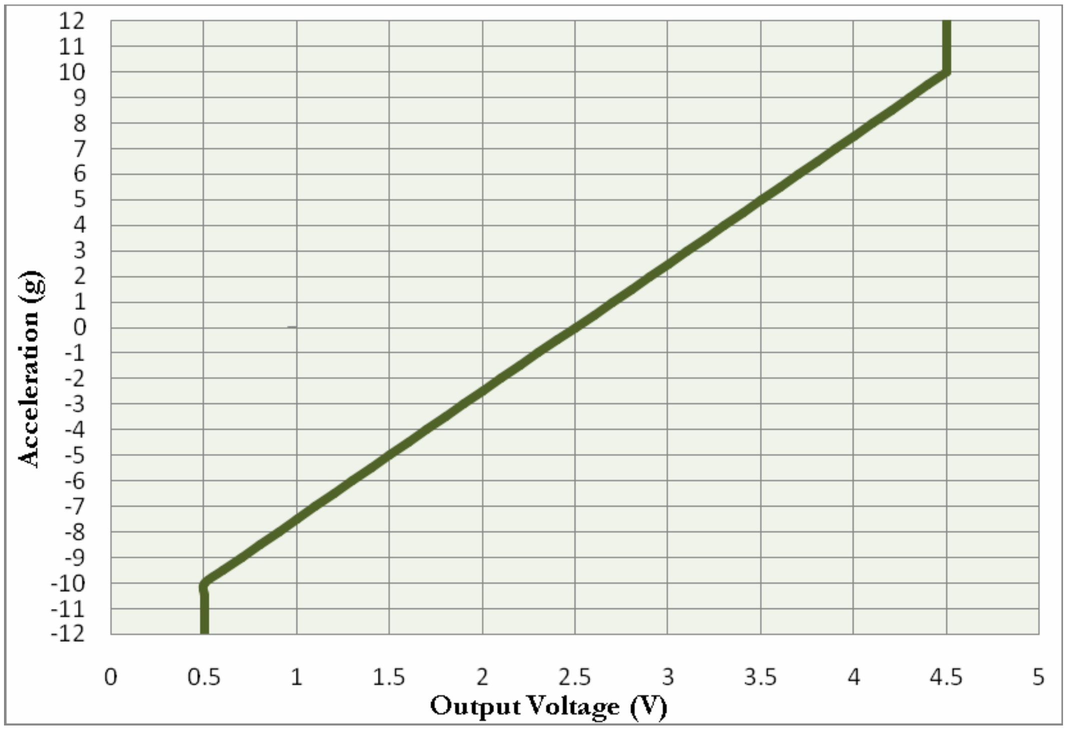


Figure 6.5: Acceleration versus Output Voltage

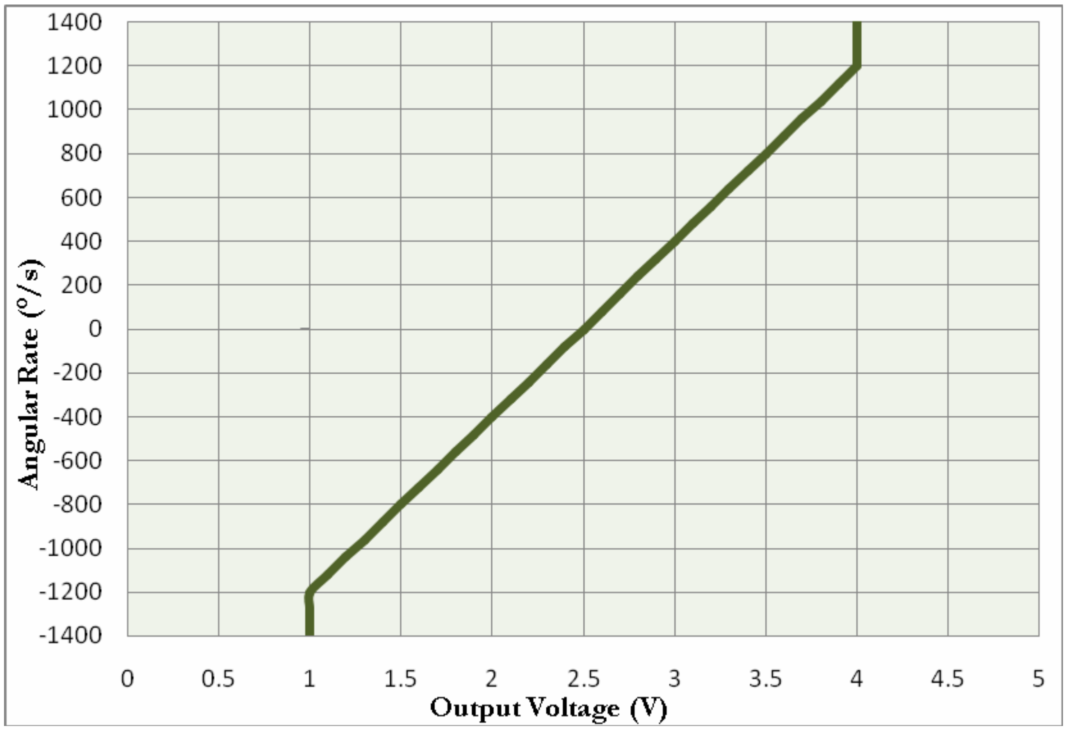


Figure 6.6: Angular Rate versus Output Voltage

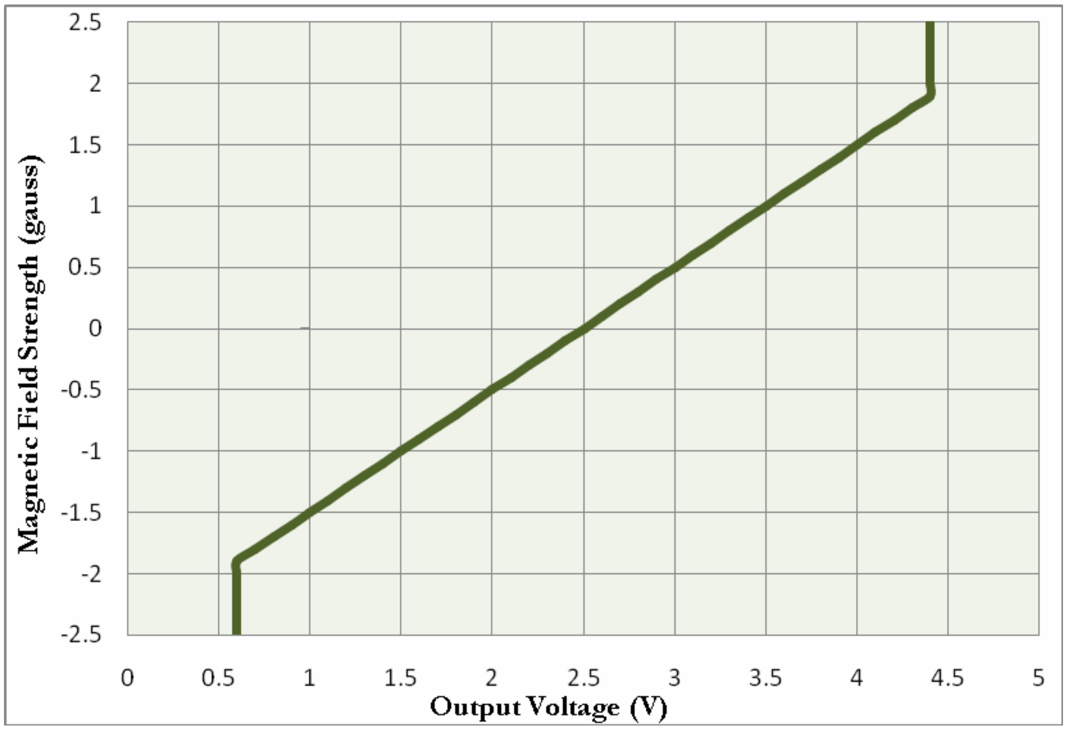


Figure 6.7: Magnetic Field Strength versus Output Voltage

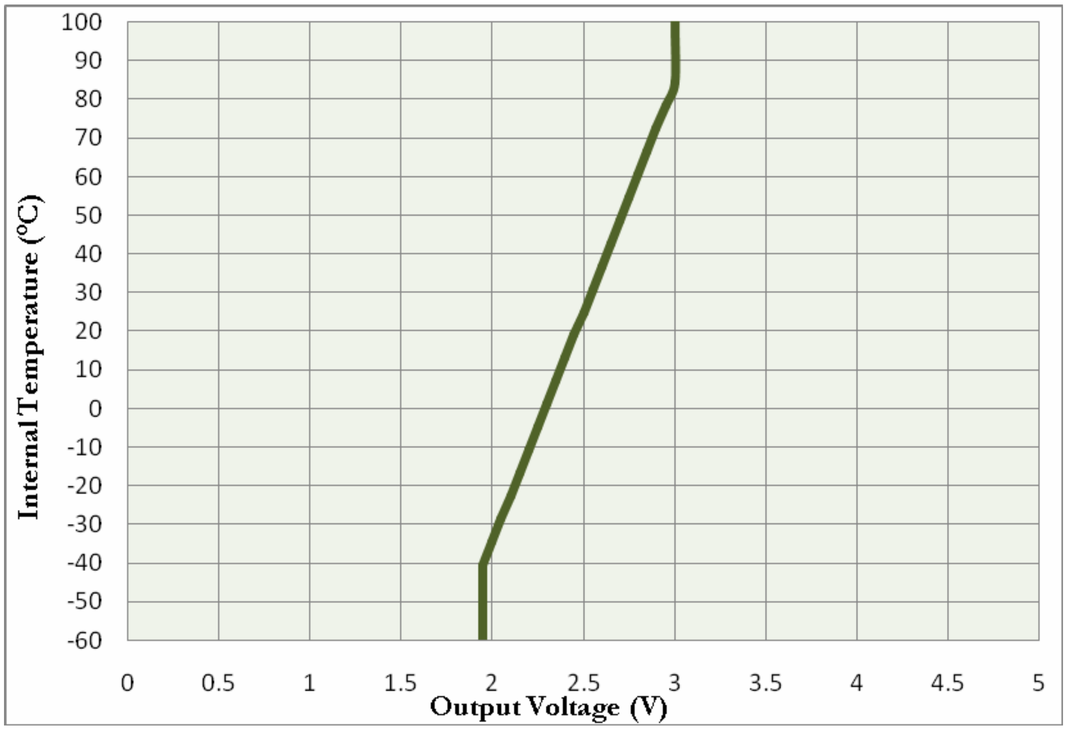


Figure 6.8: Internal Temperature versus Output Voltage

Since the MAG10 incorporates four tri-axial analog sensors, there are a total of twelve signal outputs from the inertial sensor that need to be interfaced with the ADC on the MSP430-F1611 microcontroller. However, the microcontroller only has one eight-channel ADC, in which two channels are already being used for the temperature and pressure sensors. As a result, each MCETS client sensor board incorporates three four-to-one low power (5.0 µW) and ultra-fast switching (less than 20.0 nanoseconds (ns)) multiplexers. By utilizing one multiplexer per axis (the Analog Devices© ADG704 Multiplexer), connecting the three control lines A1, A0, and EN together, and controlling each device with the general purpose I/O pins on the microcontroller, via simply applying the control signals in Table 6.2: 4:1 Analog Multiplexer Truth Table, the *x*-, *y*-, and *z*-axis of each sensor can be selected and passed on to the ADC. Furthermore, this multiplexer setup reduces the number of single-supply op-amp attenuators from twelve down to only three. (For the full details of how the MAG10 is connected and interfaced with the microcontroller development board, see Appendix B.4: Tri-Axial Analog Inertial Sensor Pin Connections, Appendix B.5: Analog Multiplexer Pin Connections, Appendix B.6: Operational Amplifier/Voltage Attenuator Pin Connections, and Appendix C.4: Tri-Axial Analog Inertial Sensor Circuit Schematic.)

A1	A0	EN	Selected Inertial Sensor
–	–	0	None
0	0	1	Tri-Axial Accelerometer (MUX 1: *x*-Axis Acceleration; MUX 2: *y*-Axis Acceleration; MUX 3: *z*-Axis Acceleration)
0	1	1	Tri-Axial Gyroscope (MUX 1: *x*-Axis Angular Rate; MUX 2: *y*-Axis Angular Rate; MUX 3: *z*-Axis Angular Rate)
1	0	1	Tri-Axial Magnetometer (MUX 1: *x*-Axis Magnetic Field; MUX 2: *y*-Axis Magnetic Field; MUX 3: *z*-Axis Magnetic Field)
1	1	1	Tri-Axial Internal Temperature Sensor (MUX 1: *x*-Axis Temperature; MUX 2: *y*-Axis Temperature; MUX 3: *z*-Axis Temperature)

Table 6.2: 4:1 Analog Multiplexer Truth Table

6.5. Javad© JNS100 GPS OEM Receiver

The GPS OEM receiver selected for the MCETS is the same receiver used in the WITS – the Javad© JNS100 GPS OEM Receiver – mainly because of its significantly higher accuracy performance over most commercial GPS units, as well as the fact that The Laboratory has both access to and familiarity with the receiver. As previously stated, the JNS100 is capable of tracking up to fifty different GPS and GLONASS (the Global Navigation Satellite System (the Russian Federation's counterpart GPS system)) satellites while producing a raw data output rate up to one-hundred Hertz. Additionally, the receiver has an onboard voltage regulator that can accept and measure unregulated voltages between 6.5 and 40.0 volts. The receiver also consumes a maximum of 2.3 watts of power when using a 40.0-volt source and a powered antenna. However, the manufacturer lists the typical power consumption around 1.1 watts when using a lower supply voltage and an unpowered antenna. In fact, initial testing of the GPS receiver reveals a measured power consumption averaging between 0.9 and 1.0 watts when using an 11.1-volt voltage source (the typical battery voltage of a lithium ion battery).

In terms of interfacing with the sensor board and the MSP43-P1611 microcontroller development board, the GPS receiver has a thirty-pin header that is matched to an identical thirty-pin header on the sensor board (Appendix B.1: Miscellaneous Header and Connector Pin Connections, and Appendix C.1: Miscellaneous Header and Connector Circuit Schematics). The sensor board supplies the receiver with an unregulated voltage directly from the battery on the power supply board, as well as provides the power and digital grounds necessary for proper operation. The sensor board header also acts as an interconnection between the female RS-232 port on the microcontroller and the RS-232 serial port on the receiver. As a result, the microcontroller is able to communicate with the GPS OEM receiver via RS-232, providing position, velocity, and timing data to each MCETS client.

6.6. Quatech© WLNB-AN-DP102 Embedded Wireless Module

The embedded wireless module selected for the communication link between the MCETS clients and the server is the Quatech© WLNB-AN-DP102 Embedded Wireless Module. This 1.60 x 1.17 x 0.46-inch module incorporates all five layers of the Internet Protocol Suite, including an application processor, the TCP transport layer, the IPv4 network layer, the IEEE 802.11b data link layer, and an over-the-air physical Wi-Fi layer. Most significantly, the application processor handles the transfer of data between the microcontroller (the master device) and the embedded wireless module (a slave device) using a high-speed four-wire SPI bus. As a result, the WLNB-AN-DP102 embedded wireless module provides all of the necessities for wireless TCP/IP communications, allowing the microcontroller to focus on efficient data acquisition, not the specific details of the Internet Protocol Suite.

Running on a supply voltage of 3.3 volts, the electrical characteristics of the WLNB-AN-DP102 wireless module are very favorable for microcontroller-based systems like the MCETS. Typically, the module draws around 420 and 350 mA of current while transmitting and receiving data, respectively, resulting in a typical power dissipation of 1.386 and 1.155 W. However, a major concern with the module is that it has an initial inrush current in excess of 1900 mA when the device first turns on, a potential problem for current-limited voltage supplies. Consequently, since the microcontroller and temperature sensor also operate on 3.3-volt sources, each MCETS employs two isolated 3.3-volt supply rails, one for the high-current (HC) embedded wireless module and another for the low-current (LC) microcontroller and temperature sensor devices.

As illustrated in Figure 6.9: Quatech© WLNB-AN-DP102 Embedded Wireless Module Block Diagram, the module has four status indicator signals for external use, including power on self test (POST), connection status (CONN), radio-frequency link (LINK), and radio frequency activity (RF Status). Specifically, the POST indicator denotes whether the module successfully loaded, the CONN indicator whether the module obtained a network-registered IP address, and the LINK indicator whether the module is connected to an access point or Ad hoc peer. Furthermore, the RF Status indicator blinks when the module is on and scanning for an access point, and is solid when the module is on and associated to an access point. Since all these indicators provide critical information about the embedded wireless module and its network connection status, the MCETS utilizes all of these indicator signals, where the CONN, LINK, and RF Status signals drive external LEDs on the sensor board, and POST, CONN, and LINK are connected to the microcontroller's general-purpose I/O pins.

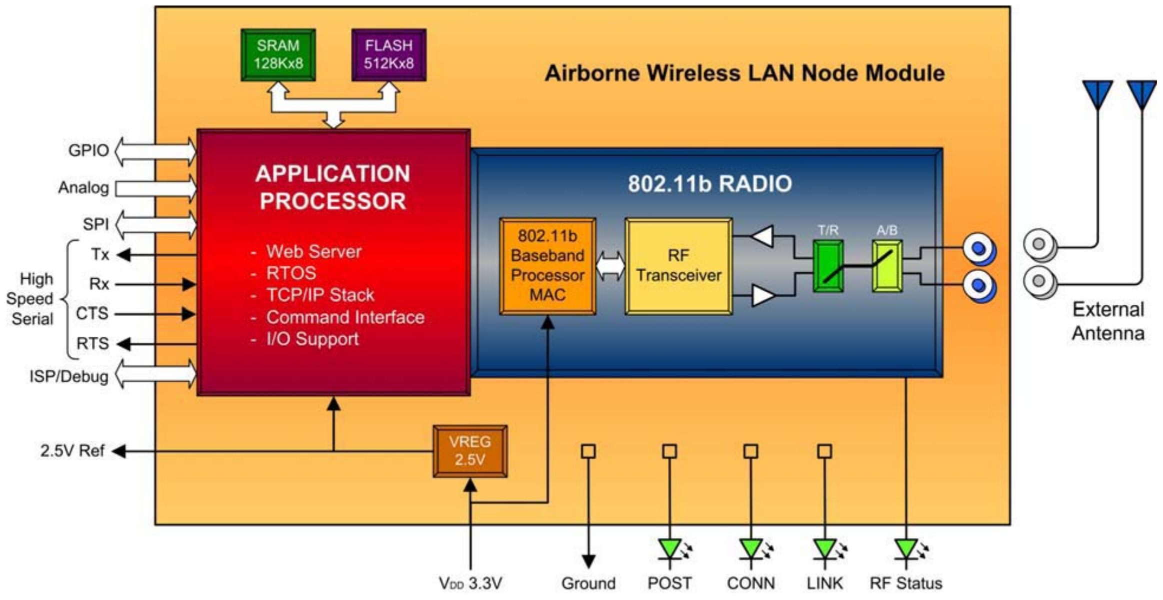


Figure 6.9: Quatech© WLNB-AN-DP102 Embedded Wireless Module Block Diagram

(Source: *Airborne Embedded Wireless Device Server*)[2]

Finally, the Quatech© WLNB-AN-DP102 Embedded Wireless Module incorporates a built-in web server for easy monitoring and controlling of the module. Within this web server, project-specific variables, including a primary and secondary static IP address, the service set identifier (SSID), and whether to operate in infrastructure or Ad hoc mode, are configured and stored in Flash memory. Properly setting these variables permits full-duplex communications between the MCETS clients and the MATLAB-based server over the configured wireless network. Additionally, with the attachment of two Omni-directional

U.FL antennas, each with a gain of five isotropic decibels (dBi), multi-path diverse signals facilitate an extended range of up to a absolute maximum line-of-sight distance of 590 meters (approximately 1,935 feet).

6.7. Power Supply Board

Having all of the components selected for the sensor board and the system completely designed, the next step is designing a power supply board that can provide adequate power to the sensor, microcontroller development, and GPS receiver boards. Each MCETS client requires four separate voltages lines, including a 3.3-volt low-current (LC) supply, a 3.3-volt high-current (HC) supply, a 5.0-volt supply, and an unregulated supply around 11.1 volts for the GPS OEM receiver. The 3.3-volt LC supply is used to power the microcontroller development board and the temperature sensor. The 3.3-volt HC supply on the other hand is used exclusively for the embedded wireless module, which has a peak inrush current in excess of 1.9 amperes when the device initially turns on. Furthermore, the 5.0-volt supply is used to power the pressure sensor, the tri-axial inertial sensor, the three analog multiplexers, the two four-bit bus switches, and the four single-supply op-amp attenuator circuits.

As illustrated in Table 6.3: Estimated Maximum MCETS Client Power Requirements, the estimated current needed to supply the system is approximately 593.069 mA, or in terms of power, 2.7036 watts. This calculated current requirement is the aggregate of the maximum supply current for each component on the sensor board (as listed on each device's datasheet) and the measured current needed to supply the GPS OEM receiver and the microcontroller development board. Specifically, the 3.3-volt LC source must supply 11.14 mA (36.762 mW), the 3.3-volt HC supply 450 mA (1.485 watts), the 5.0-volt supply

46.329 mA (152.9 mW), and the 11.1-volt source 85.6 mA (950 mW). Recall from Section 6.6: Quatech© WLNB-AN-DP102 Embedded Wireless Module, the wireless unit has a peak inrush current of approximately 1.9 amperes, and thus the low-current and high-current 3.3-volt sources are isolated from one another on the sensor board. As a result, each MCETS client requires three regulated voltage supplies (3.3 volts LC, 3.3 volts HC, and 5.0 volts) and a raw battery voltage around 11.1 volts.

Component	Supply Voltage	Current Draw	Power Consumption
MSP430-P1611 (1)	3.3 V (LC)	11 mA†	36.3 mW†
Temperature Sensor (1)	3.3 V (LC)	140 μA	462 μW
Pressure Sensor (1)	5.0 V	10 mA	50 mW
Pressure Sensor Op-Amp (1)	5.0 V	230 μA	1.150 mW
Pressure Sensor Attenuator (1)	5.0 V	100 μA	500 μW
Tri-Axial Inertial Sensor (1)	5.0 V	35 mA	175 mW
Tri-Axial Inertial Sensor Multiplexers (3)	5.0 V	3.0 μA	15.0 μW
Tri-Axial Inertial Sensor Op-Amps (3)	5.0 V	690 μA	3.450 mW
Tri-Axial Inertial Sensor Attenuator (3)	5.0 V	300 μA	1.5 mW
GPS OEM Receiver (1)	11.1 V	85.6 mA†	950 mW†
802.11 Embedded Wireless Module (1)	3.3 V (HC)	450 mA	1.485 W
Bus Switches (2)	5.0 V (HC)	6.0 μA	30 μW
† Average measured value			
		Total 3.3 V (LC) Current Draw	11.14 mA
		Total 3.3 V (HC) Current Draw	450 mA
		Total 5.0 V Current Draw	46.329 mA
		Total 11.1 V Current Draw	85.6 mA
		Total Power Consumption	2.7036 W

Table 6.3: Estimated Maximum MCETS Client Power Requirements

The battery selected for each MCETS client, the direct supply for the GPS OEM receiver, is a standard 11.1-volt lithium ion battery. This rechargeable battery has a peak voltage of 12.6 volts, though its average output voltage is approximately 11.1 volts.

Additionally, the battery has a capacity of 4.4 ampere-hours (meaning it can supply an ampere of current for approximately 4.4 hours), and has a maximum discharge current of 5.0 amperes. Since the estimated current draw is 593.069 mA, each MCETS client can run on battery for a maximum of 7.4 hours, although the actual operating time should be less with the addition of filter capacitors and as the battery ages. Finally, the battery physically measures 69 mm long by 54 mm wide by 36 mm thick (2.72 x 2.13 x 1.417 inches), and weighs about 340 grams. As a result, the 11.1-volt lithium ion battery conforms to the small form factor of the MCETS, abiding by the system's size and weight requirements.

Since linear voltage regulators are extremely inefficient, downwards of fifty to sixty percent, the choice method for stepping the 11.1-volt battery source down is through a switching voltage regulator. The switching regulators selected for the power supply board are the Bel$^{©}$ V7AH-03H series DC/DC Converters (Appendix A.7: Bel© x7AH-03H Series DC/DC Converters). The series consists of 1.2-, 1.5-, 1.8-, 2.5-, 3.3-, and-5.0 volt switching regulators that are capable of supplying 3.0 amperes of current. Consequently, the need for a separate 3.3-volt HC and LC supply is no longer necessary since the regulators can supply enough current to all 3.3-volt devices during the embedded wireless modules 1.9-ampere peak inrush current startup. (Due to project time constraints and the way the MCETS design unfolded, the sensor board utilizes both a 3.3-volt LC and a 3.3-volt HC supply. However, as illustrated in Appendix C.6: Power Supply Board Circuit Schematic, the two supply rails are shorted together on the power supply board).

In terms of power efficiency, for an 11.1-volt input source the 5.0-volt switching regulator (Bel$^{©}$ V7AH-03H500) is approximately eighty-three to ninety-two percent efficient, depending on the output current of the device. From Table 6.3, the estimated maximum current drawn from the 5.0-volt supply is 46.329 mA, which from the manufacturer's efficiency data in Appendix C.6 corresponds to an efficiency of approximately eighty-three percent (the switching regulator becomes more efficient at higher output currents). Similarly, the efficiency of the 3.3-volt switching regulator (Bel© V7AH-03H330) for an 11.1-volt input source ranges from about eighty to ninety percent, though at the 3.3-volt estimated supply current of 461.14 mA, it is only eighty-two or eighty-three percent efficient. Nevertheless, regardless of output currents, the Bel© V7AH-03H Series Non-Isolated DC/DC Converters yield significantly higher power conversion efficiencies over traditional linear voltage regulators.

6.8. Sensor Printed Circuit Board Layout

As illustrated in Appendix A.5: Javad© JNS100 GPS OEM Receiver Datasheet, the GPS OEM receiver PCB board measures 87.63 mm (3.45 inches) long by 57.13 mm (2.25 inches) wide. Being such a standard size (and slightly larger than the microcontroller development board), this form factor is used for the custom-made sensor and power supply boards as to maintain a common size for all four layers of the client board layout. The sensor board is designed to be the backbone of each MCETS client, providing the data and power connections between the microcontroller development board, GPS OEM receiver, and power supply board. Additionally, the sensor board contains the temperature, pressure, and tri-axial analog inertial sensors, all miscellaneous electrical components, and the embedded wireless module.

The software used to design the sensor board, as well as the power supply board, is the Mentor Graphics© suite. Within the suite, circuit schematics generated in Mentor Graphics© DxDesigner – Appendix C: Circuit Schematics – are transferred over to Mentor Graphics© Expedition, where the components can be properly placed and connected with electrical traces (Appendix D: Printed Circuit Board Layouts). Some of the prominent features of the Expedition software include auto-routing, a crosstalk simulator, and a propagation delay simulator. The auto-routing feature, which automatically routes the connections between components, proves to be useful in most cases, though care must taken to ensure proper trace routing. Additionally, the crosstalk simulator produces an estimated maximum crosstalk potential based on adjacent parallel traces and trace widths, and the propagation delay simulator produces an estimate on critical data lines that are time sensitive (e.g., the tri-axial inertial analog sensor traces).

Initially, the sensor board consisted of four individual layers, including a power layer for the 3.3-volt LC, 3.3-volt HC, 5.0-volt, and 11.1-volt supply rails, a ground layer for the analog, digital, and power grounds, and two trace layers for the actual interconnections between components. However, due the sheer size and number of connections on the sensor board, four individual layers are insufficient for the MCETS sensor board in terms of physically being able to route each of the traces. Consequently, as illustrated in Table 6.4: Sensor Board PCB Layers, the final sensor board design consists of eight individual layers, including a power layer, three ground layers, and four trace layers. This ultimately alleviates the "real-estate problem" of not having enough physical board space to route each of the traces; it even permits biasing signal traces in certain direction, where the first signal layer is biased to have its traces placed vertically, the second signal layer biased horizontally, and so on, helping to reduce the amount of potential crosstalk between signals.

Layer	Type	Description	Appendix Figure
1	Trace	Sensor Pads and Signal Layer	Figure D.4: First Sensor PCB Layer
2	Ground	Analog Grounding Plane	Figure D.5: Second Sensor PCB Layer (Analog Ground Plane)
3	Trace	Signal Layer	Figure D.6: Third Sensor PCB Layer
4	Ground	Digital Grounding Plane	Figure D.7: Fourth Sensor PCB Layer (Digital Ground Plane)
5	Trace	Signal Layer	Figure D.8: Fifth Sensor PCB Layer

6	Power	Power Supply Plane	Figure D.9: Sixth Sensor PCB Layer (Power Supply Plane)
Layer	**Type**	**Description**	**Appendix Figure**
7	Ground	Power Grounding Plane	Figure D.10: Seventh Sensor PCB Layer (Power Ground Plane)
8	Trace	Discrete Pads and Signal Layer	Figure D.11: Eighth Sensor PCB Layer

Table 6.4: Sensor Board PCB Layers

6.9. Power Supply Printed Circuit Board Layout

The power supply PCB is designed to provide all the necessary voltage sources for the MCETS client, specifically the 3.3-volt, 5.0-volt, and 11.1-volt supplies, as well as provide the sole connection of the three grounding planes (analog, digital, and power). Additionally, the board includes jumper configurations that allow for extended system control of the switching voltage regulators and the way the grounding planes are connected. Since the power supply PCB is far less complex than the sensor PCB (sixteen components compared to sixty-nine), only two individual signal trace layers are needed. As illustrated in Appendix D.2: Power Supply Printed Circuit Board Layout, these traces are isolated on the left-hand side of the board, allowing space for a lithium ion battery on the right-hand side. Given that the power supply board uses a male DC barrel jack, the MCETS clients are capable of utilizing any 8.0-volt to 32.0-volt battery with a female DC plug, though an 11.1-volt lithium ion battery is recommended.

With the Bel© V7AH-03H series DC/DC Converters having active-low control pins, the power supply board has two control lines that facilitate turning the 3.3-volt and 5.0-volt supplies off using the microcontroller's general-purpose I/O pins. To enable or disable the MCETS switching regulators, the "3.3V CTRL" and "5.0V CTRL" (Figure D.14: Top Power Supply Board Silk Screen) pins should be shorted with separate two-pin jumpers. Otherwise, to leave the switching regulators permanently on, the "NO CTRL" pins must be shorted with separate two-pin jumpers. Even though the two switching regulates are independently configurable, caution must be used when controlling the 3.3-volt switching regulator via the microcontroller since it traditionally uses this switching regulator for power. Furthermore, the control and no control pins should not both be shorted at the same time, which electrically results in shorting the microcontroller's output control pins to ground

---

Notes

1. "MSP430 Ultra-Low-Power Microcontrollers", Texas Instruments, n.d., <http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?sectionId=95&tabId=1200&familyId=342&family=mcu> (12 September 2007).

2. "Airborne Embedded Wireless Device Server", Quatech, August 2006, <http://www.dpactech.com/docs/wireless_products/AB%20wireless%20device%20server%20module.pdf> (6 October 2007).

7. Cost Analysis

With the overall cost of the MCETS being one of The Laboratory's top concerns (COTS telemetry systems can cost tens of thousands of dollar per module), it is now appropriate to analyze both the one-time and reoccurring expenditures required to develop, fabricate, and assemble a single MCETS client. As illustrated in the following tables, the most efficient method of analyzing the overall cost is to divide the total cost into six different categories and perform an individual cost analysis on each. By breaking it down by the sensor board, microcontroller development board, GPS OEM receiver, power supply board, system assembly, and miscellaneous nonrecurring costs, a more in-depth cost breakdown is obtained, providing insight into how individual subsystems and sensors affect the total price of the system.

Besides from the Javad© JNS100 GPS OEM receiver, the cost of the MCETS sensor board – Table 7.1: Sensor Board Cost Analysis – is the largest expenditure in the MCETS system. Although most of the components on the sensor board cost less than twenty dollars, the MemSense© MAG10 Tri-Axial Analog Inertial Sensor costs more than one-thousand dollars per unit, accounting for more than seventy-eight percent of the total sensor board price. With respect to almost all of the other components, this expense significantly raises the total price of an MCETS client. However, in comparison to other commercial tri-axial inertial sensors, the MAG10 costs about one-tenth of what similar units cost. Furthermore, the fabrication of the MCETS sensor board by Network Circuits© adds eighty-four dollars to the total sensor board cost; however, due to the nature of circuit boards, the entire lot of twenty-five boards must be purchased for approximately $2,100.00. Nonetheless, based on current electronic supplier prices as of October 2007, the total cost of all the sensor board components for a single MCETS client is $1,282.73.

QTY	Description	Manufacturer	Part Number	Price	Total Price
1	Sensor Printed Circuit Board	Network Circuits©	–	$84.00	$84.00
1	Analog-Output Temperature Sensor	Maxim©	DS600U	$2.57	$2.57
1	Analog-Output Pressure Sensor	Freescale Semiconductor©	MPXA4250A6U-ND	$14.36	$14.36
1	470 pF SMT Capacitor	AVX Corporation©	08053D105KAT2A	$0.01	$0.01
1	Tri-Axial Analog Inertial Sensor	MemSense©	MAG10-1200S050	$1,004.40	$1,004.40
1	N-Channel MOSFET Transistor	ON Semiconductor©	MMBF0201NLT1	$0.35	$0.35
3	4:1 CMOS Analog Multiplexer	Analog Devices©	ADG704BRMZ-ND	$2.33	$6.99
1	4-Channel Operational Amplifier	National Semiconductor©	LMV934MA-ND	$1.49	$1.49
4	0.1% Tolerant 20 kΩ SMT Resistor	Panasonic©	ERA6YEB203V	$0.56	$2.24
6	0.1% Tolerant 30 kΩ SMT Resistor	Panasonic©	ERA6YEB303V	$0.56	$3.36
3	2.2 µF SMT Capacitor	Panasonic©	ECJ-2FB1E225K	$0.15	$0.45
1	Airborne Embedded Wireless Module	Quatech©	600-WLNG-AN-DP102	$100.62	$100.62
2	5 dBi Rubber Duck U.FL Antenna	Quatech©	ACH2-AT-DP004-G	$9.10	$18.20
2	4-Bit Tri-State Bus Switch	Fairchild©	FST3126QSC	$0.52	$1.04
1	36-Pin Female Connector	Hirose Electronics©	DF12(4.0)-36DP-0.5V(86)	$1.58	$1.58
3	SMT Clear Red LED	CML Technologies©	CMD28-21SRC/TR8/T1	$0.32	$0.96
3	680 Ω SMT Resistor	Panasonic©	ERJ6GEYJ681V	$0.07	$0.21
4	1 MΩ SMT Resistor	Panasonic©	ERJ-8ENF1004V	$0.12	$0.48
11	1 MΩ SMT Resistor	Susumu©	RR1220P-105-D	$0.08	$0.88
3	10 µF SMT Capacitor	Panasonic©	ECJ-3YB1E106M	$0.54	$1.62
14	1.0 µF SMT Capacitor	Panasonic©	08053D105KAT2A	$1.09	$15.26
2	Push Button DPST NO Switch	Alps©	SKHMPSE010	$0.650	$1.30
1	2-Pin Male Header	Tyco Electronics©	87220-2	$1.17	$1.17
1	10-Pin Right-Angle Male Header	Molex©	87833-1020	$1.38	$1.38
1	30-Pin Male Header	Molex©	90131-0135	$3.09	$3.09
1	60-Pin Male Header	Tyco Electronics©	3-87227-0	$15.96	$15.96
1	2-Pin Female Connector	FCI©	65039-035LF	$0.94	$0.94
2	Mini PV Contacts	FCI©	47750-000LF	$0.37	$0.37
1	10-Pin Female Connector	Molex©	87568-1073	$2.61	$2.61
1	30-Pin Female Connector	Assmann Electronics©	AWP30-8240-T-R	$1.49	$1.49
1	60-Pin Female Connector	Assmann Electronics©	AWP60-8240-T-R	$2.98	$2.98
					$1,282.73

Table 7.1: Sensor Board Cost Analysis

Unlike the sensor board, the microcontroller development board and GPS OEM receiver are prebuilt devices for retail sale and require few additional components. Contrasting that of the sensor board, the total cost of the MCETS microcontroller development board (Table 7.2: Microcontroller Development Board Cost Analysis) is only $49.05. On the other hand, the total cost of the GPS OEM receiver is significantly higher than both the sensor and microcontroller development boards, costing $9,902.98. As listed in Table 7.3: GPS OEM Receiver Cost Analysis, the Javad© JNS100 GPS OEM Receiver costs $9,900.00, an aggregate of costs for individual features of the receiver. The base cost of the unit is $4,500.00, however features such as a raw data rate of 100 Hz and differential GPS increase the base cost by an additional $5,300.00. Accordingly, the total expenditure of the GPS OEM receiver could be reduced to as little as $4,502.98, though the actual implementation of the MCETS utilizes the $9,900.00 GPS OEM receiver.

QTY	Description	Manufacturer	Part Number	Price	Total Price
1	MSP430 Development Board	Olimex©	MSP430-P1611	$44.95	$44.95
2	36 pF SMT Capacitor	AVX©	06035A360JAT2A	$0.39	$0.78
1	6.00 MHz Crystal Oscillator	ABRASION©	ABL-6.000MHZ-B2	$0.34	$0.34
1	60-Pin Female Connector	Assmann Electronics©	AWP60-8240-T-R	$2.98	$2.98
					$49.05

Table 7.2: Microcontroller Development Board Cost Analysis

QTY	Description	Manufacturer	Part Number	Price	Total Price
1	GPS OEM Receiver	Javad©	JNS100	$9,900.00	$9,900.00
1	30 Pin Female Connector	Assmann Electronics©	AWP30-8240-T-R	$1.49	$1.49
1	Female DB9 Connector	Black Box©	FA110	$1.49	$1.49
					$9,902.98

Table 7.3: GPS OEM Receiver Cost Analysis

Since the MCETS sensor board is custom designed specifically for the MCETS, it also consists of a multitude of components, most of which cost less than fifteen dollars (Table 7.4: Power Supply Board Cost Analysis). The only significant expenditures include the lithium ion battery ($45.55) and the printed circuit board ($30.00). As with the sensor board fabrication, the entire lot of twenty-five power supply boards must be purchased, but because it is a two-layer board, the cost for the entire lot is only $750.00. Furthermore, as listed in Table 7.5: System Assembly Cost Analysis, there is an additional expenditure of assembling the sensor and power supply boards. Specifically, it takes approximately five hours at a rate of eighty dollars per hour to solder and assemble each MCETS client, resulting in a total price of $415.08 per module

QTY	Description	Manufacturer	Part Number	Price	Total Price
1	Power Supply Printed Circuit Board	Network Circuits©	–	$30.00	$30.00
1	2.5x5.5 mm Right-Angle DC Barrel Jack	Switchcraft©	RAPC712BK	$1.23	$1.23
1	2.5x5.5 mm DC Plug	Switchcraft©	760	$3.28	$3.28
1	4400 mAh 11.1V Lithium Battery	Tenergy©	LI18650-111V4400	$45.55	$45.55
1	Two Position Slide Switch	NKK©	MS13ANW03	$3.58	$3.58
6	2 Pin Male Header	Tyco Electronics©	87220-2	$1.17	$7.02
4	2 Pin Female Jumper	Sullins Electronics©	SPC02SYAN	$0.12	$0.48
1	5.0V, 3.0A Switching Regulator	Bel Fuse© Inc.	V7AH-03H500	$13.90	$13.90
1	3.3V, 3.0A Switching Regulator	Bel Fuse© Inc.	V7AH-03H330	$13.90	$13.90
1	10 Pin Male Header	Molex©	87833-1020	$1.38	$1.38
1	10 Pin Female Connector	Molex©	87568-1073	$2.61	$2.61
4	10 µF SMT Capacitor	Panasonic©	ECJ-3YB1E106M	$0.54	$2.16
2	1.0 µF SMT Capacitor	Panasonic©	08053D105KAT2A	$1.09	$2.18
					$127.27

Table 7.4: Power Supply Board Cost Analysis

QTY	Description	Manufacturer	Part Number	Price	Total Price
8	5 mm Male to Female Standoff	Fascomp©	728-FM2100-2545-A	$0.43	$3.44
8	20 mm Male to Female Standoff	Fascomp©	728-FM2115-2545-A	$0.54	$4.32
8	15 mm Male to Female Standoff	Fascomp©	728-FM2110-2545-A	$0.62	$4.96
4	15 mm Female to Female Standoff	Fascomp©	728-FM1262-2545-A	$0.59	$2.36
5 hr	Assembly and Testing	MIT Lincoln Laboratory	–	$80.00	$400.00
					$415.08

Table 7.5: System Assembly Cost Analysis

As listed in Table 7.6: Nonrecurring MCETS Expenditures, there are some nonrecurring one-time expenditures not included in the total cost of an MCETS client. Specifically, these components allow for programming the MSP430 microcontroller, recharging the lithium ion battery, and connecting the four boards with ribbon cables. Ultimately, these costs add an additional $114.72 to the MCETS, though they are only startup costs and do not increase with additional clients. As a result, by combing the cost of the sensor, microcontroller development, GPS OEM receiver, and power supply boards, as well as the cost to assemble each MCETS client, the total cost of an MCETS client is $11,777.11 ($9,902.98 for the GPS OEM receiver board and $1,874.13 for the remainder of the system). Therefore, a MCETS client is significantly less expensive than other retail COTS telemetry systems, even though most are physically larger, consume a lot more power, and do not provide a noteworthy increase in accuracy.

QTY	Description	Manufacturer	Part Number	Price	Total Price
1	MSP430 JTAG Programmer	MicroController Corporation©	MSP-JTAG	$19.00	$19.00
1	Universal Lithium Battery Charger	Tenergy©	TLP-2000	$45.90	$45.90
1	Battery Charger Power Cable	Tenergy©	WETM-02	$2.95	$2.95
1	Male DC Barrel Jack	Switchcraft©	RAPC712BK	$1.23	$1.23
1	10-Pin Flat Ribbon Cable (5')	Digikey©	WM11-5-ND	$3.96	$3.96
1	30-Pin Flat Shielded Ribbon Cable (5')	3M©	MB30H-5-ND	$31.92	$31.92
1	60-Pin Flat Ribbon Cable (5')	Digikey©	MC60G-5-ND	$9.76	$9.76
					$114.72

Table 7.6: Nonrecurring MCETS Expenditures

8. MCETS Client Firmware Development

Having the MCETS client hardware fully designed, the next step in the prototype of the MCETS is the development of firmware to facilitate control of the various onboard sensors, the acquisition of data, and the transmission of that data to the MCETS server. The firmware for the MSP430-F1611 microcontroller is written entirely in the assembly programming language using the Texas Instruments©-recommended IAR Embedded Workbench. By writing the MCETS client firmware in a low-level programming language like assembly, every aspect of the system can be meticulously controlled to maximize code efficiency, optimize interrupt service routines, and fully exploit the five different MSP430 low-power modes. Furthermore, the IAR Embedded Workbench permits linking and compiling these assembly instructions into their corresponding operation codes, Flashing the code onto the microcontroller, as well as debugging this firmware in real time.

Overall, the firmware for the MCETS clients – Appendix E.5: MCETS Client Firmware (Assembly Language) – can be visualized as three main procedures, including client initialization, data acquisition, and data transmission. As illustrated in the flowchart in Figure 8.1: Main Procedures of the MCETS Client Firmware, client initialization (as indicated by the dotted red box) is the very first procedure that immediately follows when the MSP430-P1611 microcontroller development board is supplied power. This process is responsible for setting the microcontroller up for proper operation, initializing the various I/O ports that control the sensors, and putting the system into a low-power standby mode until a message is received from the MCETS server. Furthermore, as indicated by the dotted purple box, the data acquisition procedure is responsible for obtaining the server-requested data from the various onboard sensors. Finally, the third procedure of the MCETS client firmware is to format the acquired data into a single packet of data, transmit this packet to the embedded wireless module for streaming data back to the MCETS server, as well as determining whether more data needs to be acquired from the sensors.

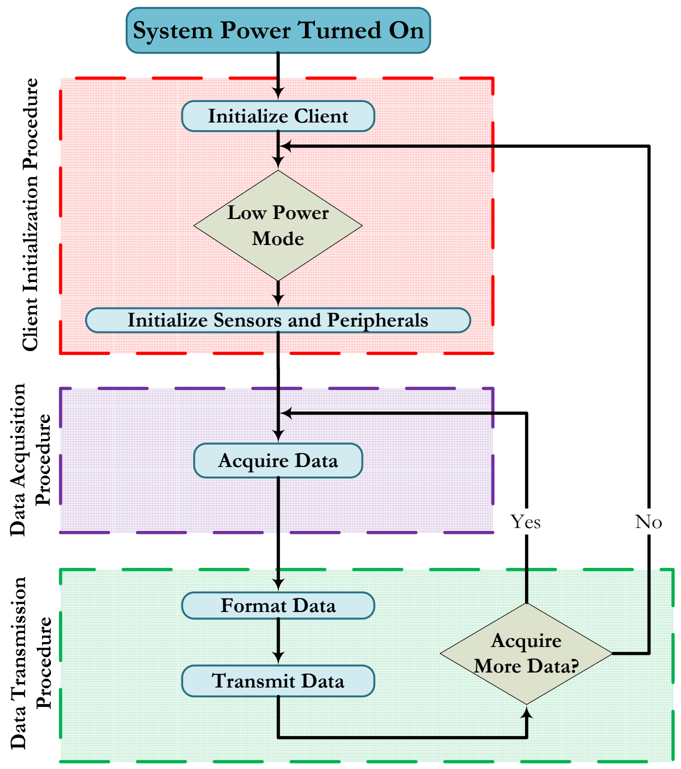

System Power Turned On
Client Initialization Procedure
Initialize Client
Low Power Mode
Initialize Sensors and Peripherals
Data Acquisition Procedure
Acquire Data
Yes
No
Data Transmission Procedure
Format Data
Transmit Data
Acquire More Data?

Figure 8.1: Main Procedures of the MCETS Client Firmware

8.1. Client Initialization Procedure

Once a sufficient voltage is supplied to the microcontroller, the firmware begins executing an initialization procedure to configure the client for both proper operation and communications with the MCETS server. This procedure (Figure 8.2: Expanded MCETS Client Initialization Procedure) is separated into two subroutines, including a routine that sets up communications with the embedded wireless module, and another that specifically turns each MCETS sensor on and prepares the system for data acquisition. These two subroutines, however, are physically separated in the fact that the microcontroller enters a low-power mode following the completion of the first routine, in which the microcontroller CPU, main clock (MCLK), submain clock (SMCLK), and digitally controlled oscillator (DCO) are turned off. These subroutines are then linked together (e.g., the CPU, MCLK, SMCLK, and DCO are turned back on to resume program execution) via an interrupt service routine that is issued by the MSP430 SPI bus peripheral (USART1).

The first subroutine of the client initialization procedure is mainly responsible for setting up communications with the embedded wireless module. However, it also stores default server settings, disables the 5.0-volt voltage supply, and turns all of the sensors off to conserve power while the client is in a low-power mode waiting for a message from the server. The main concept of the MCETS is for the server to query particular clients already in flight based upon unique module identification numbers. Ultimately, as illustrated in Appendix E.2: Packet Format for Data Transmitted from the Server to the Clients, the query consists of three sixteen-bit subpackets: a "wake-up" subpacket, a "length of data acquisition" subpacket, and a "what data" subpacket.

The first subpacket a client receives from the server – the "wake-up" subpacket – literally wakes the client microcontroller out of the low-power mode, enabling the CPU, MCLK, SMCLK, and DCO onboard the MSP430-F1611 microcontroller. This sixteen-bit message consists of the module's identification number, which is defined as the last eight bits of the client's IP address, and the server-requested data rate in Hertz. Pending that the received module identification number matches the actual identification number of the client, the microcontroller stores the received data rate as an eight-bit unsigned integer. Although the predefined maximum data rate of the MCETS is one-hundred Hertz (the maximum data rate of the Javad© JNS100 GPS OEM receiver), any received eight-bit data rate is stored in the corresponding CPU register, though it is limited to one-hundred Hertz during the data acquisition procedure.

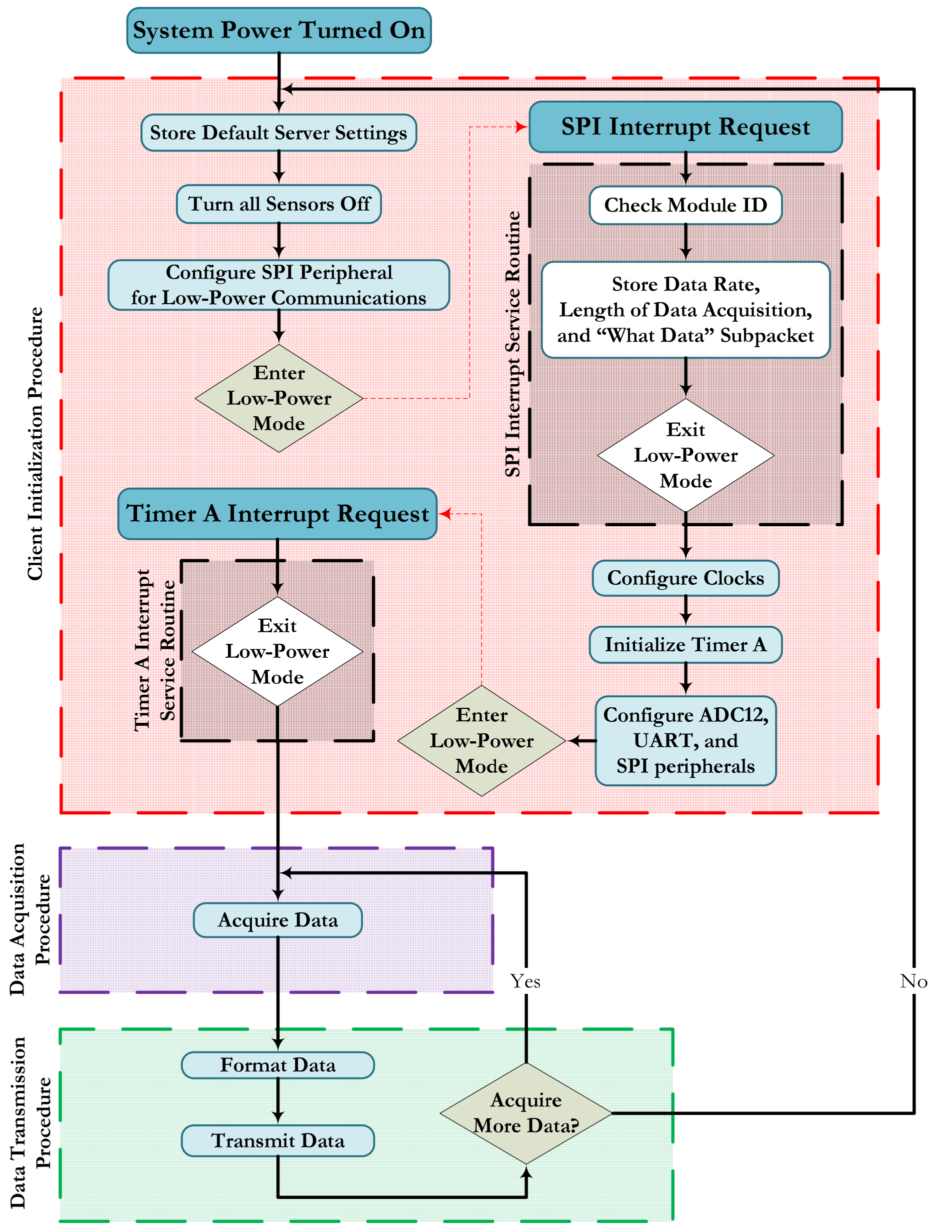

System Power Turned On
Client Initialization Procedure
Store Default Server Settings
Turn all Sensors Off
Configure SPI Peripheral for Low-Power Communications
Enter Low-Power Mode
SPI Interrupt Request
SPI Interrupt Service Routine
Check Module ID
Store Data Rate, Length of Data Acquisition, and "What Data" Subpacket
Exit Low-Power Mode
Configure Clocks
Initialize Timer A
Configure ADC12, UART, and SPI peripherals
Enter Low-Power Mode
Timer A Interrupt Request
Timer A Interrupt Service Routine
Exit Low-Power Mode
Data Acquisition Procedure
Acquire Data
Yes
No
Data Transmission Procedure
Format Data
Transmit Data
Acquire More Data?

Figure 8.2: Expanded MCETS Client Initialization Procedure

The second subpacket a client receives – the "length of data acquisition" subpacket – sets how long the MCETS client acquires data. Since the length of data acquisition is allocated sixteen bits, the client is capable of acquiring data at one-second intervals between one second and 65,535 seconds, for a maximum data acquisition period of eighteen hours, twelve minutes, and fifteen seconds. Furthermore, by setting the length of data acquisition to zero, the server can tell a client to acquire data indefinitely until a stop message is received from the server. This stop message is the same as the previously transmitted message, except for the "length of data acquisition" subpacket, which is set to the minimum data acquisition length of one second. Consequently, there is a one-second lag between when a client receives a stop message and when it actually stops acquiring data.

The third subpacket a client receives from the MCETS sever – the "what data" subpacket – allows the server to ask individual clients for particular telemetry data. Each bit in the sixteen-bit subpacket denotes whether the server wants that particular data, where a logical '1' informs the client to acquire the data and a logical '0' not to acquire the data. As illustrated in Appendix E.2, bits eleven through thirteen denote magnetic field strength, angular rate, acceleration, pressure, and temperature, and bits three through seven denote GPS receiver time, geodetic velocity, Cartesian velocity, geodetic position, and Cartesian position, respectively. Accordingly, there are six unused bits in the "what data" subpacket (bits zero, one, two, eight, nine, and ten) that are reserved for future development.

Upon receiving the six-byte packet of data from the server and the matching of identification numbers, the client is immediately taken out of the low-power mode, which is used to conserve power while the client is waiting to be queried. The received information is then moved into three sixteen-bit CPU registers for fast and easy firmware access during the data acquisition and data transmission procedures. Furthermore, since these instructions are all executed within the interrupt service routine of the SPI receiver (USART1 receiver), all values can instantly be updated while a client is acquiring and transmitting data from a previous server request.

Following the first subroutine of the client initialization procedure, as well as the reception of a valid message from the MCETS server, the second subroutine configures the system for data acquisition and transmission, as well as initializes the required sensors. Once exiting low-power mode, the three microcontroller clocks (MCLK, SMCLK, and auxiliary clock (ACLK)) are configured for high-speed operation. The MCLK, which is the clock signal used by the microcontroller CPU, is driven by the external high-frequency 6.00 MHz crystal oscillator, and the SMCLK, which is the clock signal used by most microcontroller peripheral devices, is driven by the MCLK, though it is buffered and divided by two to provide a 3.00 MHz clock signal. Since the external high-frequency clock signal is so important to proper client operation, an LED onboard the microcontroller development board is illuminated in the event of a crystal oscillator failure. Additionally, ACLK is driven by a low frequency crystal oscillator onboard the development board.

After the three clock signals are configured, a microcontroller timer peripheral (Timer A) is configured using ACLK, and initialized to provide an adequate length of time for the sensors to load properly. As defined in their respective datasheets (Appendix A: MCETS Component Data Sheets), it takes ten milliseconds for the temperature sensor to load, thirty-five milliseconds for the tri-axial analog inertial sensor, and approximately ten seconds for the GPS receiver to load (from a cold start it can take up to sixty seconds for the receiver to load, however it is assumed that at least fifty seconds elapses between when the client is supplied power and when the server queries it). As illustrated in Figure 8.2, during this ten-second startup time the required sensors are initialized, the twelve-bit ADC (ADC12), UART0, and SPI (USART1) peripherals are configured, and the microcontroller is put back into a low-power mode, where the CPU, MCLK, SMCLK, and DCO are again turned off. Once the timer finally expires and the queried MCETS client is fully initialized and configured, the microcontroller is taken back out of low-power mode to execute the data acquisition and data transmission procedures.

8.2. Data Acquisition Procedure

The data acquisition procedure of the MCETS client firmware is the portion of the assembly code that actually acquires data from the various sensors and stores it consecutively in RAM. The procedure begins by configuring and initializing a microcontroller timer peripheral (Timer B) to delineate a precise period of time in which data needs to be acquired, formatted, and transmitted. This period is equal to the inverse of the data rate, physically corresponding to how many microseconds are in one sample of telemetry data. This ultimately ensures that data is periodically transmitted to the server at the requested data rate, and that this rate is asynchronous to the actual amount of time it takes the microcontroller to acquire, format, and transmit the data.

As illustrated in Figure 8.3: Expanded MCETS Client Data Acquisition Procedure, following the initialization of the data rate timer, the procedure moves sequentially from sensor to sensor, checking with the "what data" subpacket whether the particular data has been requested by the server. Beginning with the temperature sensor, if temperature data was requested, DMA channel zero is configured to transfer a single word of data (sixteen bits) from the ADC12 channel zero register to the start address of allocated data acquisition RAM (0x1106). ADC12 channel zero is then configured to issue an interrupt request when the conversion is completed, and finally the sampling and conversion process is enabled.

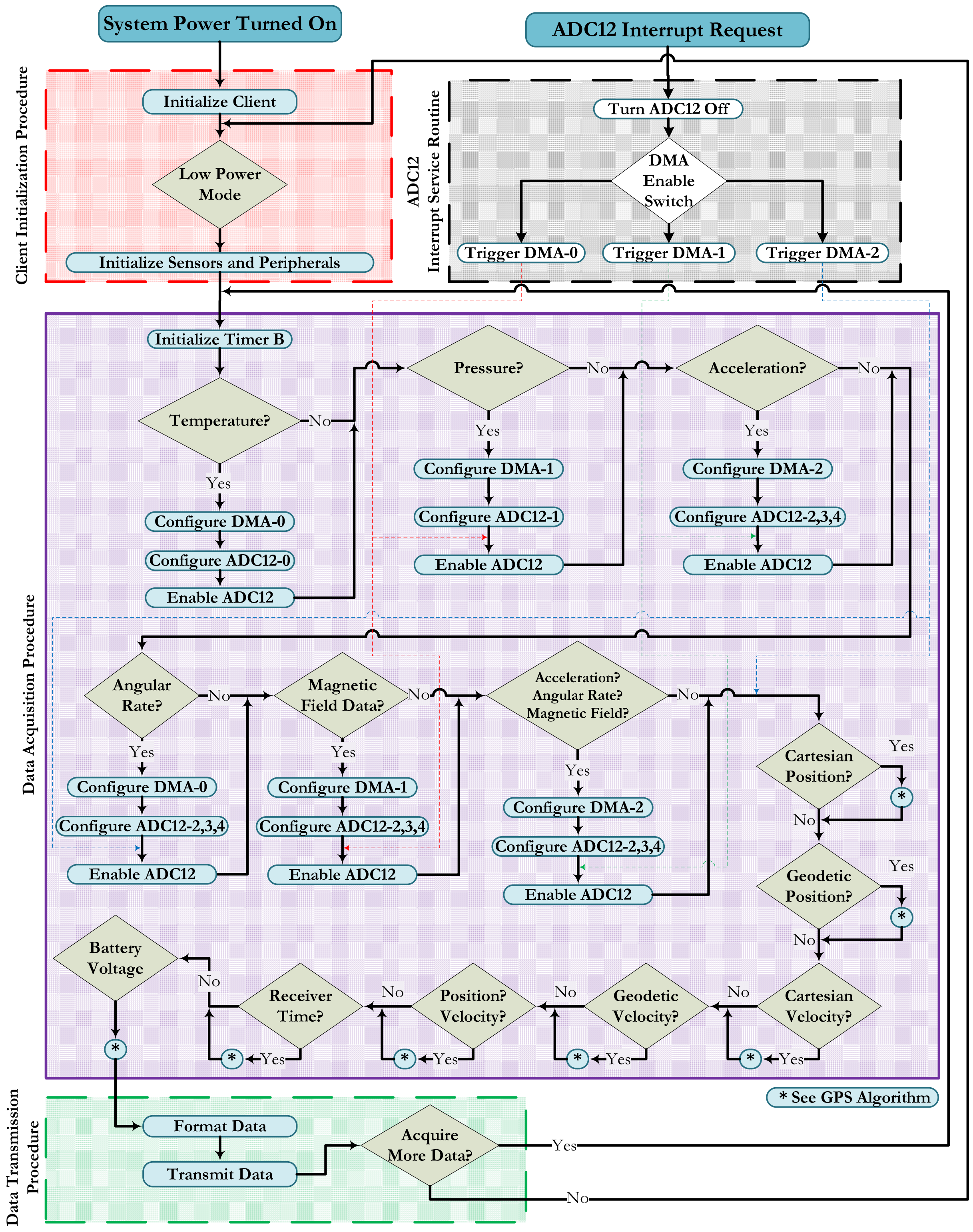


Figure 8.3: Expanded MCETS Client Data Acquisition Procedure

As the analog temperature data is converted into a binary number, the entire process is simultaneously repeated for the pressure sensor, though using DMA and ADC12 channel one. By the time the microcontroller's program counter reaches the instruction to configure ADC12 channel one (or another instruction if pressure data was not requested), ADC12 channel zero issues an interrupt request to the CPU. Within the interrupt service routine, the ADC12 is turned off to conserve power and DMA channel zero is triggered to transfer the temperature data out of the ADC12 channel zero register to RAM. Then, as DMA channel zero is moving this data, ADC12 channel one is enabled and the pressure data is sampled and converted in the exact same manner as the temperate sensor.

This parallel sequence of events is then continued for the four tri-axial analog sensors, where ADC12 channels two, three, and four are concurrently used for the *x*-, *y*-, and *z*-axes, respectively. The accelerometer then uses DMA channel two to move the forty-eight bit block of data out of the ADC12 registers to the next available location in RAM, the gyroscope DMA channel zero, the magnetometer DMA channel one, and the internal temperature sensor DMA channel two. Ultimately, by performing these three analog sensor data acquisition processes in parallel – configuring the microcontroller peripherals, sampling and converting the data, and transferring that data from the ADC12 registers to RAM – the client can utilize the CPU clock cycles wasted while the ADC12 samples and converts the data and as the data is transferred to RAM.

Once the server-requested analog sensor data is obtained and stored sequentially in RAM, GPS data is acquired from the Javad© JNS100 GPS OEM receiver using the GPS Receiver Interface Language (GRIL). GRIL is a generic receiver-independent language that allows a user (e.g., a microcontroller) to control a GPS receiver "using an appropriate set of named objects".[1] This effectively allows for manual control of GPS receivers, where ASCII-character (American Standard Code for Information Exchange) and line-feed terminated GRIL commands are transmitted to and executed by the receiver. The receiver then performs the desired operation, and if needed, returns the requested data to the user. In the case of the MCETS client firmware, the MSP430-F1611 microcontroller transmits GRILL commands, as listed in Table 8.1: Applicable GRIL Commands for the MCETS, using the onboard UART microcontroller peripheral and an RS-232 driver.

ASCII GRIL Command	**Description**
set,dev/ser/a/rate,230400<LF>	Configure serial port A's baud rate to 460800 bps
set,dev/ser/a/stops,2<LF>	Configure serial port A for 2 stop bits
set,dev/ser/a/parity,odd<LF>	Configure serial port A for odd parity
init,/dev/nvm/a<LF>	Reset and reboot the receiver
set,lpm,on<LF>	Enables the GPS processor to enter low power mode
set,sleep,on<LF>	Put the receiver into sleep mode
out,,jps/PO<LF>	Fetch Cartesian position
out,,jps/VE<LF>	Fetch Geodetic position
out,,jps/PG<LF>	Fetch Cartesian velocity
out,,jps/VG<LF>	Fetch Geodetic velocity
out,,jps/DP<LF>	Fetch dilution of precision
out,,jps/PS<LF>	Fetch position statistics
out,,jps/RD<LF>	Fetch receiver date
out,,jps/RT<LF>	Fetch receiver time
print,pwr/board<LF>	Fetch the raw battery voltage

Table 8.1: Applicable GRIL Commands for the MCETS

The first six GRILL commands listed in Table 8.1 apply to configuring the GPS receiver for communications with the microcontroller, as well as resetting and initializing it to conserve power while the client waits to be queried. By default, the serial ports on the JNS100 GPS OEM receiver use a baud rate of 115200 bps, eight data bits, no parity bit, and one stop bit. Nonetheless, in order to increase the information throughput and the reliability of communications between the microcontroller and the GPS receiver, the MCETS clients utilize a baud rate of 230400 bps, an odd parity bit, and two stop bits, parameters that are configured in the client initialization procedure. After the serial ports are configured, the microcontroller reboots the GPS receiver, configures it to enter low-power mode when its processor is not in use, and finally puts it into sleep mode until another message is transmitted from the microcontroller.

The later nine GPS GRIL commands listed in Table 8.1 apply to actually fetching GPS data from the receiver. The specific GPS data the MCETS server can request includes Cartesian (*x*-, *y*-, and *z*-axis) position, Geodetic (latitude, longitude, and altitude) position, Cartesian (*x*-, *y*-, and *z*-axis) velocity, Geodetic (northing, easting, and height) velocity, and receiver time and date. Additionally, if any position or velocity data is requested by the server, dilution of precision and satellite statistics are automatically transmitted to the server as to provide measurements of accuracy. Finally, the raw battery voltage applied to the GPS receiver is also measured by the MCETS client, providing a way to monitor a client's battery voltage during operation (if no GPS receiver is detected by the microcontroller, the clients returns a raw battery voltage of zero volts).

As illustrated in Figure 8.4: MCETS Client GPS Data Acquisition Process, GPS data is acquired in a similar manner to the analog sensor data, however the GPS process uses all three DMA channels and the microcontroller UART peripheral (USART0). The first DMA channel (DMA-0) is configured to transfer a block of data – ASCII GRIL commands stored in Flash memory (0xA000) – to the UART0 transmit register. Conversely, the second DMA channel (DMA-1) is configured to transfer a block of received data from the UART0 receive register to a GPS dump address in RAM (0x2000) for temporary storage. This temporary storage allows the microcontroller to extract the MCETS-desired data from the GPS receiver standard data stream, which from Appendix E.4: Data Format for Standard GRIL Output Messages, includes miscellaneous information including message identification numbers, data lengths, and error-checking checksums.

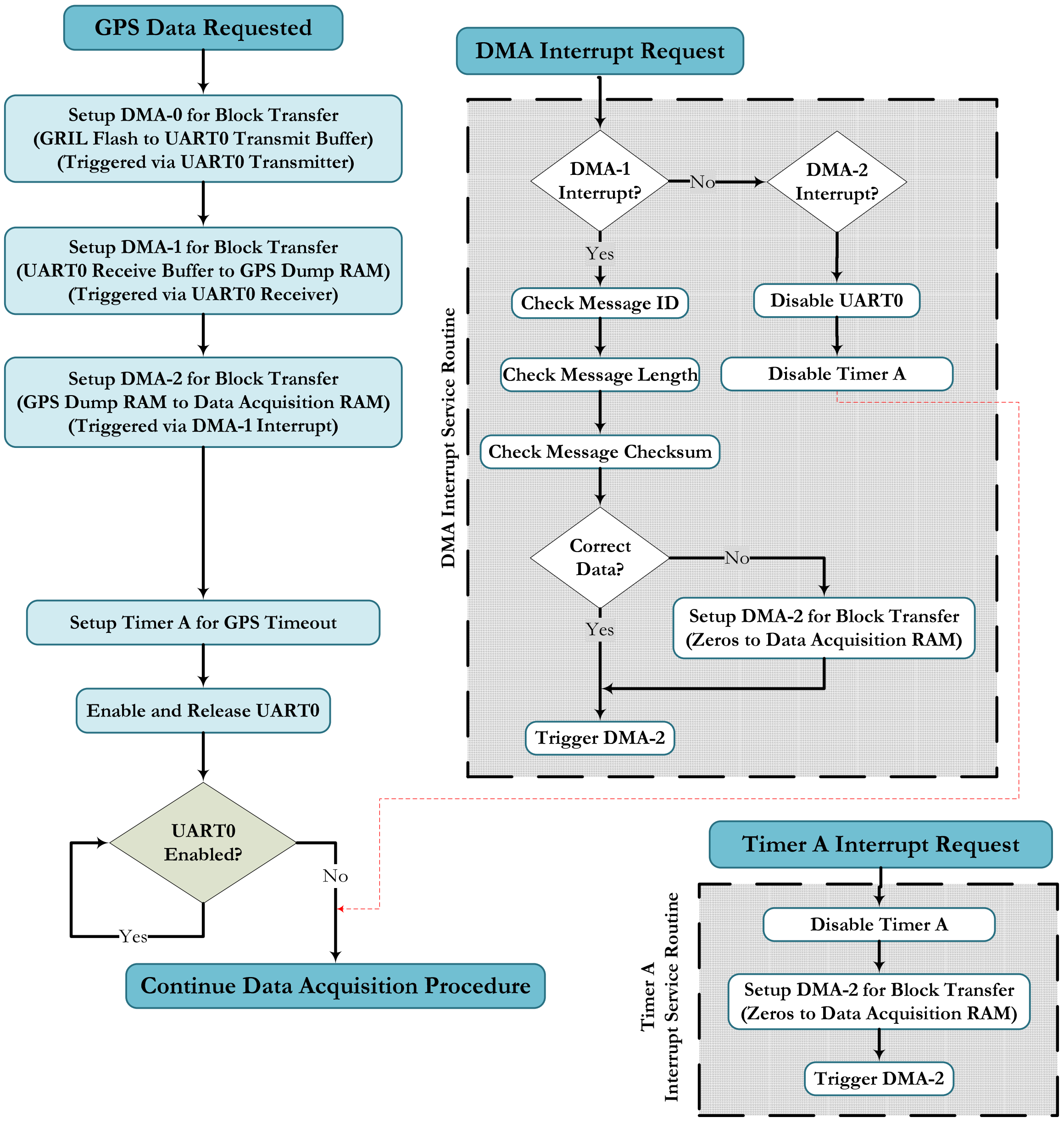

GPS Data Requested
Setup DMA-0 for Block Transfer (GRIL Flash to UART0 Transmit Buffer) (Triggered via UART0 Transmitter)
Setup DMA-1 for Block Transfer (UART0 Receive Buffer to GPS Dump RAM) (Triggered via UART0 Receiver)
Setup DMA-2 for Block Transfer (GPS Dump RAM to Data Acquisition RAM) (Triggered via DMA-1 Interrupt)
Setup Timer A for GPS Timeout
Enable and Release UART0
UART0 Enabled?
Yes
No
Continue Data Acquisition Procedure
DMA Interrupt Request
DMA Interrupt Service Routine
DMA-1 Interrupt?
No
DMA-2 Interrupt?
Yes
Check Message ID
Disable UART0
Check Message Length
Disable Timer A
Check Message Checksum
Correct Data?
No
Yes
Setup DMA-2 for Block Transfer (Zeros to Data Acquisition RAM)
Trigger DMA-2
Timer A Interrupt Request
Timer A Interrupt Service Routine
Disable Timer A
Setup DMA-2 for Block Transfer (Zeros to Data Acquisition RAM)
Trigger DMA-2

Figure 8.4: MCETS Client GPS Data Acquisition Process

Furthermore, the third DMA channel (DMA-2) is configured in the GPS data acquisition process to transfer and append the extracted GPS data from the GPS dump RAM to the data acquisition RAM that is used for storing the acquired analog sensor data. Pending that the message identification number, data length, and checksum are correct, DMA channel one triggers DMA channel two for block transfer; however, if an error or the wrong message is detected, DMA channel two is triggered to transfer a block of zeros to the data acquisition RAM. Lastly, to ensure that all three of these DMA transfers are executed successfully, the GPS data acquisition process employs a microcontroller timer peripheral (Timer A), which is configured to timeout in the event of a communication failure between the microcontroller and the GPS OEM receiver.

8.3. Data Transmission Procedure

The final procedure in the MCETS client firmware is data transmission, which is responsible for formatting the data acquired in the data acquisition procedure, transmitting it to the embedded wireless module (and therefore the MCETS server), and determining whether further data acquisition is needed. As illustrated in Appendix E.3: Packet Format for Data Transmitted from the Clients to the Server, the format for data is almost exactly as it is formatted and stored in the data acquisition RAM (0x1106). As a result, the only real formatting needed in the data transmission procedure is the appending of a sixteen-bit "data status" subpacket to the beginning of the data, and another sixteen-bit "module status" subpacket to the end of the data. Specifically, the "data status" subpacket informs the server of the transmitting client's module identification number and the number of bytes it will transmit, and the "module status" subpacket of the data acquisition rate, whether data acquisition and transmission was successful, and finally if the high-frequency 6.00 MHz crystal oscillator inadvertently failed.

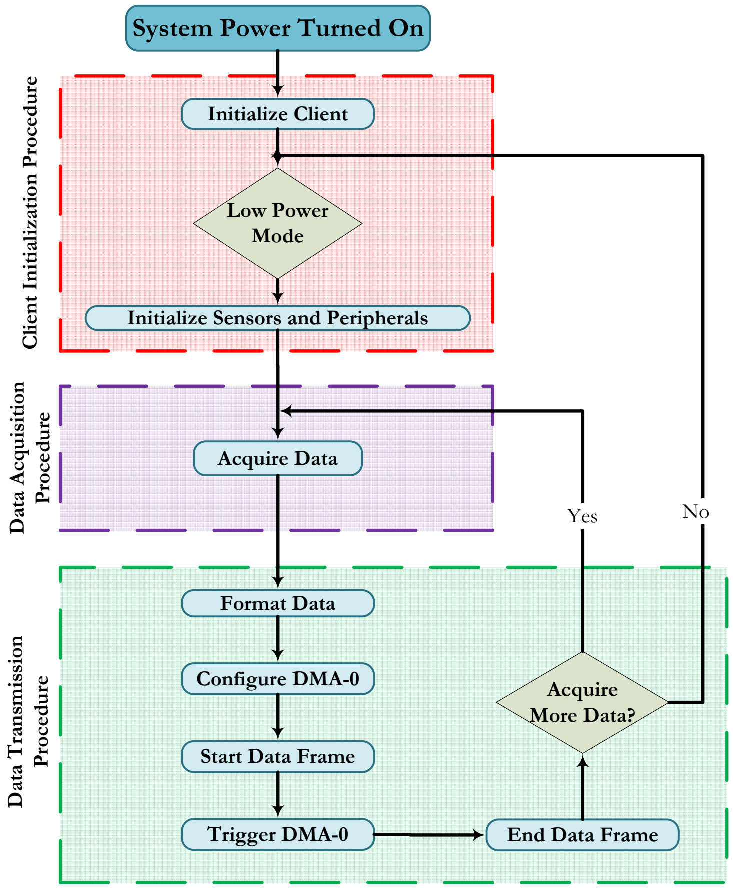


Figure 8.5: Expanded MCETS Client Data Transmission Procedure

As illustrated in Figure 8.5: Expanded MCETS Client Data Transmission Procedure, once the acquired data is properly formatted, DMA channel zero is configured to transfer the block of data stored in the data acquisition RAM to the SPI transmit buffer. Before triggering the DMA transfer, the data is framed via a microcontroller active-low signal that enables the embedded wireless module (a SPI slave device) for communications. Following this data transmission, the data transmission procedure determines whether more data needs to be acquired as specified by the server. In the event that further data acquisition is required, the firmware continues acquiring data beginning at the top of the data acquisition procedure. However if data acquisition is complete, the firmware proceeds back to the client initialization procedure to turn the sensors off and enter a low-power mode until the client is again queried by the server.

Notes

1. "GPS Receiver Interface Language (GRIL) Reference Guide", JAVAD Navigation Systems, April 2007, <http://stroage.javad.com/downloads/manuals/GRIL_Reference_Guide.pdf> (22 September 2007).

9. MCETS Server Software Development

The final step in the prototype of the MCETS is the development of software to initialize communications with the clients, collect and organize all of the transmitted client-acquired data, and to present and store this data in the best possible manner. Though other programming languages prove to be faster in terms of viewing data live, the software and graphical user interfaces for the MCETS server are written entirely in the MATLAB programming language. Since The Laboratory uses this language for virtually all of their data analysis requirements, a MATLAB-based MCETS server provides the greatest flexibility and ease of use for mission analysts. As a result, the MCETS is bundled with MATLAB software capable of configuring the communication links with clients, as well as reading, storing, and interpreting the telemetry data acquired from multiple clients.

9.1. Server-to-Client and Client-to-Server Communications

The server-to-client and client-to-server communications in the MCETS are handled by the server using MATLAB's Instrument Control Toolbox, which provides the ability to communicate with the MCETS clients (specifically the embedded wireless modules) using the TCP and IP protocols in the Internet Protocol suite. In particular, the Instrument Control Toolbox includes a built-in `tcpip` function that creates a TCP/IP object (i.e., a socket) between each individual client and the server. This object can then be opened for full-duplex communications using the built-in MATLAB function `fopen`. Furthermore, an open MATLAB TCP/IP object permits data to be transmitted to each client using the `fwrite` function, and received from each client using the `fread` function.

Once a TCP/IP object is opened using the built-in tcpip and fopen functions, several properties are made available that control the functionality of the connection, as well as how MATLAB handles data transmission over the communication channel. As illustrated in Table 9.1: MATLAB TCP/IP Object Properties, the BytesAvailableFcn property is used by MATLAB to trigger a function call to the MCETS program tcpip_cbk.m. In this MATLAB function (Appendix F.1: MCETS Main Figure Functions (MATLAB Language)), the BytesAvailable property is tested to ensure that information is available in the MATLAB TCP/IP object buffer, and then the fread function is called to read the data and cast it into an array of unsigned eight-bit integers. It is actually very important to read data as unsigned eight-bit integers because, as illustrated in Appendix E.3, the data format of client-transmitted subpackets varies from byte to byte. Finally, after the tcpip_cbk.m function reads the received data off of the TCP/IP buffer, it saves it directly to a binary file with file identification number fid.

Property Name	Description	MCETS Setting
BytesAvailable	Number of bytes available in the input buffer.	†
BytesAvailableFcn	The callback function executed when a specified amount of data is available in the input buffer, or when a terminator character is received.	{@tcpip_cbk, fid}
BytesAvailableFcnCount	The number bytes that must be available in the input buffer to generate a BytesAvailableFcn callback.	10
BytesAvailableFcnCountMode	Specifies whether the BytesAvailableFcn is generated after a	"byte"

	number of bytes are available in the input buffer, or after a terminator character is received.	
InputBufferSize	Size of the input buffer in bytes.	65,536
RemoteHost	Specifies the remote host.	‡
RemotePort	Specifies the remote port.	80
Terminator	Specifies the terminator character.	‘’
Timeout	Specifies the time to complete a read or write operation.	0.01
UserData	Specifies the data associated with the instrument object	[ ]
† Automatically set by MATLAB ‡ Varies for every MCETS client; set to the module’s identification number		

Table 9.1: MATLAB TCP/IP Object Properties

To specify exactly when the tcpip_cbk.m callback function is executed, the TCP/IP object property BytesAvailableFcnCountMode indicates whether a callback event is triggered after a specified number of bytes are received or after an ASCII terminator character is received. By setting the Terminator and BytesAvailableFcnCountMode properties to the strings ‘’ and ‘byte’, respectively, the MCETS callback function is configured to execute when the buffer reaches the size of BytesAvailableFcnCount, not when a terminator is received. Since the clients transmit raw binary data, the server cannot use ASCII terminator characters like the carriage return and line-feed because there is a relatively good chance – one in two-hundred fifty-six – that they have the same binary value. Therefore, the best solution is to trigger the tcpip_cbk.m callback whenever a set number of bytes are available in the buffer. Moreover, since the received data is constantly appended to a binary file within the tcpip_cbk.m function, the particular number of bytes in BytesAvailableFcnCount is in the end irrelevant.

Finally, for transmitting data from the MCETS server to each of the clients, the required subpackets – "wake up", "length of data acquisition", and "what data" – are generated using the MCETS program make_request_packet.m (Appendix F.2: MCETS Server-to-Client Packet Generator (MATLAB Language)). Using information obtained from one of the MCETS's graphical user interfaces, this function returns a six-element ASCII-character string for each byte in the transmitted packet. The generated string is then written to the associated client's TCP/IP object using MATLAB's fwrite function. As soon as the clients receive their corresponding data packets, they independently begin acquiring and transmitting data back to the server, therefore triggering callbacks to the MCETS tcpip_cbk function.

9.2. Data Parsing and Processing

After the raw binary data from the MCETS clients is saved to a file using the tcpip_cbk.m function, a process that occurs periodically during flight when the value in the BytesAvailableFcnCount property is exceeded, the individual data packets are parsed using the MCETS program parse_data.m. The input arguments of this function are a single client data packet (i.e., one complete sample of telemetry data), read as a string of eight-bit ASCII characters, and a ten-element Boolean array that represents the particular data requested by the server (i.e., the "what data" subpacket). As listed in Table 9.2: MATLAB-Parsed MCETS Data Structure, the program returns the packet of data parsed completely into a MATLAB structure with a field for each type of requested data. Consequently, the parse_data.m function produces a structure with a variable number of fields, depending on the particular data requested by the server via the make_request_packet.m function, that differ in both size and data format.

Structure Field Name	Data Type	Size of Data	Associated GUI Checkbox
temperature	Unsigned 16-bit Integer	2 Bytes	Temperature
pressure	Unsigned 16-bit Integer	2 Bytes	Pressure
x_accel	Unsigned 16-bit Integer	2 Bytes	Acceleration
y_accel	Unsigned 16-bit Integer	2 Bytes	Acceleration
z_accel	Unsigned 16-bit Integer	2 Bytes	Acceleration
x_gyro	Unsigned 16-bit Integer	2 Bytes	Angular Rate
y_gyro	Unsigned 16-bit Integer	2 Bytes	Angular Rate
z_gyro	Unsigned 16-bit Integer	2 Bytes	Angular Rate
x_magnet	Unsigned 16-bit Integer	2 Bytes	Magnetic Field
y_magnet	Unsigned 16-bit Integer	2 Bytes	Magnetic Field
z_magnet	Unsigned 16-bit Integer	2 Bytes	Magnetic Field
x_intemp	Unsigned 16-bit Integer	2 Bytes	†
y_intemp	Unsigned 16-bit Integer	2 Bytes	†
z_intemp	Unsigned 16-bit Integer	2 Bytes	†
Structure Field Name	Data Type	Size of Data	Associated GUI Checkbox

x pos	Double*	8 Bytes	Cartesian Position
y_pos	Double*	8 Bytes	Cartesian Position
z_pos	Double*	8 Bytes	Cartesian Position
latitude	Double*	8 Bytes	Geodetic Position
longitude	Double*	8 Bytes	Cartesian Position
altitude	Double*	8 Bytes	Geodetic Position
x_vel	Single**	4 Bytes	Cartesian Velocity
y_vel	Single**	4 Bytes	Cartesian Velocity
z_vel	Single**	4 Bytes	Cartesian Velocity
north_vel	Single**	4 Bytes	Geodetic Velocity
east vel	Single**	4 Bytes	Geodetic Velocity
atl_vel	Single**	4 Bytes	Geodetic Velocity
hdop	Single**	4 Bytes	‡
vdop	Single**	4 Bytes	‡
tdop	Single**	4 Bytes	‡
pdop	Single**	4 Bytes§	‡
gdop	Single**	4 Bytes§	‡
sats_locked	Unsigned 8-Bit Integer	1 Byte	‡
sats_available	Unsigned 8-Bit Integer	1 Byte	‡
sats_used	Unsigned 8-Bit Integer	1 Byte	‡
year	Unsigned 16-Bit Integer	2 Bytes	Receiver Time

month	Unsigned 8-Bit Integer	1 Byte	Receiver Time
day	Unsigned 8-Bit Integer	1 Byte	Receiver Time
time_ref	Unsigned 8-Bit Integer	1 Byte	Receiver Time
time_ms	Unsigned 32-Bit Integer	4 Bytes	Receiver Time
$\text{batt}_{\text{voltage}}$	ASCII Character String	8 Bytes	Any Checkbox
corrupt_str	ASCII Character String	Variable	Any Checkbox

† Any Tri-Axial Analog Inertial Sensor Data (Acceleration, Angular Rate, and Magnetic Field)

‡ Any GPS Data (Cartesian/Geodetic Position and Velocity)

§ Calculated Value not encoded in MCETS Client Data Packet

* Double-Precision Floating Point Number

** Single-Precision Floating Point Number

Table 9.2: MATLAB-Parsed MCETS Data Structure

Since the parse_data.m function only parses a single packet of data from one client, the routine is called by the MCETS program mcets2txt.m, which reads a *.mcets file (the binary file created from the tcpip_cbk.m callback function) and converts it into a human-readable text file with the extension *.txt. The most important feature of this function (Appendix F.3: MCETS Data Packet Parsing Functions (MATLAB Language)) is that it converts the binary data stored in MATLAB structures via the parse_data.m function into a meaningful text file that is organized with column headings for all of the variables and recorded data. Inside the function, parse_data.m is repeatedly called until it reaches an empty data packet, consisting of a client's eight-bit module identification number and eight zero bits that indicate data transmission has completed.

In addition to properly formatting acquired data in a human-readable file, the `mcets2txt.m` function also converts the binary data from the analog sensors and the GPS OEM receiver to numbers in base-ten. The primary reason these calculations are not executed in the `parse_data.m` function but in the `parse_data.m` function is simply to save processing time while the MCETS server receives live data through open TCP/IP connections. Since the GPS data is mainly formatted as floating-point numbers, unsigned integers, and ASCII characters, only basic conversions are needed to produce human-readable values. The analog sensor data, however, must first be converted from binary ADC12 counts to voltages, where because of the microcontrollers twelve-bit ADC with 3.3-volt and 0.0-volt references,

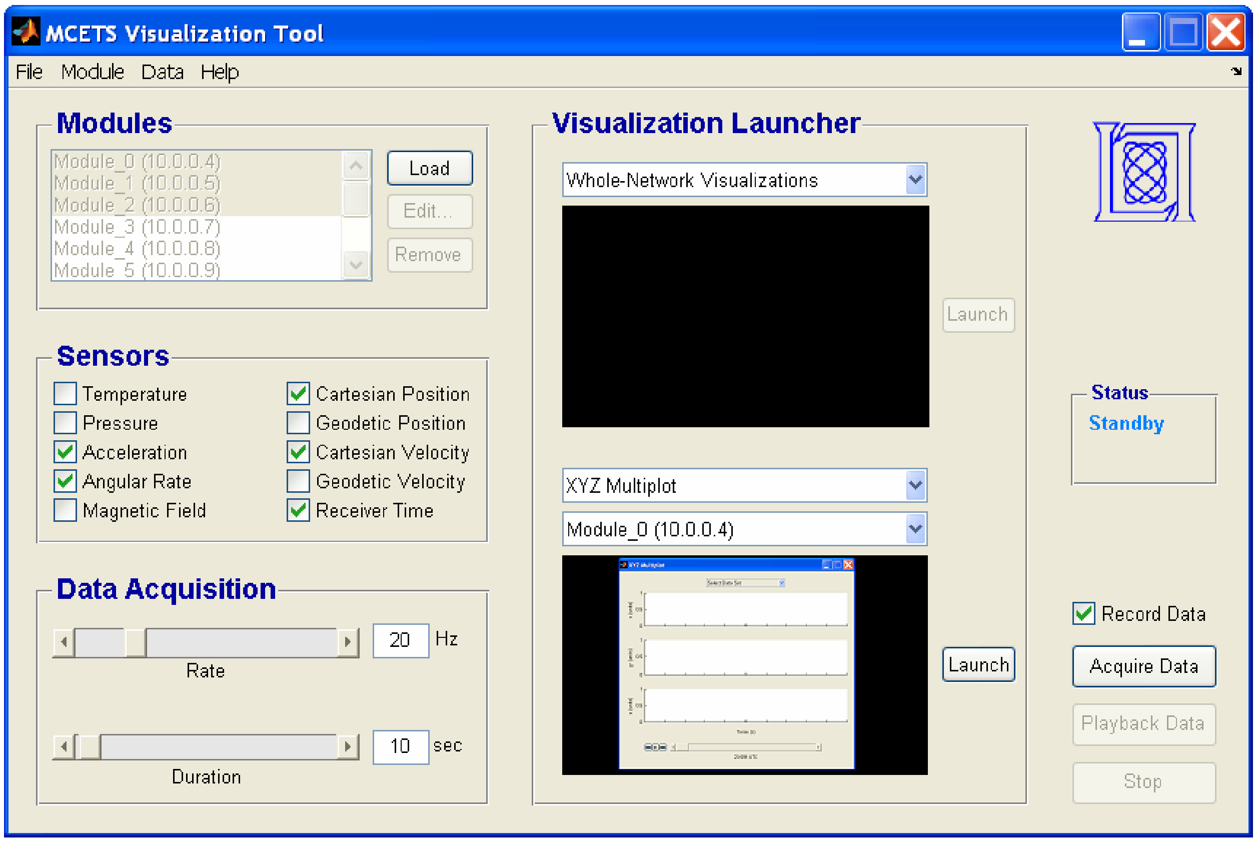
$$v_{out} = \frac{N_{ADC12}\left(V_{R_+} - V_{R_-}\right)}{2^{12} - 1} + V_{R_-} = \frac{N_{ADC12}(3.3 - 0.0)}{4095} + 0.0 = \frac{3.3N_{ADC12}}{4095}. \qquad (9.1)$$

These voltages are then up-scaled from microcontroller voltages to sensor voltages via a factor of one for the temperature sensor, and five-thirds for the pressure and tri-axial inertial sensors. Finally, the up-scaled analog sensor voltages are converted from sensor output voltages to physical measurements using the transfer equations in Section 6.

9.3. MATLAB Graphical User Interface

In order to incorporate all of the MCETS MATLAB functions in a user-friendly manner, the MCETS server employs a MATLAB-based graphical user interface that permits communicating with clients, and reading, storing, and interpreting the telemetry data that is acquired from multiple clients. When the graphical user interface is first opened via the MATLAB function call `MCETS_Main`, a default file directory is prompted for saving `*.mcets` and other miscellaneous files. After this location is specified, the main graphical user interface window in Figure 9.1: Main MCETS Graphical User Interface Window opens, containing five different panes, including a module configuration pane, sensor select pane, data acquisition settings pane, visualization launcher pane, and a system status pane.

The module configuration pane in the upper left-hand corner of the main MCETS graphical user interface window allows an analyst to load a MCETS client (e.g., module) list, remove clients from the list, and edit client settings through an "Edit Module" window. Selecting the "Edit Module" button opens another graphical user interface (Figure 9.2: MCETS "Edit Modules" Graphical User Interface Window) that facilitates adding and removing individual clients via their name and IP address. In order to prevent conflicts and errors, if clients are entered under the same name in the form `name_#` but have different IP addresses, the server automatically increments the number and adds the client to the module list. Additionally, the "Clear" button removes all clients from the module list, the "Cancel" button returns to the main MCETS window without saving any changes made, and the "OK" button saves all changes to the clients and also returns to the main window.



Module_0 (10.0.0.4)
Module_1 (10.0.0.5)
Module_2 (10.0.0.6)
Module_3 (10.0.0.7)
Module_4 (10.0.0.8)
Module_5 (10.0.0.9)

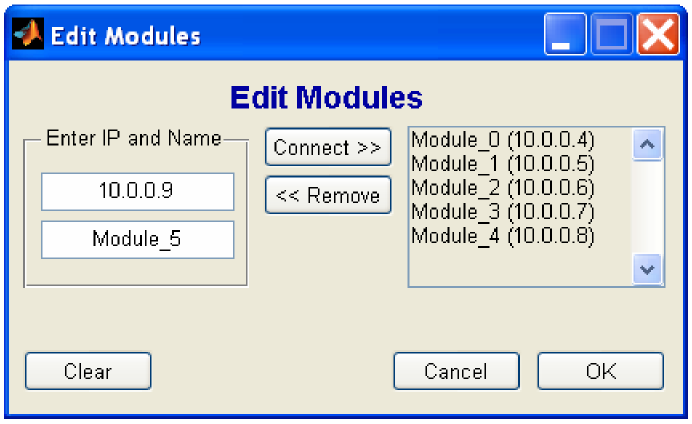
[ ] Temperature
[ ] Pressure
[✓] Acceleration
[✓] Angular Rate
[ ] Magnetic Field
[✓] Cartesian Position
[ ] Geodetic Position
[✓] Cartesian Velocity
[ ] Geodetic Velocity
[✓] Receiver Time

Rate: 20 Hz
Duration: 10 sec

Whole-Network Visualizations
XYZ Multiplot
Module_0 (10.0.0.4)

Status: Standby
[✓] Record Data

Figure 9.1: Main MCETS Graphical User Interface Window



10.0.0.9
Module_5

Module_0 (10.0.0.4)
Module_1 (10.0.0.5)
Module_2 (10.0.0.6)
Module_3 (10.0.0.7)
Module_4 (10.0.0.8)

Figure 9.2: MCETS "Edit Modules" Graphical User Interface Window

The pane directly under the module configuration pane is the sensor select pane, which via checkboxes allows for the individual selection of telemetry data acquirable by the

MCETS clients. Below this pane, the data acquisition settings pane employs sliders and text fields for setting the desired data rate and length of data acquisition. The data rate, measured in Hertz, is adjustable from one to one-hundred Hertz in data-rate increments of one. The length of data acquisition, measured in seconds, is adjustable from one to 65,535 seconds; through by simply entering zero ('0') into the duration text field, the clients can be configured to acquire data indefinitely. What's more, the changes made in the sensor select and data-acquisition setting panes apply to all of the clients entered in the module configuration pane.

The visualization launcher pane, the largest pane of the server's graphical user interface, is where mathematical plots are selected for viewing data live. These plots are separated into whole-network and client-specific visualizations, where whole-network plots incorporate data streams from multiple MCETS clients (e.g., the relative position of all of the clients to the server) and client-specific plots utilize data from only one selected client (e.g., temperature and pressure data). Essentially, each of these plots is selected via a drop-down menu and is launched in separate windows via the "Launch" button.

The final pane in the MCETS graphical user interface is the system status pane, which allows users to actually connect to the MCETS clients using the "Acquire" button, record data using the "Record" checkbox, view previously acquired data using the "Playback" button, and stop acquiring data using the "Stop" button. Selecting the data acquisition button opens TCP/IP objects for each of the selected MCETS clients, and sends the data request packet to each of the corresponding clients to initialize data acquisition. By selecting the record data checkbox, the incoming data is also accumulated and saved into a *.mcets file for each of the different modules, where the name of the client is used as the name of the file. Furthermore, by selecting the playback button, data can be loaded from a *.mcets file and viewed for analysis.

In the end, the graphical user interfaces employed by the MCETS server attempts to make the implementation of a multi-client telemetry system as straightforward and user-friendly as possible. Most of the features in the five different pains may also be performed in the window's toolbar, in which some even have a keyboard shortcut key. This toolbar also permits loading and saving client lists, starting new sessions (where all user settings are cleared), and exiting the MCETS server program. The graphical user interface also takes preventative measures to ensure as few errors as possible are generated during normal operation, particularly where user-defined values can be entered. Furthermore, another method of error prevention is through automatically enabling and disabling control objects based on predefined conditions. Ultimately, this prevents changing graphical user interface options while other options are currently being processed.

10. Conclusion

The integration of the three major components of the Multi-Client Embedded Telemetry System – the hardware, firmware, and software – proves that the concept of a low power and cost effective data acquisition system that employs multiple modules is both feasible and practical. Currently, the MCETS is a fully functional system capable of acquiring atmospheric and kinematic data at a variable data rate between one and one-hundred Hertz. Specifically, the four-layer client hardware design, including the custom-designed sensor and power supply printed circuit boards, operate flawlessly and exactly to specification. In fact, each client only draws around 2.22 watts (200 mA at 11.1 volts), which is approximately one half-watt less than the projected absolute maximum power consumption of 2.70 watts. Furthermore, the client firmware efficiently acquires server-requested data while exploiting the five low-power modes of the MSP430 microcontroller, and the MCETS server properly receives, formats, and logs telemetry data from multiple MCETS clients.

However, as with all prototype systems, there are improvements that can be made in future revisions of the MCETS to enhance the functionality and flexibility of the system. Most notably, the MATLAB server software should be rewritten in a more efficient programming language such as C or using National Instruments LabView©. Unfortunately, the current MCETS server does not facilitate live viewing of data, mainly because MATLAB cannot handle both parsing incoming data streams and processing this data in real-time. Additionally, greater care should be taken with communicating between the Javad© JNS100 GPS OEM Receiver and the MSP430-F1611 microcontroller. Currently, the GPS receiver stops responding after receiving several GRIL messages from the microcontroller, an issue believed to be associated with outdated GPS receiver firmware.

After realizing these recommended system improvements, the only phase left before the MCETS is ready for full-scale implementation is the compensation and recalibration of the sensors on the receiver. Though neglected because of a strict project deadline, the analog sensors embedded on the client sensor boards need to be tested in a controlled environment, where temperature, pressure, shock, angular rate, and magnetic field strength can be precisely monitored. By comparing the output of the MCETS sensors with known conditions in a test chamber, the MCETS clients can be compensated for temperature and pressure changes, and calibrated to output the exact atmospheric and kinematic conditions to which they are subjected. Ultimately, following strict and detailed testing and sensor calibration, the MCETS could be a complete and accurate standalone-system that could assist MIT Lincoln Laboratory accomplish their testing and analysis objectives for the Ballistic Missile Defense System.

References

"Airborne Embedded Wireless Device Server". Quatech. August 2006. <http://www.dpactech.com/docs/wireless_products/AB%20wireless%20device%20server%20module.pdf> (6 October 2007).

Becke, Georg. "Comparing Bus Solutions". Texas Instruments. February 2004. <http://focus.ti.com/lit/an/slla067a/slla067a.pdf> (5 September 2007).

Durda, Frank. "The UART: What it is and how it works". 13 January 1996. <http://www.freebsd.org/doc/en_US.ISO8859-1/articles/serial-uart/index.html#UART> (5 September 2007).

"Fundamentals of RS-232 Serial Communications". Maxim IC. 29 March 2001. <http://pdfserv.maxim-ic.com/en/an/AN83.pdf> (6 September 2007).

"Global Ballistic Missile Defense". The United States Missile Defense Agency. n.d.. <http://www.mda.mil/mdalink/pdf/bmdsbook.pdf> (15 September 2007).

"Glossary of Telemetry, Technology & Technical Terms". Texas A&M University. June 2003. <http://www.tamug.edu/labb/Technology/Glossary.htm> (26 August 2007).

"GPS Receiver Interface Language (GRIL) Reference Guide". JAVAD Navigation Systems. April 2007. <http://stroage.javad.com/downloads/manuals/GRIL_Reference_Guide.pdf> (22 September 2007).

"IP, Internet Protocol". Network Sorcery. n.d.. <http://www.networksorcery.com/enp/protocol/ip.htm> (10 September 2007).

"Javad Navigation Systems JNS100". Javad. n.d.. <http://javad.com/jns/index.html?/jns/support/manuals.html> (6 October 2007).

Kalinsky, David and Roee Kalinsky. "Introduction to Serial Peripheral Interface". Embedded Systems Design. 1 February 2002. <http://embedded.com/columns/beginerscorner/9900483?printable=true> (2 September 2007).

Kozierok, Charles M.. "Data Link Layer (Layer 2)". The TCP/IP Guide. 20 September 2005. <http:// www.tcpipguide.com/free/t_DataLinkLayerLayer2.htm> 11 September 2007.

Larijani, Casey L. GPS For Everyone. New York: American Interface Corporation, 1998.

"MSP430 Ultra-Low-Power Microcontrollers". Texas Instruments. n.d.. <http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?sectionId=95&tabId=1200&familyId=342&family=mcu> (12 September 2007).

"Navigation for Weapons". Federation of American Scientists. n.d.. <http://www.fas.org/man/dod-101/navy/docs/es310/GPS/GPS.htm> (9 September 2007).

"The Navy & Satellites: Global Positioning System (GPS)". The United States Navy. n.d.. <http://www.onr.navy.mil/Focus/spacesciences/satellites/gps.htm> (9 September 2007).

"Novell Open Enterprise Server". Novell. 1 June 2005. <http://www.novell.com/documentation/oes/pdfdoc/tcpipenu/tcpipenu.pdf> (9 September 2007).

Jon Person. "Mastering GPS Programming: Part Two". GeoFrameworks. n.d.. <http://www.geoframeworks.com/Articles/WritingApps2_3.aspx> (6 October 2007).

"RS-232 Serial Interface Pinout". 25 June 2006. <http://pinouts.ru/SerialPorts/RS232_pinout.shtml> (5 September 2007).

"RS232 Tutorial on Data Interface and Cables". ARC Electronics. n.d.. <http://www.arcelect.com/rs232.htm> (7 September 2007).

Sauchyn, D.J. (Dave). "Global Positioning Systems". The University of Regina. n.d.. <http://uregina.ca/~sauchyn/geog411/global_positioning_systems.html> (9 September 2007).

Schwerdtfeger, Martin. "SPI – Serial Peripheral Interface". June 2000. <http://www.mct.net/faq/spi.html> (2 September 2007).

"Serial Buses Information Page: SPI". n.d.. <http://www.epanorama.net/links/serialbus.html#spi> (3 September 2007).

"Si-Flex SF3000l Low-Noise Analog 3g Accelerometer". Colibrys. n.d.. <http://www.colibrys.com/files/e/pdf/inertial/data_sheet_siflex3000L.pdf> (6 October 2007).

"TCP, Transmission Control Protocol". Network Sorcery. n.d.. <http://www.networksorcery.com/enp protocol/tcp.htm> (10 September 2007).

Wikipedia Contributors. "Application Layer". Wikipedia, The Free Encyclopedia. 3 September 2007. <http://en.wikipedia.org/w/index.php?title=Application_layer&oldid=155347969> (9 September 2007).

Wikipedia Contributors. "Data Link Layer". Wikipedia, The Free Encyclopedia. 16 August 2007. <http://en.wikipedia.org/w/index.php?title=Data_link_layer&oldid=151664494> (8 September 2007).

Wikipedia Contributors. "IEEE 802.11". Wikipedia, The Free Encyclopedia. 8 September 2007. <http://en.wikipedia.org/w/index.php?title=IEEE_802.11&oldid=156466248> (9 September 2007).

Wikipedia Contributors. "Physical Layer". Wikipedia, The Free Encyclopedia. 6 September 2007. <http://en.wikipedia.org/w/index.php?title=Physical_layer&oldid=156014135> (8 September 2007).

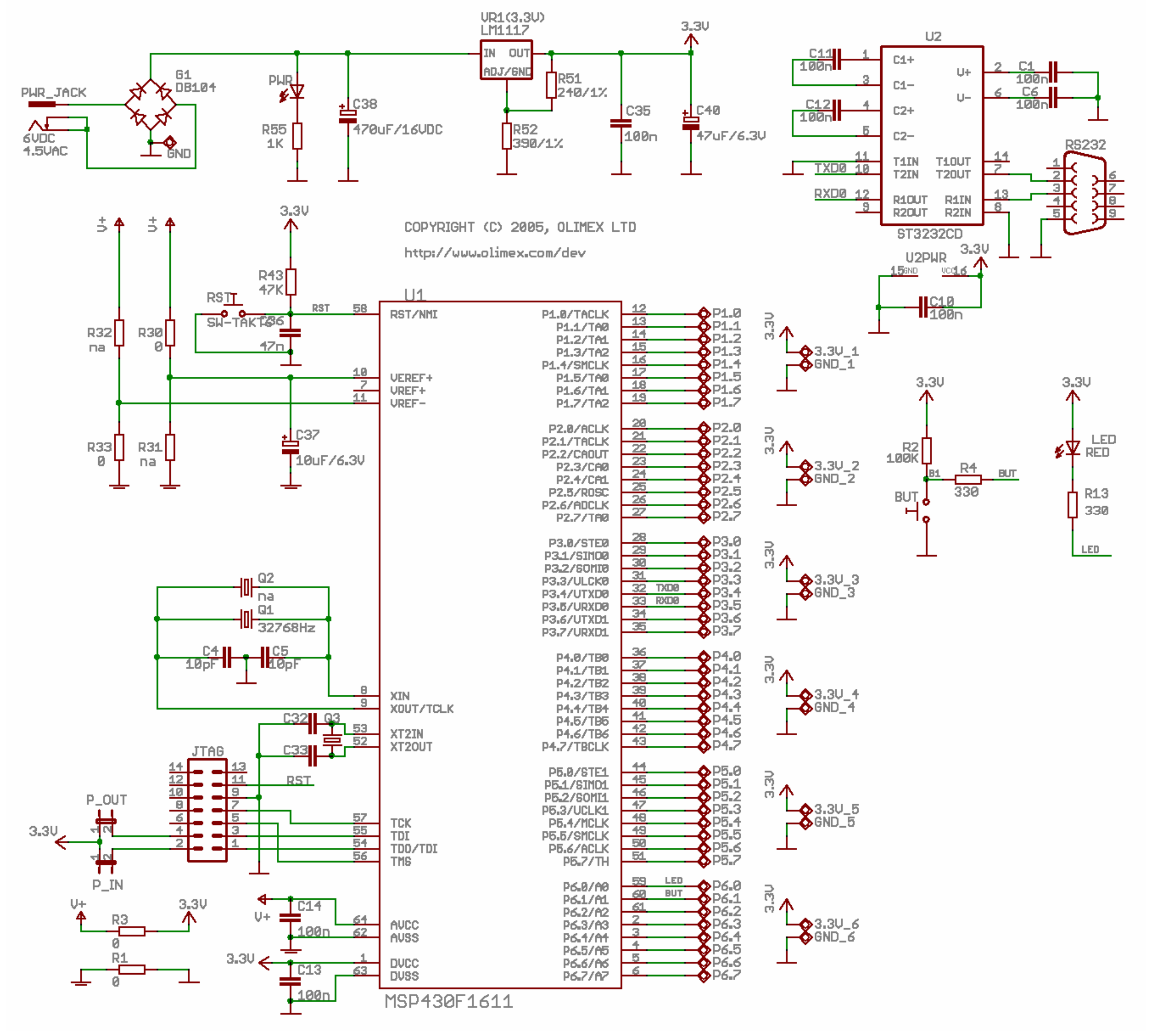
Wikipedia Contributors. "RS-232". Wikipedia, The Free Encyclopedia. 4 September 2006. <http://en.wikipedia.org/w/index.php?title=RS-232&oldid=155696305> (7 September 2007).

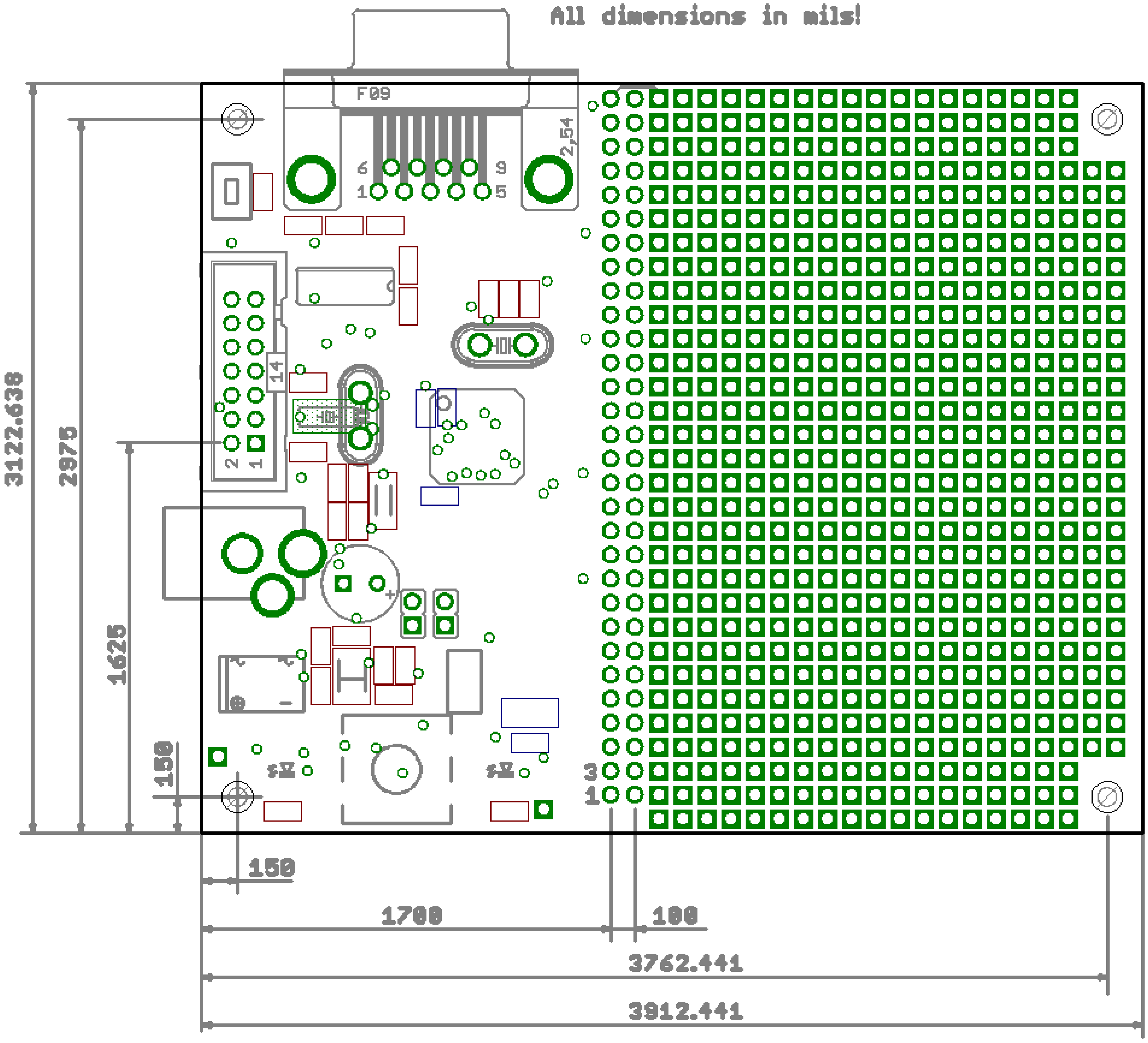
Wikipedia Contributors. "Serial Peripheral Interface". Wikipedia, The Free Encyclopedia. 6 September 2007. <http://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface_Bus&oldid=156061198> (3 September 2007).

Wikipedia Contributors. "Transport Layer". Wikipedia, The Free Encyclopedia. 7 September 2007. <http://en.wikipedia.org/w/index.php?title=Transport_layer&oldid=156366144> (8 September 2007).

Appendix A: MCETS Component Datasheets

A.1. Olimex© MSP430-P1611 Development Board Datasheet




All dimensions in mils!
F09
2,54
6
9
1
5
14
2
1
3
1
3122.638
2975
1625
150
150
1700
100
3762.441
3912.441

A.2. Maxim© DS600U Analog-Output Temperature Sensor Datasheet

DALLAS SEMICONDUCTOR MAXIM

DS600 ±0.5 Accurate Analog-Output Temperature Sensor

www.maxim-ic.com

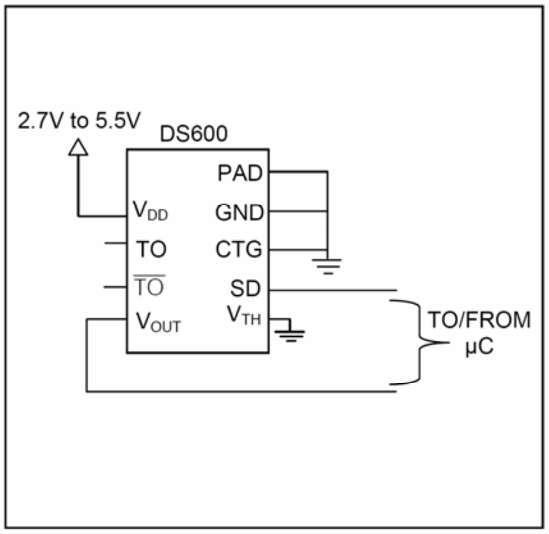
GENERAL DESCRIPTION

The DS600 is a ±0.5°C accurate analog-output temperature sensor. This accuracy is valid over its entire operating voltage range of 2.7V to 5.5V and the wide temperature range of -20°C to +100°C. The DS600 can also act as a thermostat, with user-programmable trip points. A shutdown mode enables the DS600 to be placed in a low-power standby state. The DS600 is available in an 8-pin µSOP package.

APPLICATIONS

Cold-Junction Thermocouple Compensation
Portable Medical Equipment
Thermally Sensitive Systems that Require a High-Accuracy Analog-Output Temperature Sensor

TYPICAL OPERATING CIRCUIT



FEATURES

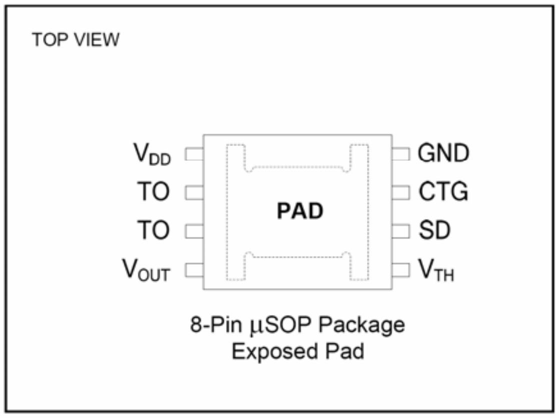
- ±0.5°C Accuracy (-20°C to +100°C)
- ±0.75°C Accuracy Over Entire Temperature Range of -40°C to +125°C
- Requires No External Components
- 6.45mV/°C Output Gain with 509mV Offset at 0°C
- 2.7V to 5.5V Supply Voltage Range
- User-Programmable Thermostat Function
- Shutdown Function Puts Device into a Low-Power Standby Mode
- Exposed Pad 8-Pin µSOP Package for Quick Thermal Response

ORDERING INFORMATION

PART	TEMP RANGE	PIN-PACKAGE
DS600U	-40°C to +125°C	Exposed Pad 8 µSOP
DS600U+	-40°C to +125°C	Exposed Pad 8 µSOP
DS600U/T&R	-40°C to +125°C	Exposed Pad 8 µSOP Tape-and-Reel
DS600U+/T&R	-40°C to +125°C	Exposed Pad 8 µSOP Tape-and-Reel

+ Denotes lead-free package.

PIN CONFIGURATION



Note: *Some revisions of this device may incorporate deviations from published specifications known as errata. Multiple revisions of any device may be simultaneously available through various sales channels. For information about device errata, click here: www.maxim-ic.com/errata.*

ABSOLUTE MAXIMUM RATINGS

Voltage Range on Any Pin (except CTG) Relative to Ground	-0.5V to +6.0V
Voltage Range on CTG Relative to Ground	-0.5 to +0.5V
Operating Temperature Range	-40°C to +125°C
Storage Temperature Range	-55°C to +125°C
Soldering Temperature (10s)	+260°C (See IPC/JEDEC J-STD-020A)
Reflow Oven Temperature	+220°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to the absolute maximum rating conditions for extended periods may affect device reliability.

DC ELECTRICAL CHARACTERISTICS

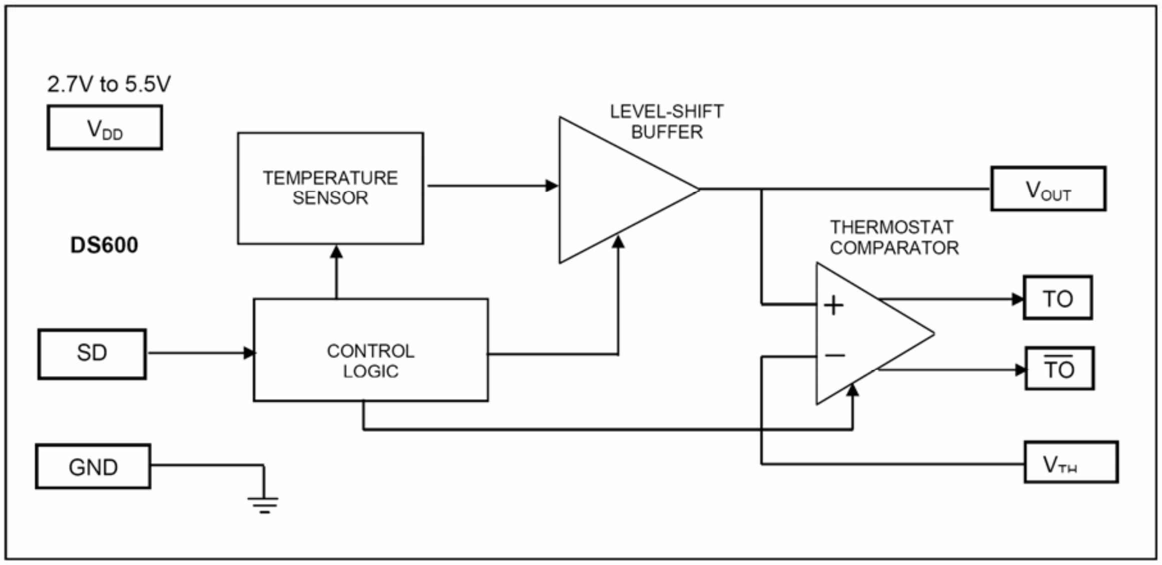
(V_{CC} = 2.7V to 5.5V, T_A = -40°C to +125°C.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Supply Voltage	V_{DD}		2.7		5.5	V
Thermometer Error	T_{ERR}	-20°C to +100°C			±0.5	°C
		-40°C to +125°C			±0.75	
Output Gain	ΔV/ΔT			6.45		mV/°C
V_{OUT} DC Offset	V_{OS}	0°C		509		mV
Low-Level Input Voltage (SD)	V_{IL}		-0.5		0.3 x V_{DD}	V
High-Level Input Voltage (SD)	V_{IH}		0.7 x V_{DD}		V_{DD} + 0.5	V
SD Input Capacitance	C_{SD}			5		pF
VTH Input Capacitance	C_{VTH}			5		pF
Low-Level Output Voltage (TO, $\overline{TO}$)	V_{OL}	4mA sink current	0		0.4	V
Supply Current	I_{DD}				140	µA
Shutdown Current	I_{SD}				2.5	µA
Input Current (V_{TH})	I_{TH}			0.01	1	µA
Input Resistance (V_{TH})	R_{TH}		5			MΩ
Leakage Current (SD)	I_L			0.01	1	µA
External Load Capacitance on V_{OUT}	C_{EL}				50	pF
V_{OUT} Source Current	I_{OSO}		10			µA
V_{OUT} Sink Current	I_{OSI}		10			µA
Output Impedance (V_{OUT})	R_{OUT}				100	Ω
Power-Up Time	$t_{POWERUP}$				10	ms
Nonlinearity					±0.2	°C
Comparator Offset					±3	mV
Comparator Response Time	t_{COMP}				20	ms

PIN DESCRIPTION

PIN	NAME	FUNCTION
1	V_{DD}	**Supply Voltage.** 2.7V to 5.5V
2	TO	**Active-High Thermostat Output.** Open-drain output transitions from low to high when the output voltage exceeds V_{TH}. In shutdown mode, (SD = 1), TO is low.
3	$\overline{TO}$	**Active-Low Thermostat Output.** Open-drain output transitions from high to low when the output voltage exceeds V_{TH}. In shutdown mode, (SD = 1), $\overline{TO}$ is high.
4	V_{OUT}	**Temperature Output.** Outputs a voltage that is proportional to the die temperature in degrees centigrade. In shutdown mode, this pin goes high-Z.
5	V_{TH}	**Thermostat Trip Voltage.** User-selectable voltage that sets the thermostat trip-point temperature. TO and $\overline{TO}$ are asserted when V_{OUT} crosses this voltage. (No on-chip hysteresis is present).
6	SD	**Shutdown.** Power consumption and thermal sensor function are controlled through SD. This pin functions as an active-high input pin. Driving this pin high puts the device in a low-power state and discontinues thermal sensing.
7	CTG	Must be connected to GND.
8	GND	**Ground.**
	PAD	**PAD.** Connect to GND or float. DO NOT CONNECT TO SUPPLY. The exposed pad is the best way to conduct temperature into the package. Connecting PAD to a ground plane can assist in properly measuring the temperature of the circuit board.

Figure 1. Block Diagram

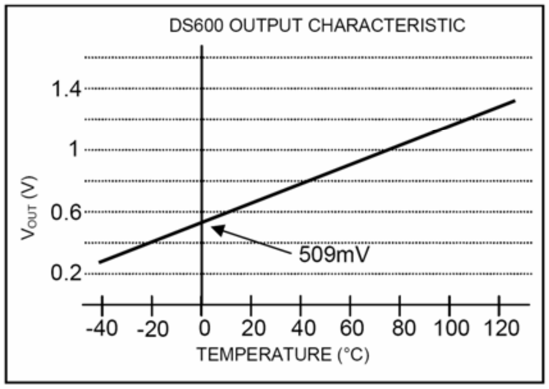


TEMPERATURE MEASUREMENT

The DS600 analog temperature sensor measures it own temperature and provides these measurements to the user in the form of an output voltage, V_{OUT}, that is proportional to degrees centigrade. The output voltage characteristic is factory-calibrated for a typical output gain ($\Delta V/\Delta T$) of +6.45mV/°C and a DC offset (V_{OS}) of 509mV. Its operating temperature range is -40°C to +125°C, corresponding to an output voltage range of 251mV to 1315mV. (V_{OUT} = Device Temperature (°C) x $\Delta V/\Delta T$ + V_{OS}). The DS600 has ±0.5°C accuracy over a -20°C to +100°C temperature range and over the full 2.7V to 5.5V voltage range. Because the output voltage is positive for the entire temperature range, there is no need for a negative supply.

Figure 2 shows the output voltage characteristic for the DS600.

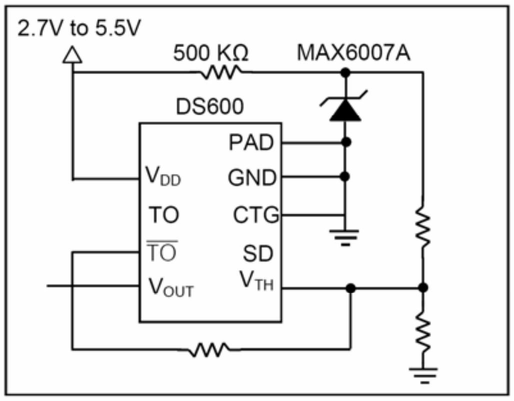
Figure 2. Output Voltage Characteristic



THERMOSTAT OPERATION

The DS600 can also be used as a thermostat with either an active-high (TO) or active-low ($\overline{\text{TO}}$) output. To function as a thermostat, a precise voltage reference equal to the desired threshold must be applied to the V_{TH} pin. When the temperature with the equivalent voltage value is reached (voltage on V_{OUT} = voltage on V_{TH}), thermostat outputs TO and $\overline{\text{TO}}$ become active. Figure 3 shows an example thermostat application circuit.

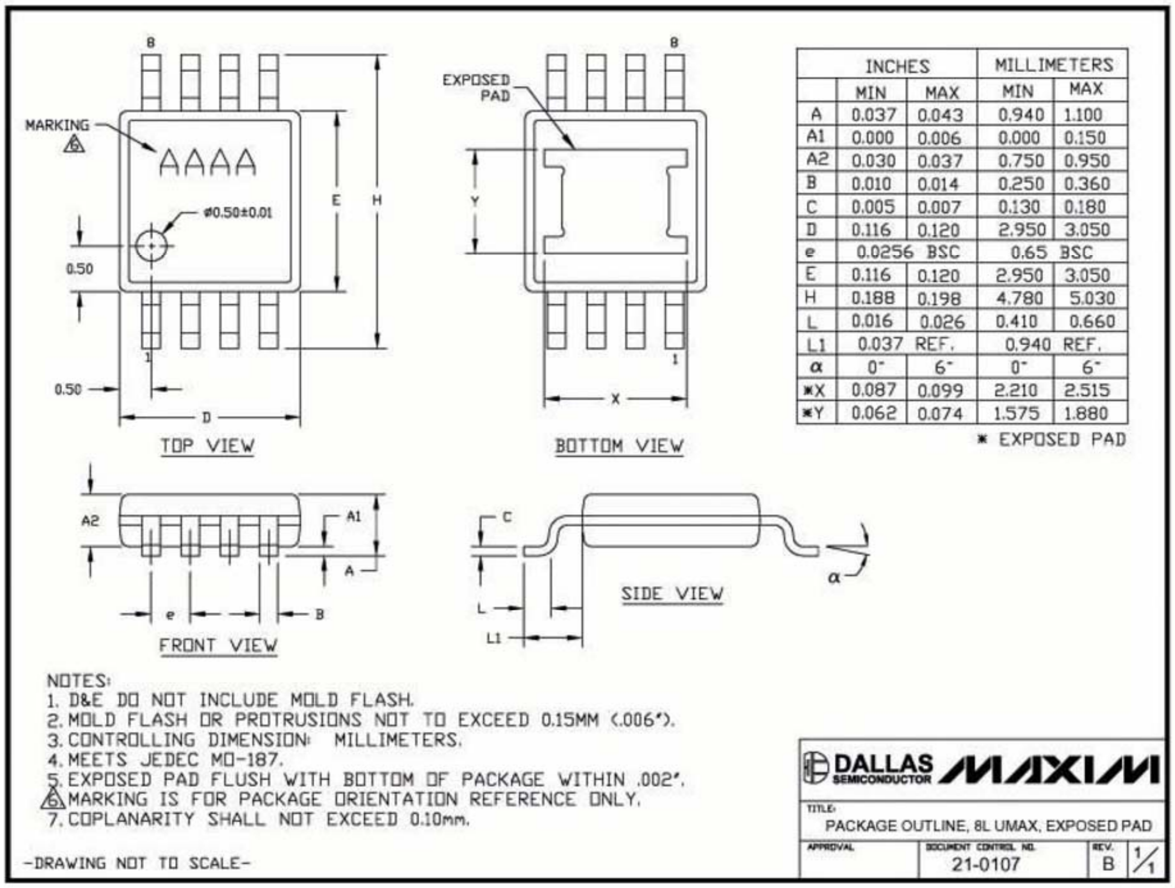
Figure 3. Thermostat Application Circuit



A.3. Motorola© MPXA4250A6U Pressure Sensor Datasheet

PACKAGE INFORMATION

For the latest package outline information, go to www.maxim-ic.com/DallasPackInfo.



	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.037	0.043	0.940	1.100
A1	0.000	0.006	0.000	0.150
A2	0.030	0.037	0.750	0.950
B	0.010	0.014	0.250	0.360
C	0.005	0.007	0.130	0.180
D	0.116	0.120	2.950	3.050
e	0.0256 BSC		0.65 BSC	
E	0.116	0.120	2.950	3.050
H	0.188	0.198	4.780	5.030
L	0.016	0.026	0.410	0.660
L1	0.037 REF.		0.940 REF.	
α	0°	6°	0°	6°
*X	0.087	0.099	2.210	2.515
*Y	0.062	0.074	1.575	1.880

* EXPOSED PAD

NOTES:
1. D&E DO NOT INCLUDE MOLD FLASH.
2. MOLD FLASH OR PROTRUSIONS NOT TO EXCEED 0.15MM (.006").
3. CONTROLLING DIMENSION: MILLIMETERS.
4. MEETS JEDEC MO-187.
5. EXPOSED PAD FLUSH WITH BOTTOM OF PACKAGE WITHIN .002".
6. MARKING IS FOR PACKAGE ORIENTATION REFERENCE ONLY.
7. COPLANARITY SHALL NOT EXCEED 0.10mm.

-DRAWING NOT TO SCALE-

DALLAS SEMICONDUCTOR MAXIM

TITLE: PACKAGE OUTLINE, 8L UMAX, EXPOSED PAD

APPROVAL	DOCUMENT CONTROL NO.	REV.	
	21-0107	B	1/1

MAXIMUM RATINGS(1)

Parametrics	Symbol	Value	Unit
Maximum Pressure(2) (P1 > P2)	P_{max}	1000	kPa
Storage Temperature	T_{stg}	−40 to +125	°C
Operating Temperature	T_A	−40 to +125	°C

NOTES:
1. T_C = 25°C unless otherwise noted.
2. Exposure beyond the specified limits may cause permanent damage or degradation to the device.

OPERATING CHARACTERISTICS (V_S = 5.1 Vdc, T_A = 25°C unless otherwise noted, P1 > P2, Decoupling circuit shown in Figure 3 required to meet electrical specifications.)

Characteristic		Symbol	Min	Typ	Max	Unit
Pressure Range(1)		P_{OP}	20	—	250	kPa
Supply Voltage(2)		V_S	4.85	5.1	5.35	Vdc
Supply Current		I_o	—	7.0	10	mAdc
Minimum Pressure Offset(3) @ V_S = 5.1 Volts	(0 to 85°C)	V_{off}	0.133	0.204	0.274	Vdc
Full Scale Output(4) @ V_S = 5.1 Volts	(0 to 85°C)	V_{FSO}	4.826	4.896	4.966	Vdc
Full Scale Span(5) @ V_S = 5.1 Volts	(0 to 85°C)	V_{FSS}	—	4.692	—	Vdc
Accuracy(6)	(0 to 85°C)	—	—	—	±1.5	$\%V_{FSS}$
Sensitivity		$\Delta V/\Delta P$	—	20	—	mV/kPa
Response Time(7)		t_R	—	1.0	—	msec
Output Source Current at Full Scale Output		I_{o+}	—	0.1	—	mAdc
Warm–Up Time(8)		—	—	20	—	msec
Offset Stability(9)		—	—	±0.5	—	$\%V_{FSS}$

NOTES:
1. 1.0 kPa (kiloPascal) equals 0.145 psi.
2. Device is ratiometric within this specified excitation range.
3. Offset (V_{off}) is defined as the output voltage at the minimum rated pressure.
4. Full Scale Output (V_{FSO}) is defined as the output voltage at the maximum or full rated pressure.
5. Full Scale Span (V_{FSS}) is defined as the algebraic difference between the output voltage at full rated pressure and the output voltage at the minimum rated pressure.
6. Accuracy (error budget) consists of the following:

• Linearity:	Output deviation from a straight line relationship with pressure over the specified pressure range.
• Temperature Hysteresis:	Output deviation at any temperature within the operating temperature range, after the temperature is cycled to and from the minimum or maximum operating temperature points, with zero differential pressure applied.
• Pressure Hysteresis:	Output deviation at any pressure within the specified range, when this pressure is cycled to and from the minimum or maximum rated pressure, at 25°C.
• TcSpan:	Output deviation over the temperature range of 0° to 85°C, relative to 25°C.
• TcOffset:	Output deviation with minimum rated pressure applied, over the temperature range of 0° to 85°C, relative to 25°C.
• Variation from Nominal:	The variation from nominal values, for Offset or Full Scale Span, as a percent of V_{FSS}, at 25°C.

7. Response Time is defined as the time for the incremental change in the output to go from 10% to 90% of its final value when subjected to a specified step change in pressure.
8. Warm–up is defined as the time required for the product to meet the specified output voltage after the Pressure has been stabilized.
9. Offset stability is the product's output deviation when subjected to 1000 hours of Pulsed Pressure, Temperature Cycling with Bias Test.

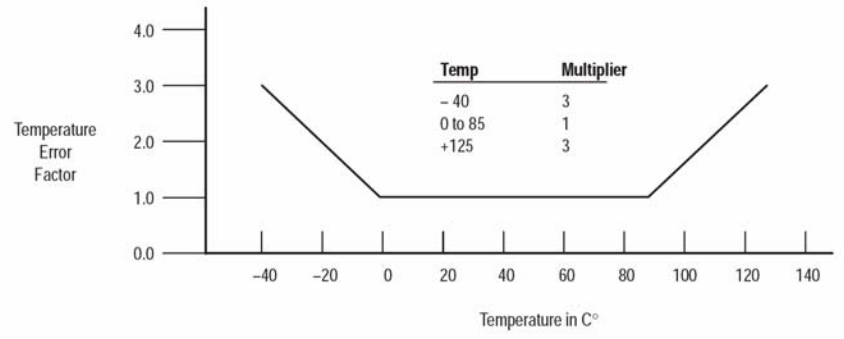
MECHANICAL CHARACTERISTICS

Characteristics	Typ	Unit
Weight, Basic Element (Case 867)	4.0	Grams
Weight, Small Outline Package (Case 482)	1.5	Grams

Transfer Function

Nominal Transfer Value: $V_{out} = V_S (P \times 0.004 - 0.04)$
+/– (Pressure Error x Temp. Factor x 0.004 x V_S)
$V_S = 5.1\ V \pm 0.25\ Vdc$

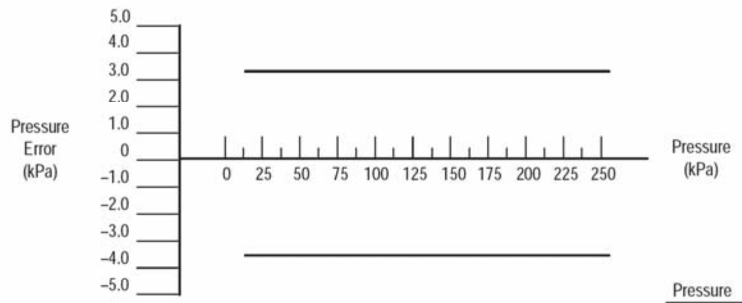
Temperature Error Band



Temp	Multiplier
– 40	3
0 to 85	1
+125	3

NOTE: The Temperature Multiplier is a linear response from 0° to –40°C and from 85° to 125°C.

Pressure Error Band



Pressure	Error (Max)
20 to 250 kPa	± 3.45 (kPa)

ORDERING INFORMATION – UNIBODY PACKAGE (CASE 867)

The MPX4250A series pressure sensors are available in the basic element package or with pressure port fittings that provide mounting ease and barbed hose connections.

Device Type/Order No.	Options	Case No.	Marking
MPX4250A	Basic Element	867	MPX4250A
MPX4250AP	Ported Element	867B	MPX4250AP

ORDERING INFORMATION – SMALL OUTLINE PACKAGE (CASE 482)

The MPXA4250A series pressure sensors are available in the basic element package or with a pressure port fitting. Two packing options are offered for each type.

Device Type/Order No.	Case No.	Packing Options	Device Marking
MPXA4250A6U	482	Rails	MPXA4250A
MPXA4250A6T1	482	Tape and Reel	MPXA4250A
MPXA4250AC6U	482A	Rails	MPXA4250A
MPXA4250AC6T1	482A	Tape and Reel	MPXA4250A

INFORMATION FOR USING THE SMALL OUTLINE PACKAGE (CASE 482)

MINIMUM RECOMMENDED FOOTPRINT FOR SURFACE MOUNTED APPLICATIONS

Surface mount board layout is a critical portion of the total design. The footprint for the surface mount packages must be the correct size to ensure proper solder connection interface between the board and the package. With the correct fottprint, the packages will self align when subjected to a solder reflow process. It is always recommended to design boards with a solder mask layer to avoid bridging and shorting between solder pads.

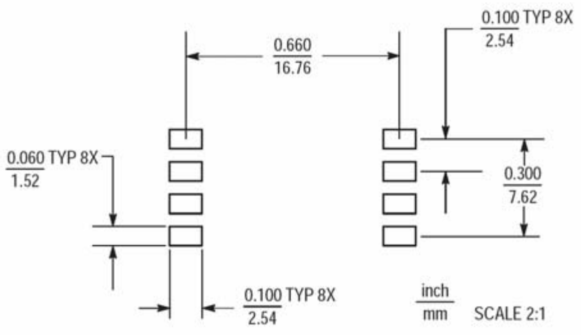
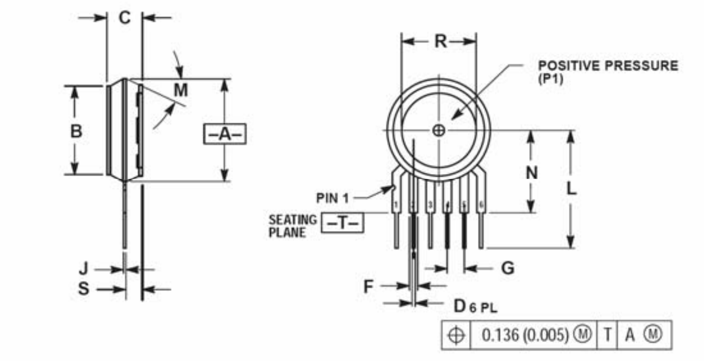


Figure 5. SOP Footprint (Case 482)

PACKAGE DIMENSIONS



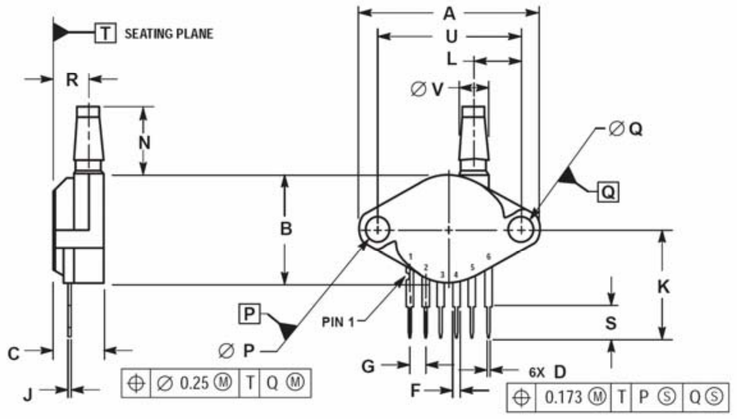
NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSION –A– IS INCLUSIVE OF THE MOLD STOP RING. MOLD STOP RING NOT TO EXCEED 16.00 (0.630).

DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.595	0.630	15.11	16.00
B	0.514	0.534	13.06	13.56
C	0.200	0.220	5.08	5.59
D	0.027	0.033	0.68	0.84
F	0.048	0.064	1.22	1.63
G	0.100 BSC		2.54 BSC	
J	0.014	0.016	0.36	0.40
L	0.695	0.725	17.65	18.42
M	30°NOM		30°NOM	
N	0.475	0.495	12.07	12.57
R	0.430	0.450	10.92	11.43
S	0.090	0.105	2.29	2.66

STYLE 1:
PIN 1. VOUT
2. GROUND
3. VCC
4. V1
5. V2
6. VEX

CASE 867
ISSUE N

UNIBODY, BASIC ELEMENT (A)



NOTES:
1. DIMENSIONS ARE IN MILLIMETERS.
2. DIMENSIONS AND TOLERANCES PER ASME Y14.5M, 1994.

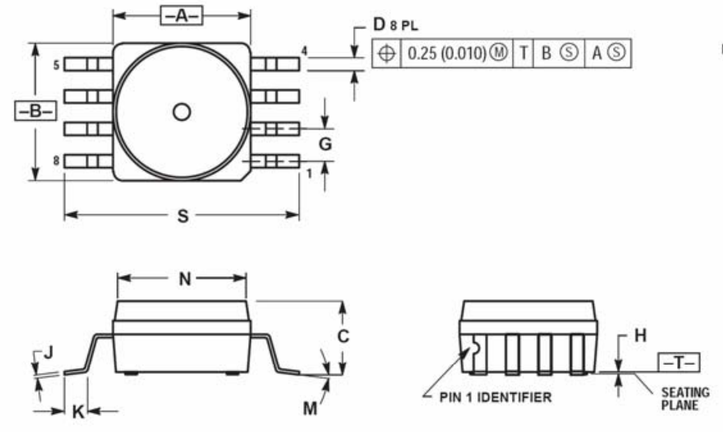
DIM	MILLIMETERS	
	MIN	MAX
A	29.08	29.85
B	17.4	18.16
C	7.75	8.26
D	0.68	0.84
F	1.22	1.63
G	2.54 BSC	
J	0.36	0.41
K	17.65	18.42
L	7.37	7.62
N	10.67	11.18
P	3.89	4.04
Q	3.89	4.04
R	5.84	6.35
S	5.59	6.1
U	23.11 BSC	
V	4.62	4.93

STYLE 1:
PIN 1. V_{OUT}
2. GROUND
3. V_{CC}
4. V1
5. V2
6. V_{EX}

CASE 867B
ISSUE E

UNIBODY, PRESSURE SIDE PORTED (AP)

PACKAGE DIMENSIONS – continued

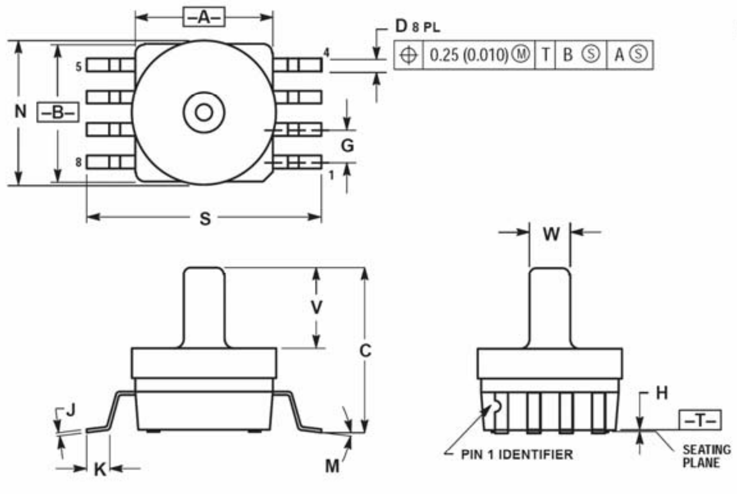


NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSION A AND B DO NOT INCLUDE MOLD PROTRUSION.
4. MAXIMUM MOLD PROTRUSION 0.15 (0.006).
5. ALL VERTICAL SURFACES 5° TYPICAL DRAFT.

	INCHES		MILLIMETERS	
DIM	MIN	MAX	MIN	MAX
A	0.415	0.425	10.54	10.79
B	0.415	0.425	10.54	10.79
C	0.212	0.230	5.38	5.84
D	0.038	0.042	0.96	1.07
G	0.100 BSC		2.54 BSC	
H	0.002	0.010	0.05	0.25
J	0.009	0.011	0.23	0.28
K	0.061	0.071	1.55	1.80
M	0°	7°	0°	7°
N	0.405	0.415	10.29	10.54
S	0.709	0.725	18.01	18.41

CASE 482
ISSUE O

SMALL OUTLINE PACKAGE, BASIC ELEMENT



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSION A AND B DO NOT INCLUDE MOLD PROTRUSION.
4. MAXIMUM MOLD PROTRUSION 0.15 (0.006).
5. ALL VERTICAL SURFACES 5° TYPICAL DRAFT.

	INCHES		MILLIMETERS	
DIM	MIN	MAX	MIN	MAX
A	0.415	0.425	10.54	10.79
B	0.415	0.425	10.54	10.79
C	0.500	0.520	12.70	13.21
D	0.038	0.042	0.96	1.07
G	0.100 BSC		2.54 BSC	
H	0.002	0.010	0.05	0.25
J	0.009	0.011	0.23	0.28
K	0.061	0.071	1.55	1.80
M	0°	7°	0°	7°
N	0.444	0.448	11.28	11.38
S	0.709	0.725	18.01	18.41
V	0.245	0.255	6.22	6.48
W	0.115	0.125	2.92	3.17

CASE 482A
ISSUE A

SMALL OUTLINE PACKAGE, PRESSURE SIDE PORTED

How to reach us:
USA / EUROPE / Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217. 1–303–675–2140 or 1–800–441–2447

JAPAN: Motorola Japan Ltd.; SPS, Technical Information Center, 3–20–1, Minami–Azabu. Minato–ku, Tokyo 106–8573 Japan. 81–3–3440–3569

Technical Information Center: 1–800–521–6274

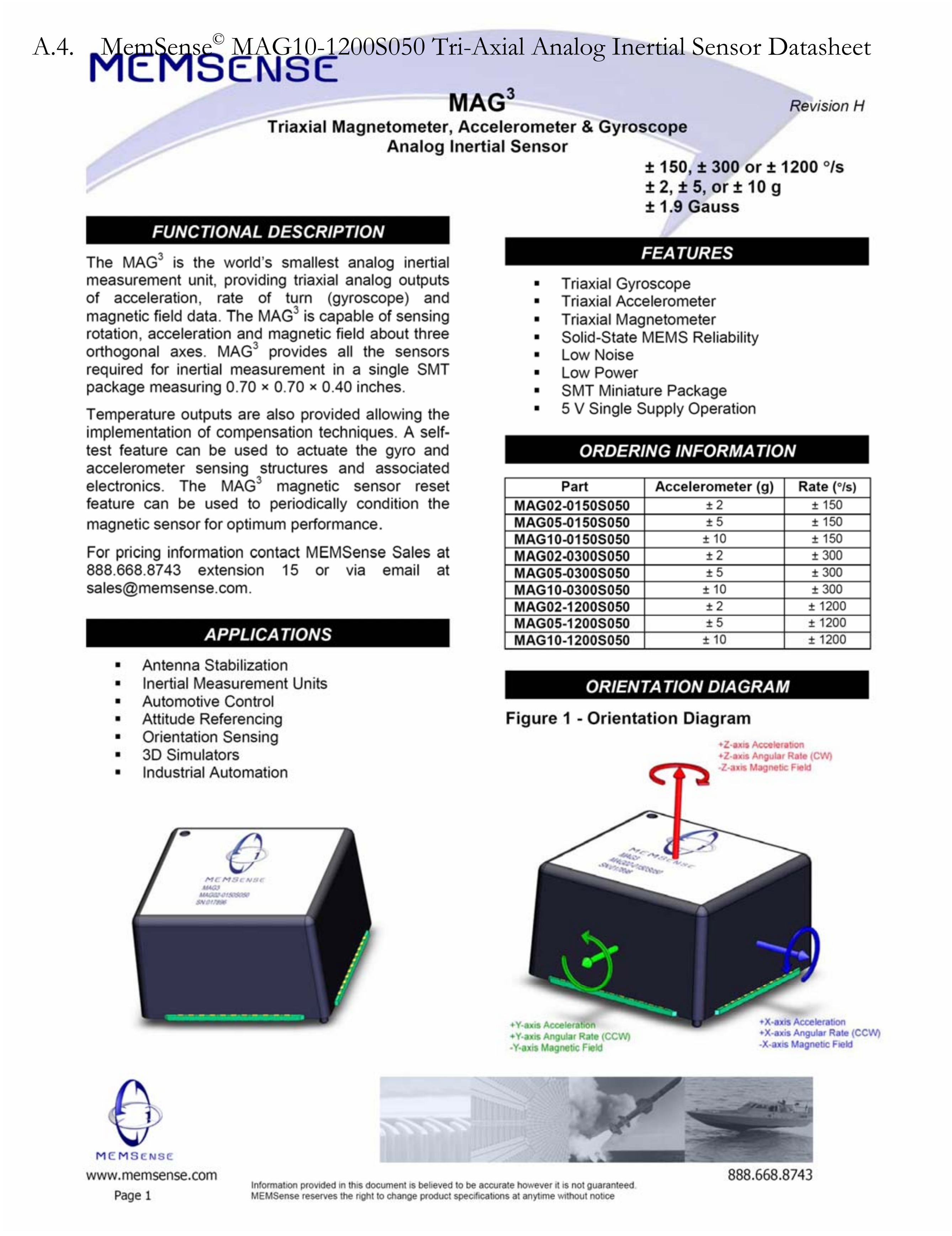
ASIA / PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2, Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852–26668334

HOME PAGE: http://www.motorola.com/semiconductors/

MOTOROLA

◊

A.4. MemSense© MAG10-1200S050 Tri-Axial Analog Inertial Sensor Datasheet



MEMSENSE

MAG^3

Revision H

Triaxial Magnetometer, Accelerometer & Gyroscope Analog Inertial Sensor

± 150, ± 300 or ± 1200 °/s
± 2, ± 5, or ± 10 g
± 1.9 Gauss

FUNCTIONAL DESCRIPTION

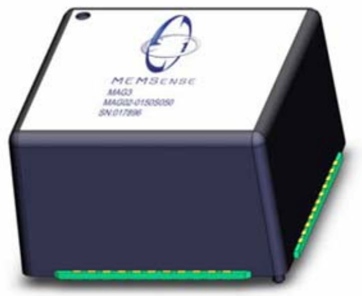
The MAG^3 is the world's smallest analog inertial measurement unit, providing triaxial analog outputs of acceleration, rate of turn (gyroscope) and magnetic field data. The MAG^3 is capable of sensing rotation, acceleration and magnetic field about three orthogonal axes. MAG^3 provides all the sensors required for inertial measurement in a single SMT package measuring 0.70 × 0.70 × 0.40 inches.

Temperature outputs are also provided allowing the implementation of compensation techniques. A self-test feature can be used to actuate the gyro and accelerometer sensing structures and associated electronics. The MAG^3 magnetic sensor reset feature can be used to periodically condition the magnetic sensor for optimum performance.

For pricing information contact MEMSense Sales at 888.668.8743 extension 15 or via email at sales@memsense.com.

APPLICATIONS

- Antenna Stabilization
- Inertial Measurement Units
- Automotive Control
- Attitude Referencing
- Orientation Sensing
- 3D Simulators
- Industrial Automation



FEATURES

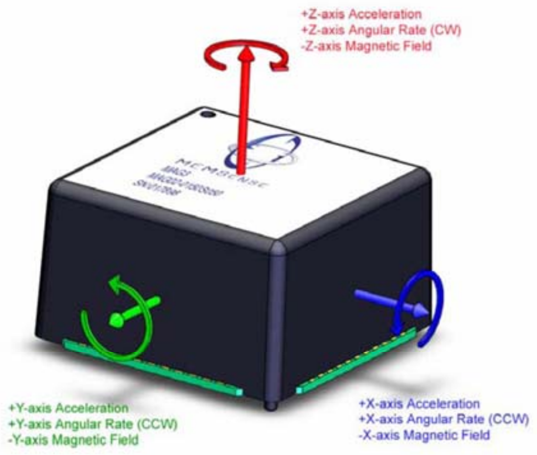
- Triaxial Gyroscope
- Triaxial Accelerometer
- Triaxial Magnetometer
- Solid-State MEMS Reliability
- Low Noise
- Low Power
- SMT Miniature Package
- 5 V Single Supply Operation

ORDERING INFORMATION

Part	Accelerometer (g)	Rate (°/s)
MAG02-0150S050	± 2	± 150
MAG05-0150S050	± 5	± 150
MAG10-0150S050	± 10	± 150
MAG02-0300S050	± 2	± 300
MAG05-0300S050	± 5	± 300
MAG10-0300S050	± 10	± 300
MAG02-1200S050	± 2	± 1200
MAG05-1200S050	± 5	± 1200
MAG10-1200S050	± 10	± 1200

ORIENTATION DIAGRAM

Figure 1 - Orientation Diagram



MEMSENSE

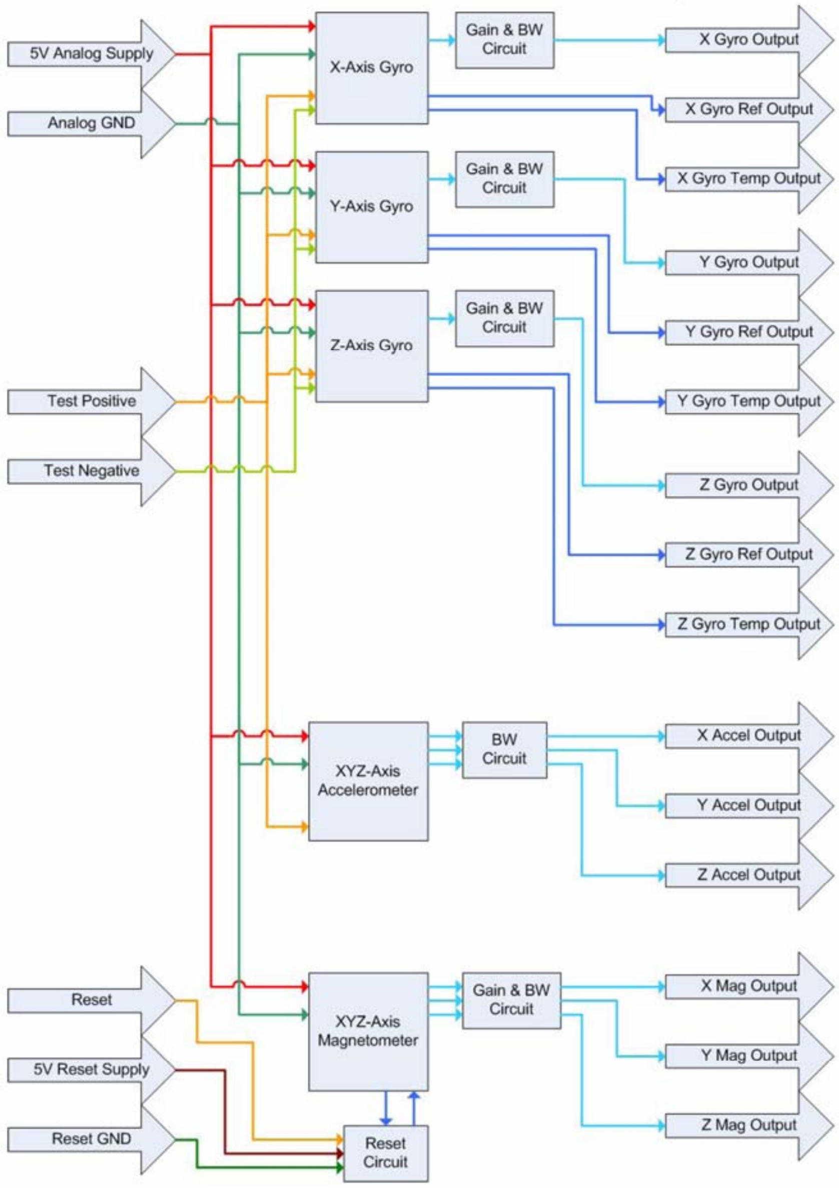
www.memsense.com

Page 1

Information provided in this document is believed to be accurate however it is not guaranteed.
MEMSense reserves the right to change product specifications at anytime without notice

888.668.8743

Figure 2 - Functional Block Diagram



MEMSENSE

MAG^3

Revision H

Triaxial Magnetometer, Accelerometer & Gyroscope
Analog Inertial Sensor

± 150, ± 300 or ± 1200 °/s
± 2, ± 5, or ± 10 g
± 1.9 Gauss

Table 1 – Specifications

Parameter	Specification			Units	Conditions
Sensor					
Operating Voltage Range	4.75 to 5.25			V	
Supply Current	30, (35)			mA	Typical, (Maximum)
Mass	5			Grams	Maximum
Commercial Temperature Range	0 to +70			°C	Temperature for max and min specs.
Military Temperature Range	-40 to +85			°C	Request quotation for 100% test.
Accelerometers	**MAG02**	**MAG05**	**MAG10**		
Range	± 2	± 5	± 10	g	
Sensitivity	1000	400	400	mV/g	Ratiometric to supply voltage
Offset Vs Temp	±150	± 60	± 31	mV	0 to 70 °C
Noise X and Z	35	35	35	μg/Hz^{½}	
Noise Y	65	65	65	μg/Hz^{½}	
Bandwidth [1]	50	50	50	Hz	Factory set 3dB point
Nonlinearity	± 0.4, (± 1.0)	± 0.4, (± 1.0)	± 0.4, (± 1.0)	% of FS	Typical, (Maximum)
Cross Axis Sensitivity	2	2	2	%	
Rate Output	**0150S050**	**0300S050**	**1200S050**		
Dynamic Range	±150	± 300	±1200	°/s	Full scale range over specified temperature
Sensitivity	12.5	5.0	1.25	mV/°/s	
Nonlinearity	0.1	0.1	0.1	% of FS	Best fit straight line
Zero Rate	2.50	2.50	2.50	V	
Turn On Time	35	35	35	ms	Power on to ± ½ °/s of Final
Rate Noise Density	0.05	0.1	0.1	°/s/Hz^{½}	
Bandwidth [1]	50	50	50	Hz	Factory set 3dB point
Cross Axis Sensitivity	1	1	1	%	
Vibration Rectification	$20e^{-6}$	$20e^{-6}$	$20e^{-6}$	°/s/(m/s^2)^2	0 – 20 kHz
Rate Reference Output					
Voltage Value	2.5			V	
Power Supply Rejection	60			db	4.75 Vs to 5.25 Vs
Temperature Drift	5.0			mV	Deviation from 25°C
Temperature Output					
Voltage at 25 °C	2.50			V	
Scale Factor	8.4			mV/°C	
Magnetic Field					
Dynamic Range	± 1.9			gauss	
Sensitivity Drift	2700			ppm/°C	
Sensitivity	1.0			V/gauss	
Nonlinearity	0.5			% of FS	Best fit straight line
Noise Density	68			nV/Hz^{½}	
Bandwidth [2]	50			Hz	Magnetic signal
Cross Axis Sensitivity	3			%	
Absolute Maximum Ratings					
Acceleration Powered	2000 max			*g*	Any axis 0.5 ms
Vdd	-0.3, +6.0			V	Minimum, Maximum
Operating Temperature	-40 to +85			°C	
Storage Temperature	-65 to +150			°C	
Typical Values at 25 °C, Vdd = 5.0V, 0 °/s unless otherwise noted					

1. Other bandwidth configurations are available upon request.
2. Addition of external 2.2uF capacitor to ground sets each magnetic signal's bandwidth to 50Hz. $F_{3db} = 1/(2*\pi*1.5k*C)$

MAG^{3}

Revision H

Triaxial Magnetometer, Accelerometer & Gyroscope
Analog Inertial Sensor

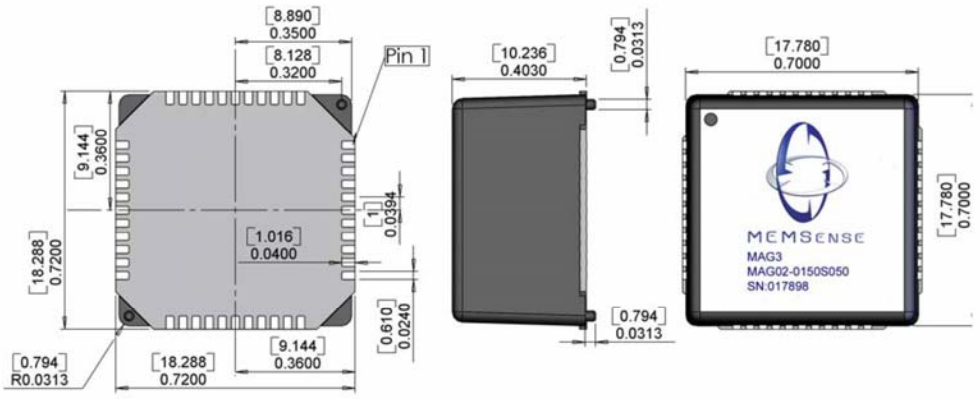
± 150, ± 300 or ± 1200 °/s
± 2, ± 5, or ± 10 g
± 1.9 Gauss

Table 2 - Pin Function Descriptions

Pin No.	Name	Function
1	XREF	X axis analog precision reference output.
2	XRATE	X axis analog rate signal output.
3	ZREF	Z axis analog precision reference output.
4	ZRATE	Z axis analog rate signal output.
5	TEMPZ	Analog temperature voltage output, Z gyro.
6	AGND	Analog power supply return.
7	TEMPX	Analog temperature voltage output, X gyro.
8	TEMPY	Analog temperature voltage output, Y gyro.
9	XMAG	X axis analog magnetic signal output
10	YMAG	Y axis analog magnetic signal output
11	ZMAG	Z axis analog magnetic signal output
12-22		No connect (open)[1]
23	MGND	Magnetic sensor reset circuit ground
24	MAG RESET	Magnetic reset input
25	MGND	Magnetic sensor reset circuit ground
26	VDDM	Magnetic sensor reset power supply
27-35		No connect (open) [1]
36	AGND	Analog power supply return.
37	VDDA	Analog power supply.
38	TESTN	High-level activated digital input stimulating X, Y and Z rate to Ref - 660mV. [2]
39	TESTP	High-level activated digital input stimulating X, Y and Z rate to Ref +660mV. [2]
40	YACCEL	Y axis analog acceleration signal output.
41	ZACCEL	Z axis analog acceleration signal output.
42	XACCEL	X axis analog acceleration signal output.
43	YREF	Y axis analog precision reference output.
44	YRATE	Y axis analog rate signal output.

1. Physical solder connection recommended.
2. The 300°/s and 1200°/s rate sensor will produce a 270 mV and 67.5 mV output change respectively.
3. **Do Not Ground 2.5V Precision Reference Outputs, Damage to the Device May Occur (Recommend floating or the use of a 20k resistor or higher)**

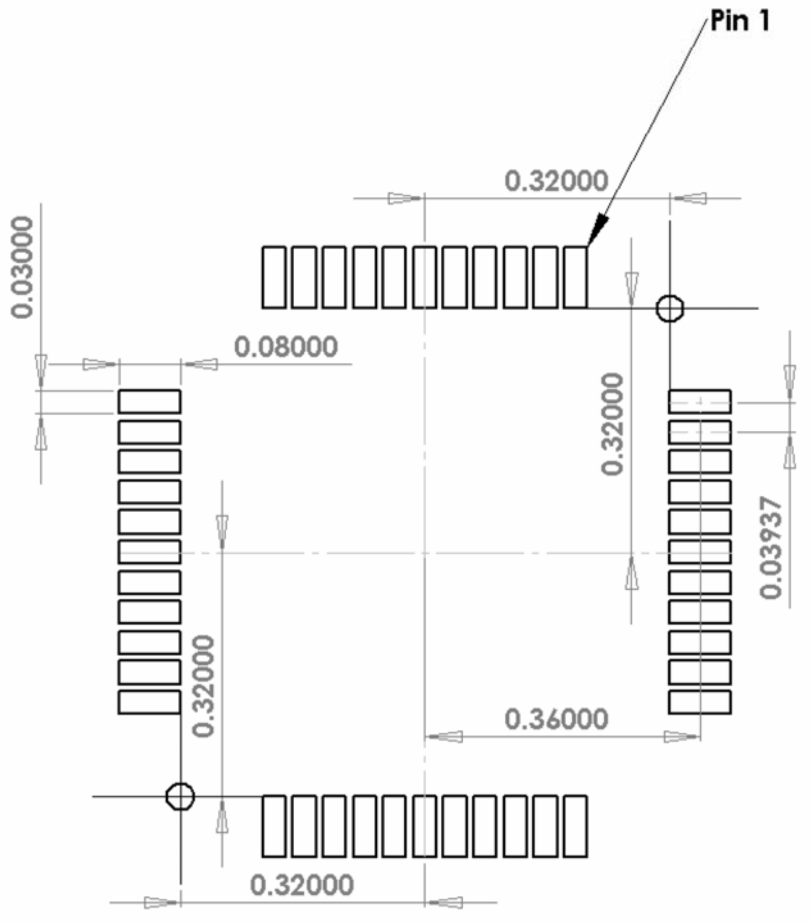
Figure 3 – Physical Dimensions



All dimensions in [mm] inches - Hand solder attachment recommended

888.668.8743

Figure 4 – Recommended PCB Pattern



Dimensions in Inches

MAG RESET FUNCTION

The accuracy of the MAG3's triaxial magnetometer may be degraded after exposure to strong magnetic fields of 20 gauss or larger. Such fields may cause a decrease in the sensor's sensitivity, linearity or a complete "stuck" output. For this reason, the MAG3 includes a set/reset circuit that regains magnetic sensing accuracy after exposure to strong magnetic fields. The mag reset function should be performed as an initialization process for the MAG3.

The set or reset operations can be initiated via the Mag Reset pin (pin 24) which is a 5 volt TTL input. The mag reset function begins with the Mag Reset pin held at 5V through a pull up resistor followed by a transition to 0V, which performs the reset. After a delay of 1.2 ms, the Mag Reset pin should be returned to its 5V state, which performs the set. A graphical depiction of the mag reset waveform is shown in Figure 5 below.

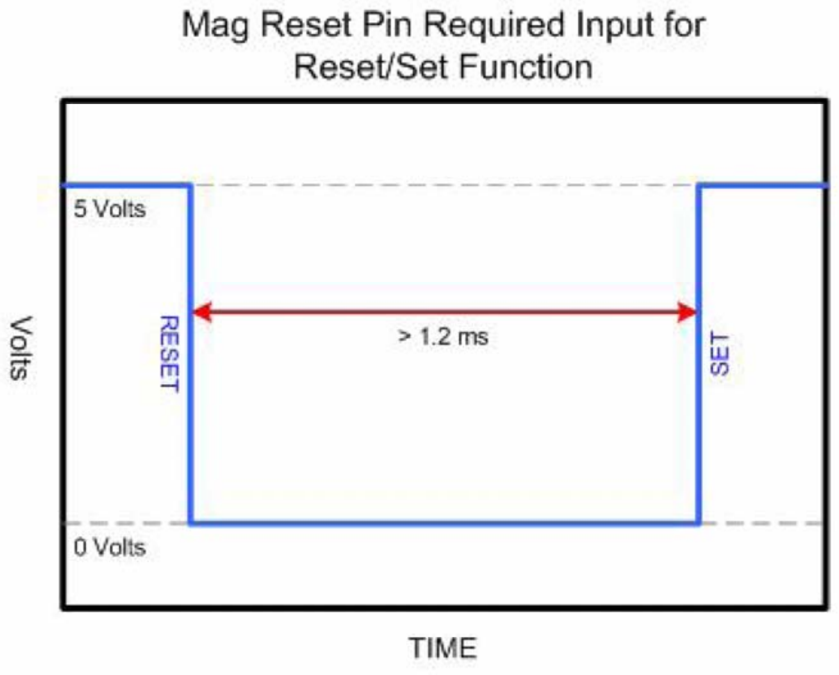
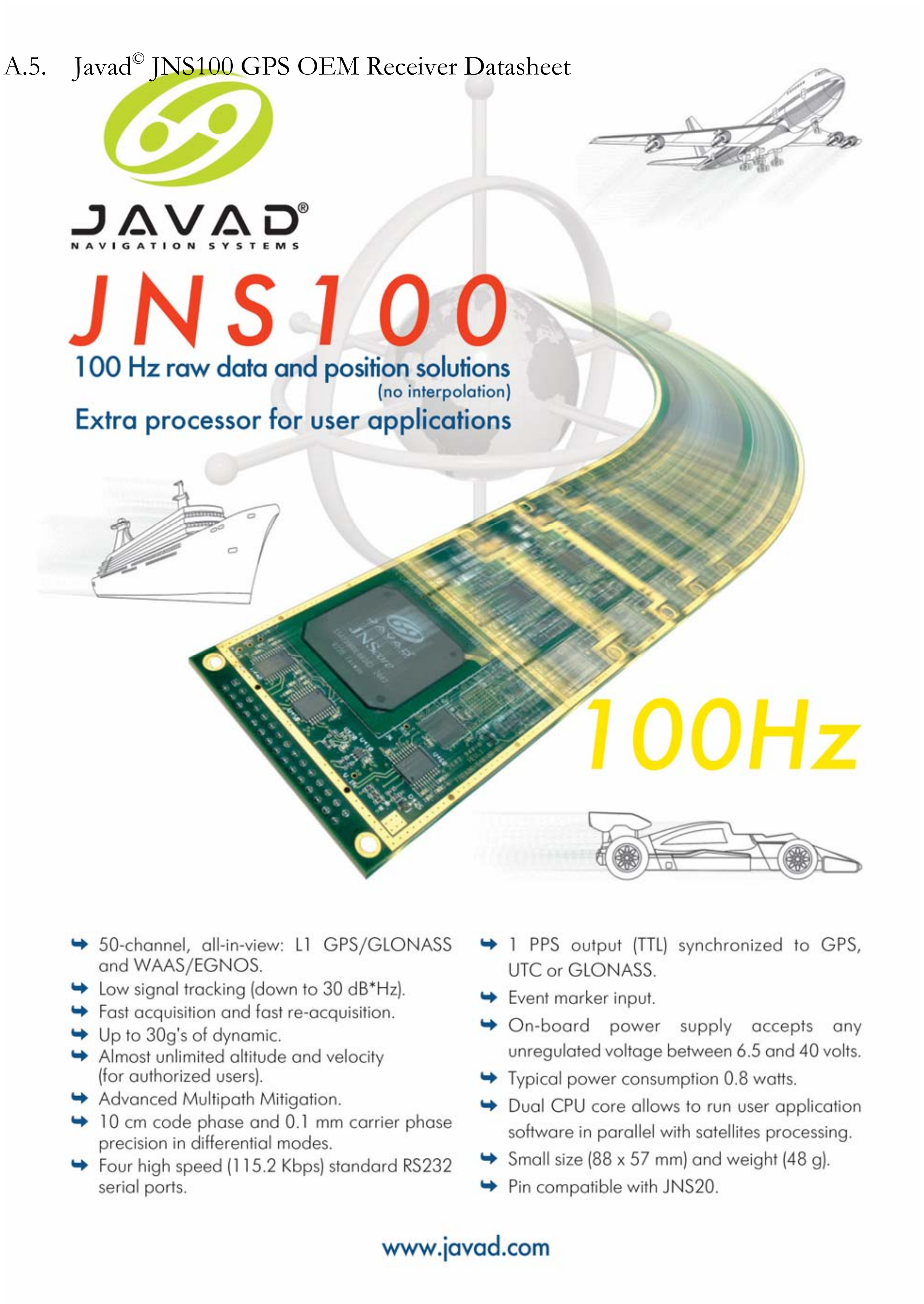


Figure 5 – Pin 24 Mag Reset function required waveform.

A.5. Javad© JNS100 GPS OEM Receiver Datasheet



JAVAD®
NAVIGATION SYSTEMS

JNS100

100 Hz raw data and position solutions
(no interpolation)
Extra processor for user applications

100Hz

- 50-channel, all-in-view: L1 GPS/GLONASS and WAAS/EGNOS.
- Low signal tracking (down to 30 dB*Hz).
- Fast acquisition and fast re-acquisition.
- Up to 30g's of dynamic.
- Almost unlimited altitude and velocity (for authorized users).
- Advanced Multipath Mitigation.
- 10 cm code phase and 0.1 mm carrier phase precision in differential modes.
- Four high speed (115.2 Kbps) standard RS232 serial ports.
- 1 PPS output (TTL) synchronized to GPS, UTC or GLONASS.
- Event marker input.
- On-board power supply accepts any unregulated voltage between 6.5 and 40 volts.
- Typical power consumption 0.8 watts.
- Dual CPU core allows to run user application software in parallel with satellites processing.
- Small size (88 x 57 mm) and weight (48 g).
- Pin compatible with JNS20.

www.javad.com

JNS100 OEM Board

Tracking Features
- 50-channel, all-in-view: L1 GPS/GLONASS and WAAS/EGNOS.
- Low signal tracking (down to 30 db*Hz)
- Fast acquisition (warm start <10 sec) and fast re-acquisition (<1 sec)
- Up to 30 g's of dynamic
- Almost unlimited altitude and velocity (for authorized users)
- Advanced Multipath mitigation

Data Features
- Up to 100 Hz update rate for real time position and raw data (code and carrier)
- 10 cm code phase and 0.1 mm carrier phase precision
- RTCM SC104 version 2.3 input
- NMEA 0183 version 3.0 output
- Geoid and Magnetic Variation models
- RAIM
- Different DATUMs support
- Output of grid coordinates

Input/Output
- Four high speed (115.2 Kbps) standard (-6 to +6 voltage swing) RS232 serial ports
- 1 PPS output (TTL) synchronized to GPS, UTC, or GLONASS
- Event marker input
- 2 external LED drivers

Electrical
- On-board power supply accepts any unregulated voltage between 6.5 to 40 volts
- On-board backup battery saves data for about 10 years
- 0.8 Watt power consumption

Environmental
- Operating Temperature -30 to +85 °C
- Storage Temperature -40 to +85 °C

Physical
- Dimensions: 88 x 57 x 15 mm
- Weight: 48 g
- Connectors: 30 pin for digital, MMCX for antenna

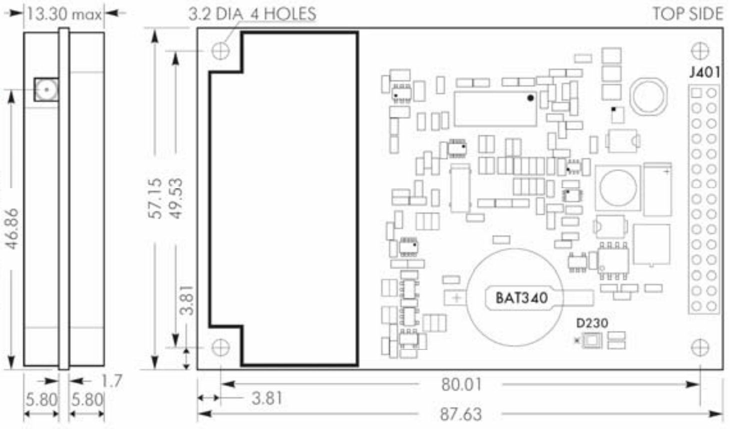
RF Connector
J001 is GPS/GLONASS antenna input connector, MMCX. The central pin of this connector is power supply for LNA, 5 VDC.

Note: LED_RED and LED_GRN are used to control the STAT LED of the MinPad. The output is a +3.3V driver in series with 100 Ohm resistor for each LED. LEDs should be with common cathode.

Pin Out:

Pin #	Signal Name	Description	I/O	Comments
1	GND	Digital Ground		
2	CTSA	Clear to Send for Serial Port A	I	
3	TXDA	Transmitted data for Serial Port A	O	
4	RTSA	Request to Send for Serial Port A	O	
5	RXDA	Received Data for Serial Port A	I	
6	NC	Not Connected		
7	GND	Digital Ground		
8	CTSB	Clear to Send for Serial Port B	I	
9	TXDB	Transmitted data for Serial Port B	O	
10	RTSB	Request to Send for Serial Port B	O	
11	RXDB	Received Data for Serial Port B	I	
12	BOOT	Boot Loader — Factory use only	I	*1
13	PWR_IN	Power Supply	PWR	*2
14	PWR_IN	Power Supply	PWR	*2
15	NC	Not Connected		
16	NC	Not Connected		
17	EXT_RESET*	External Reset Control	I	*3
18	1PPS	1 Pulse Per Second	O	*4
19	PWR_GND	Power Ground	PWR	
20	PWR_GND	Power Ground	PWR	
21	LED_RED	External LED	O	See Note
22	LED_GRN	External LED	O	See Note
23	TXDC	Transmitted data for Serial Port C	O	
24	GND	Digital Ground		
25	RXDC	Received Data for Serial Port	I	
26	GND	Digital Ground		
27	EVENT	Event Marker	I	*5
28	TXDD	Transmitted Data for Serial Port D	O	
29	NC	Not Connected		
30	RXDD	Received Data for Serial Port D	I	

*1. Must be grounded or open *2. +6.5 to +40 volt *3. Connect to ground to activate
*4 Voh > 2.0V @ 50Ohm, T=3.2mS *5 10kOhm internal pull-up to +3.3V



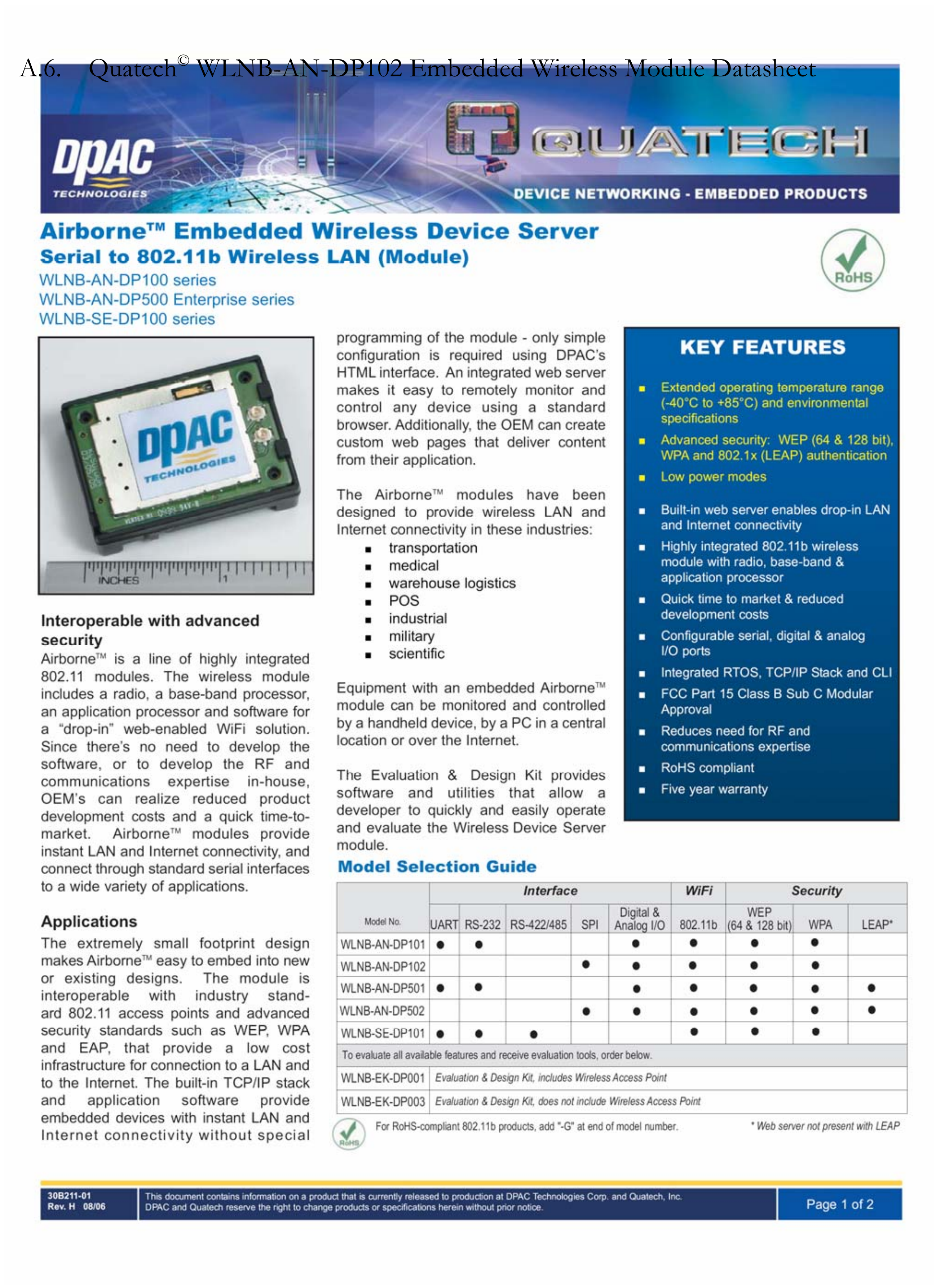
*Specifications are subject to change without notice.

The drawing shows the actual size of the board

JAVAD NAVIGATION SYSTEMS

www.javad.com

A.6. Quatech© WLNB-AN-DP102 Embedded Wireless Module Datasheet



DPAC TECHNOLOGIES

QUATECH

DEVICE NETWORKING - EMBEDDED PRODUCTS

Airborne™ Embedded Wireless Device Server

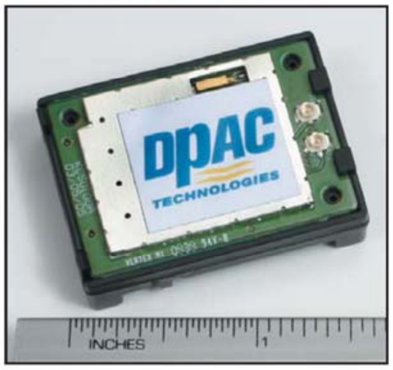
Serial to 802.11b Wireless LAN (Module)

WLNB-AN-DP100 series

WLNB-AN-DP500 Enterprise series

WLNB-SE-DP100 series

RoHS



Interoperable with advanced security

Airborne™ is a line of highly integrated 802.11 modules. The wireless module includes a radio, a base-band processor, an application processor and software for a "drop-in" web-enabled WiFi solution. Since there's no need to develop the software, or to develop the RF and communications expertise in-house, OEM's can realize reduced product development costs and a quick time-to-market. Airborne™ modules provide instant LAN and Internet connectivity, and connect through standard serial interfaces to a wide variety of applications.

Applications

The extremely small footprint design makes Airborne™ easy to embed into new or existing designs. The module is interoperable with industry standard 802.11 access points and advanced security standards such as WEP, WPA and EAP, that provide a low cost infrastructure for connection to a LAN and to the Internet. The built-in TCP/IP stack and application software provide embedded devices with instant LAN and Internet connectivity without special programming of the module - only simple configuration is required using DPAC's HTML interface. An integrated web server makes it easy to remotely monitor and control any device using a standard browser. Additionally, the OEM can create custom web pages that deliver content from their application.

The Airborne™ modules have been designed to provide wireless LAN and Internet connectivity in these industries:

- transportation
- medical
- warehouse logistics
- POS
- industrial
- military
- scientific

Equipment with an embedded Airborne™ module can be monitored and controlled by a handheld device, by a PC in a central location or over the Internet.

The Evaluation & Design Kit provides software and utilities that allow a developer to quickly and easily operate and evaluate the Wireless Device Server module.

KEY FEATURES

- Extended operating temperature range (-40°C to +85°C) and environmental specifications
- Advanced security: WEP (64 & 128 bit), WPA and 802.1x (LEAP) authentication
- Low power modes
- Built-in web server enables drop-in LAN and Internet connectivity
- Highly integrated 802.11b wireless module with radio, base-band & application processor
- Quick time to market & reduced development costs
- Configurable serial, digital & analog I/O ports
- Integrated RTOS, TCP/IP Stack and CLI
- FCC Part 15 Class B Sub C Modular Approval
- Reduces need for RF and communications expertise
- RoHS compliant
- Five year warranty

Model Selection Guide

	Interface					*WiFi*	*Security*		
Model No.	UART	RS-232	RS-422/485	SPI	Digital & Analog I/O	802.11b	WEP (64 & 128 bit)	WPA	LEAP*
WLNB-AN-DP101	●	●			●	●	●	●	
WLNB-AN-DP102				●	●	●	●	●	
WLNB-AN-DP501	●	●			●	●	●	●	●
WLNB-AN-DP502				●	●	●	●	●	●
WLNB-SE-DP101	●	●	●			●	●	●	
To evaluate all available features and receive evaluation tools, order below.									
WLNB-EK-DP001	*Evaluation & Design Kit, includes Wireless Access Point*								
WLNB-EK-DP003	*Evaluation & Design Kit, does not include Wireless Access Point*								

RoHS For RoHS-compliant 802.11b products, add "-G" at end of model number.

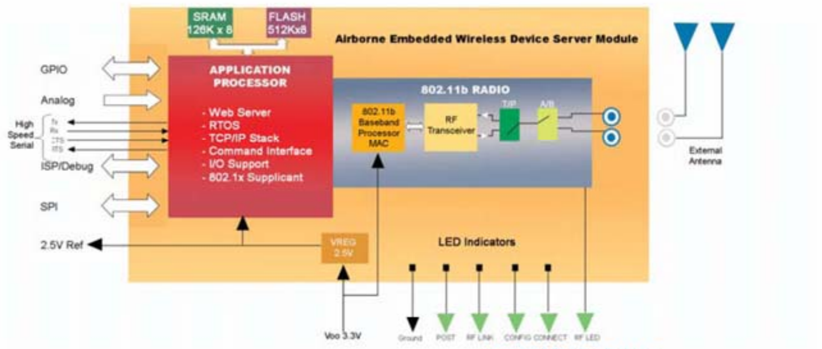
** Web server not present with LEAP*

30B211-01
Rev. H 08/06

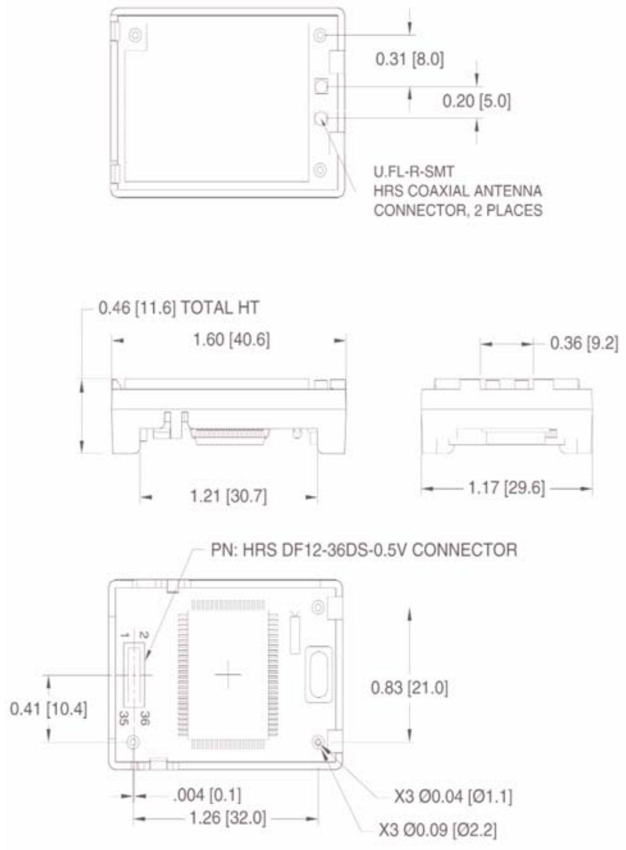
This document contains information on a product that is currently released to production at DPAC Technologies Corp. and Quatech, Inc. DPAC and Quatech reserve the right to change products or specifications herein without prior notice.

Page 1 of 2

Block Diagram



Mechanical Outline



Specifications

Technology	IEEE 802.11b DSSS, WiFi compliant
Frequency	2.4 ~ 2.4835 GHz (US/Can/Japan/Europe) 2.471 ~ 2.497 GHz (Japan)
Modulation	DQPSK, DBPSK and CCK
Channels	11 channels - USA/Canada 13 channels - Europe 14 channels - Japan 4 channels - France
Data Rate	11, 5.5, 2, 1 Mbps
MAC	CSMA/CA with ACK, RTS, CTS
Protocols Data Transfer	TCP/IP, ARP, ICMP, DHCP, DNS, HTTP UDAP Discovery TCP/IP, HTTP, UDP
RF Power	+15 dBm (typical) Approx. 32 mW
Sensitivity	-82dBm for 11Mbps -86dBm for 5.5Mbps -88dBm for 2 Mbps -90dBm for 1Mbps
Security	WEP (64 & 128 bit), WPA (PSK & TKIP), WPA with LEAP
Antenna	Supports diversity antennas, using U.FL coaxial connectors 50 ohms (on WLNB-AN-DPxxx models)
Supply	3.3 VDC
Current	420mA - transit mode (typical) 350mA - receive mode (typical) 75mA - sleep mode (typical) 15mA - 5% duty cycle*
Operating Temperature	-40°C - +85°C
GPIO	Up to 8 digital I/O ports and Status
Serial	UART up to 921.6 Kbps I^2C Master to 400KHz SPI up to 1Mb/s (Master clock up to 20MHz) Supports RS-232/422/485 (on WLNB-SE-DP101)
Analog	Up to 8 channels, 10 bit resolution
Connector	36 Pin (P/N: HRS DF 12-36DS-0.5V)
Agency Approvals	FCC Part 15 Class B Sub C Intentional Radiator Modular Approval Industry of Canada RoHS and WEEE Compliant

* Low power mode requires external circuitry.

Quatech, Inc. High Performance Device Networking and Connectivity
5675 Hudson Industrial Parkway, Hudson, OH 44236 Tel 800.553.1170 +1 330.655.9000 Fax 330.655.9070 www.quatech.com

A.7. Bel© x7AH-03H Series DC/DC Converters

NON-ISOLATED DC/DC CONVERTERS
4.5V-32V Input 1.2V-5.0V/3A Output

bel POWER PRODUCTS

x7AH-03H Series

- Non-Isolated
- High Efficiency
- High Power Density
- Excellent Thermal Performance
- Remote On/Off
- Input Under Voltage Lockout
- OCP/SCP
- Low Cost

Description

The Bel x7AH-03Hxx0 is part of the low cost non-isolated DC/DC power converter series. It is packaged in a compact, overmolded package rated at 3A. Optional lead forming provides a vertical mount product for minimal footprint or a surface mount option for a very low profile. The output is closely regulated and the efficiency of 3.3V output is typically 90% at full load. Typical features include remote on/off, input under voltage lockout, over current protection and short circuit protection.

Part Selection

Output Voltage	Input Voltage	Max. Output Current	Max. Output Power	Typical Efficiency	Part Number Surface Mount	Part Number Vertical Mount
5.0V	8.0V – 32V	3A	15W	92%	S7AH-03H500	V7AH-03H500
3.3V	4.9V – 32V	3A	10W	90%	S7AH-03H330	V7AH-03H330
2.5V	4.5V –32V	3A	7.5W	88%	S7AH-03H250	V7AH-03H250
1.8V	4.5V –32V	3A	5.4W	85%	S7AH-03H180	V7AH-03H180
1.5V	4.5V – 32V	3A	4.5W	83%	S7AH-03H150	V7AH-03H150
1.2V	4.5V – 32V	3A	3.6W	81%	S7AH-03H120	V7AH-03H120

Note: Add "0" suffix at the end of the model number to indicate "Tube Packaging", and "R" for "Reel Packaging", and "G" for "Tray Packaging".

Absolute Maximum Ratings

Parameter	Min	Typ	Max	Notes
Input Voltage (continuous)	-0.3V	-	34V	
Output Enable Terminal Voltage	-0.3V	-	12V	
Ambient Temperature	-40°C	-	85°C	
Storage Temperature	-40°C	-	125°C	

Input Specifications

Parameter	Min	Typ	Max	Notes
Input Voltage	4.5V	-	32V	See "Part Selection" for more details.
Input Current (no load)	-	30mA	-	
Input Current (full load)	-	-	3A	
Remote Off Input Current	-	4mA	-	
Input Reflected Ripple Current (pk-pk)	-	200mA	400mA	Tested with simulated source impedance of 500nH, 5Hz to 20MHz and two 100uF/50V electrolytic capacitors and a 3.3uF/50V ceramic capacitor at the input.
Input Reflected Ripple Current (RMS)	-	100mA	150mA	
I^2t Inrush Current Transient	-	$0.02A^2s$	$0.1A^2s$	
Turn on Voltage Threshold [1]	-	4.1V	4.5V	
Turn off Voltage Threshold [2]	-	3.3V	4.0V	

Notes: 1. The max Turn on Voltage threshold of the 3.3V & 5.0V output module will be relaxed to 4.9V & 8.0V respectively.
2. The max Turn off Voltage threshold of the 3.3V output module will be relaxed to 4.5V. The 5.0V output module does not have such function.

Output Specifications

Parameter			Min	Typ	Max	Notes
Output Voltage Set Point		Vo=5.0V	4.900V	5.0V	5.100V	Test conditions: Vin=12V, Io=50% full load
		Vo=3.3V	3.234V	3.3V	3.366V	
		Vo=2.5V	2.450V	2.5V	2.550V	
		Vo=1.8V	1.764V	1.8V	1.836V	
		Vo=1.5V	1.470V	1.5V	1.530V	
		Vo=1.2V	1.176V	1.2V	1.224V	
Line Regulation		Vo=5.0V	-	±10mV	±15mV	
		Vo=1.2-3.3V	-	±5mV	±10mV	
Load Regulation		Vo=5.0V	-	±10mV	±15mV	
		Vo=1.2-3.3V	-	±5mV	±10mV	
Regulation Over Temperature (-40°C to +85°C)			-	30mV	50mV	
Output Current			0A	-	3A	
Current Limit Threshold			3.3A	-	9A	
Short Circuit Surge Transient		Vo=1.2V-5.0V	-	$0.02A^2s$	$0.1A^2s$	
Ripple and Noise (RMS)		Vo=1.2V-5.0V	-	25mV	50mV	Tested with 0-20MHz BW, with a 220uF tantalum capacitor at the output.
Ripple and Noise (pk-pk)		Vo=1.2V-5.0V	-	60mV	100mV	
Turn on Time			-	15mS	50mS	
Overshoot at Turn on			-	2%	5%	
Output Capacitance			220uF	-	1200uF	
Transient Response						
50% ~ 100% Max Load	Overshoot	Vo=5.0V	-	150mV	200mV	Test conditions: di/dt = 0.5A/uS; Vin = 12V; with a 220uF Tantalum capacitor at the output.
	Settling Time		-	100uS	150uS	
100% ~ 50% Max Load	Overshoot		-	150mV	200mV	
	Settling Time		-	100uS	150uS	
50% ~ 100% Max Load	Overshoot	Vo=3.3V	-	130mV	180mV	
	Settling Time		-	100uS	150uS	
100% ~ 50% Max Load	Overshoot		-	130mV	180mV	
	Settling Time		-	100uS	150uS	
50% ~ 100% Max Load	Overshoot	Vo=1.8V - 2.5V	-	100mV	150mV	
	Settling Time		-	50uS	100uS	
100% ~ 50% Max Load	Overshoot		-	100mV	150mV	
	Settling Time		-	50uS	100uS	
50% ~ 100% Max Load	Overshoot	Vo=1.2V - 1.5V	-	90mV	140mV	
	Settling Time		-	40uS	80uS	
100% ~ 50% Max Load	Overshoot		-	90mV	140mV	
	Settling Time		-	40uS	80uS	

Note: All specifications are typical at nominal input, full load at 25°C unless otherwise stated.

General Specifications

Parameter	Min	Typ	Max	Notes
Efficiency				Measured at Vin=12V, full load and Ta=25°C
Vo=5.0V	89%	92%	-	
Vo=3.3V	87%	90%	-	
Vo=2.5V	85%	88%	-	
Vo=1.8V	82%	85%	-	
Vo=1.5V	80%	83%	-	
Vo=1.2V	78%	81%		
Switching Frequency	200KHz	300KHz	400KHz	
Output Trim Range (narrow trim)	90%Vo	-	110%Vo	
MTBF		8,120,000 hours		Calculated Per Bell Core TR-332 (Io = Nominal; Ta = 25°C)
Dimensions (surface mount)				
Inches (L × W × H)		0.78 x 0.70 x 0.32		
Millimeters (L × W × H)		19.81 x 17.78 x 8.13		
Dimensions (vertical)				
Inches (L × W × H)		0.70 x 0.308 x 0.65		
Millimeters (L × W × H)		17.78 x 7.82 x 16.51		
Weight	-	5.1g	-	

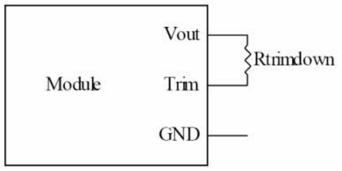
Control Specifications

Parameter	Min	Typ	Max	Notes
Remote On/Off				
Signal Low (Unit On)	-0.3V	-	1V	Remote on/off pin open, unit on.
Signal High (Unit Off)	2.8V	-	12V	

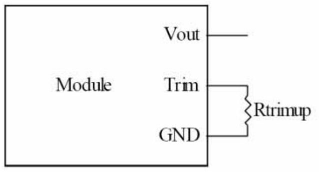
Output Trim Equations

Equations for calculating the trim resistor (in kΩ) given the desired adjusted voltage (Vadj) and the nominal output voltage of the converter (Vnom) are shown below. The Trim Down resistor should be connected between the Trim pin and Vout. The Trim Up resistor should be connected between the Trim pin and Ground. Only one of the resistors should be used for any given application.

$$R_{trimdown} = \frac{A}{V_{nom} - V_{adj}} - B$$

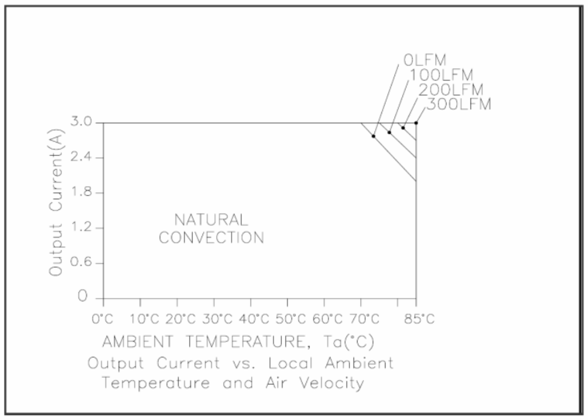


$$R_{trimup} = \frac{C}{V_{adj} - V_{nom}} - D$$



Vnom	A	B	C	D
5.0	61.850	29.400	11.760	14.700
3.3	53.840	61.700	17.200	40.200
2.5	9.556	15.620	4.496	10.000
1.8	3.849	13.830	3.064	10.000
1.5	3.102	14.420	3.536	10.000
1.2	1.794	10.910	3.536	6.490

Thermal Derating Curve

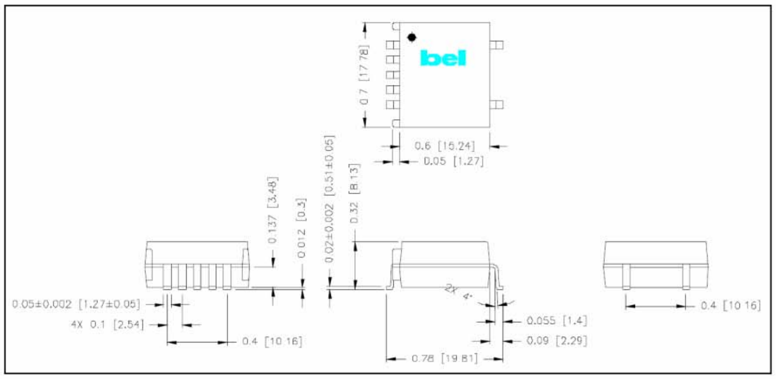


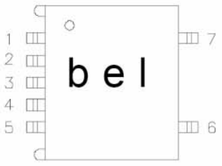
Vin=24V

NON-ISOLATED DC/DC CONVERTERS

4.5V-32V Input **1.2V-5.0V/3A Output**

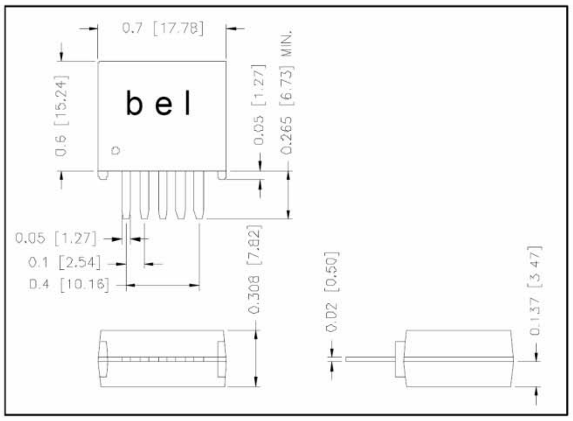
POWER PRODUCTS

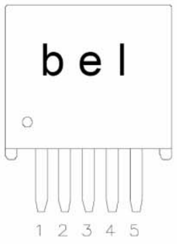




Pin Connections

Pin	Function
1	Remote On/Off (option)
2	Vin
3	Ground
4	Vout
5	Trim (option)
6	N/A
7	N/A





Pin Connections

Pin	Function
1	Remote On/Off (option)
2	Vin
3	Ground
4	Vout
5	Trim (option)

CORPORATE

Bel Fuse Inc.
206 Van Vorst Street
Jersey City, NJ 07302
Tel 201-432-0463
Fax 201-432-9542
www.belfuse.com

FAR EAST

Bel Fuse Ltd.
8F/ 8 Luk Hop Street
San Po Kong
Kowloon, Hong Kong
Tel 852-2328-5515
Fax 852-2352-3706
www.belfuse.com

EUROPE

Bel Fuse Europe Ltd.
Preston Technology Management Centre
Marsh Lane, Suite G7, Preston
Lancashire, PR1 8UD, U.K.
Tel 44-1772-556601
Fax 44-1772-888366
www.belfuse.com

Appendix B: MCETS Sensor Board Pin Connections

B.1. Miscellaneous Header and Connector Pin Connections

Pin	Name	Connected To	Pin	Name
1	P1.0/TACLK	CON10	9	3.3V_CTL
2	P1.1/TA0	CON10	10	5.0V_CTL
3	P1.2/TA1	Reserved for Battery Supply Control	-	-
4	P1.3/TA2	Temperature Sensor	6	SD
5	P1.4/SMCLK	Reserved for Pressure Sensor Control	-	-
6	P1.5/TAO	MUX 1, 2 & 3	5	EN
7	P1.6/TA1	MUX 1, 2 & 3	1	A0
8	P1.7/TA2	MUX 1, 2 & 3	10	A1
9	3.3V_1	3.3 V LC Supply Rail 1.0 µF Capacitor to PGND	-	-
10	GND	DGND	-	-
11	P2.0/ACLK	MOSFET	1	GATE
12	P2.1/TACLK	Bus Switch 2	6	2A
13	P2.2/CAOUT	Bus Switch 2 Red LED to 680 Ω Resistor to DGND	11	3A
14	P2.3/CA0	Bus Switch 2 Red LED to 680 Ω Resistor to DGND	14	4A
15	P2.4/CA1	Bus Switch 1	14	4A
16	P2.5/ROSC	Bus Switch 2	3	1A
17	P2.6/ADCLK	Bus Switch 1 Bus Switch 2	2 2	OE1 OE1
18	P2.7/TA0	NC	-	-
19	3.3v_2	3.3 V LC Supply Rail 1.0 µF Capacitor to PGND	-	-
20	GND	DGND	-	-
21	P3.0/STE0	NC	-	-
22	P3.1/SIMO0	NC	-	-
23	P3.2/SOMI0	NC	-	-
24	P3.3/ULCK0	NC	-	-
25	P3.4/UTXD0	NC	-	-
26	P3.5/URXT0	NC	-	-
27	P3.6/UTXD1	NC	-	-
28	P3.7/URXD1	NC	-	-
29	3.3v_3	3.3 V LC Supply Rail 1.0 µF Capacitor to PGND	-	-
30	GND	DGND	-	-
31	P4.0/TB0	NC	-	-
32	P4.1/TB1	NC	-	-
33	P4.2/TB2	NC	-	-
34	P4.3/TB3	NC	-	-
35	P4.4/TB4	NC	-	-
36	P4.5/TB5	NC	-	-
37	P4.6/TB6	NC	-	-

Pin	Name	Connected To	Pin	Name
38	P4.7/TBCLK	NC	-	-
39	3.3v_4	3.3 V LC Supply Rail 1.0 µF Capacitor to PGND	-	-
40	GND	DGND	-	-
41	P5.0/STE1	NC	-	-
42	P5.1/SIMO1	Bus Switch 1	6	2A
43	P5.2/SOMI1	Bus Switch 1	11	3A
44	P5.3/UCLK1	Bus Switch 1	3	1A
45	P5.4/MCLK	NC	-	-
46	P5.5/SMCLK	NC	-	-
47	P5.6/ACLK	NC	-	-
48	P5.7/TH	NC	-	-
49	3.3v_5	3.3 V LC Supply Rail 1.0 µF Capacitor to PGND	-	-
50	GND	DGND	-	-
51	P6.0/A0	NC	-	-
52	P6.1/A1	NC	-	-
53	P6.2/A2	NC	-	-
54	P6.3/A3	Op-Amp	8	3OUT
55	P6.4/A4	Op-Amp	7	2OUT
56	P6.5/A5	Op-Amp	1	1OUT
57	P6.6/A6	Op-Amp	14	4OUT
58	P6.7/A7	Temperature Sensor	4	VOUT
59	v3.3_6	3.3 V LC Supply Rail 1.0 µF Capacitor to PGND	-	-
60	GND	DGND	-	-

Table B.1: 60-Pin Microcontroller Header Pin Connections

Pin	**Name**	**Connected To**	**Pin**	**Name**
1	VBATT	Battery Voltage	-	-
2	PGND	Battery Ground	-	-
3	5.0V	5.0 V Supply Rail	-	-
4	DGND	Digital Ground	-	-
5	3.3V_HC	3.3 V HC Supply Rail	-	-
6	AGND	Analog Ground	-	-
7	3.3V_LC	3.3 V LC Supply Rail	-	-
8	PGND	Battery Ground	-	-
9	3.3V_CTL	Microcontroller Header	1	P1.0/TACLK
10	5.0V_CTL	Microcontroller Header	2	P1.1/TA0

Table B.2: 10-Pin Power and Ground Header Pin Connections

Pin	**Name**	**Connected To**	**Pin**	**Name**
1	RS232_RX_2	GPS Receiver	3	TXDA
2	RS232_TX_3	GPS Receiver	5	RXDA

Table B.3: 2-Pin RS-232 Header Pin Connections

Pin	Name	Connected To	Pin	Name
1	GND	DGND	-	-
2	CTSA	DGND	-	-
3	TXDA	CON2	1	RS232_TX_3
4	RTSA	DGND	-	-
5	RXDA	CON2	2	RS232_RX_5
6	NC	NC	-	-
7	GND	DGND	-	-
8	CTSB	NC	-	-
9	TXDB	NC	-	-
10	RTSB	NC	-	-
11	RXDB	NC	-	-
12	BOOT	DGND	-	-
13	PWR_IN	VBATT	-	-
14	PWR_IN	VBATT	-	-
15	NC	NC	-	-
16	NC	NC	-	-
17	EXT_RESET	Push Button to DGND	-	-
18	1PPS	1 MΩ Resistor to DGND	-	-
19	PWR_GND	PGND	-	-
20	PWR_GND	PGND	-	-
21	LED_RED	NC	-	-
22	LED_GRN	NC	-	-
23	TXDC	NC	-	-
24	GND	DGND	-	-
25	RXDC	NC	-	-
26	GND	DGND	-	-
27	EVENT	NC	-	-
28	TXDD	NC	-	-
29	NC	NC	-	
30	RXDD	NC	-	

Table B.4: 30-Pin GPS Header Pin Connections

B.2. Temperature Sensor Pin Connections

Pin	Name	Connected To	Pin	Name
1	VDD	3.3 V LC Supply Rail 1.0 μF Capacitor to PGND	-	-
2	TO	NC	-	-
3	TO'	NC	-	-
4	VOUT	Microcontroller Header	58	P6.7/A7
5	VTH	AGND	-	-
6	SD	Microcontroller Header	4	P1.3/TA2
7	CTG	AGND	-	-
8	GND	AGND	-	-

Table B.5: Temperature Sensor Pin Connections

B.3. Pressure Sensor Pin Connections

Pin	Name	Connected To	Pin	Name
1	-	NC	-	-
2	VS	5.0 V Supply Rail 1.0 µF Capacitor to PGND	-	-
3	AGND	AGND	-	-
4	VOUT	Op-Amp Attenuator 470 pF Capacitor to AGND	12	4IN+
5	-	NC	-	-
6	-	NC	-	-
7	-	NC	-	-
8	-	NC	-	-

Table B.6: Pressure Sensor Pin Connections

B.4. Tri-Axial Analog Inertial Sensor Pin Connections

Pin	Name	Connected To	Pin	Name
1	XREF	1 MΩ Resistor to AGND	-	-
2	XRATE	MUX 1	9	S2
3	ZREF	1 MΩ Resistor to AGND	-	-
4	ZRATE	MUX 3	9	S2
5	TEMPZ	MUX 3	7	S4
6	AGND	AGND	-	-
7	TEMPX	MUX 1	7	S4
8	TEMPY	MUX 2	7	S4
9	XMAG	MUX 1 2.2 µF Capacitor to AGND	4	S3
10	YMAG	MUX 2 2.2 µF Capacitor to AGND	4	S3
11	ZMAG	MUX 3 2.2 µF Capacitor to AGND	4	S3
12 to 22	-	NC	-	-
23	MGND	PGND	-	-
24	MAG_RESET	MOSFET	3	DRAIN
25	MGND	PGND	-	-
26	VDDM	5.0 V Supply Rail 1.0 µF Capacitor to PGND	-	-
27 to 35	-	NC	-	-
36	AGND	AGND	-	-
37	VDDA	5.0V Supply Rail 1.0 µF Capacitor to PGND	-	-
38	TESTN	AGND	-	-
39	TESTP	AGND	-	-
40	YACCEL	MUX 2	2	S1
41	ZACCEL	MUX 3	2	S1
42	XACCEL	MUX 1	2	S1
43	YREF	1 MΩ Resistor to AGND	-	-
44	YRATE	MUX 2	9	S2

Table B.7: Tri-Axial Analog Inertial Sensor Pin Connections

Pin	Name	Connected To	Pin	Name
1	GATE	Microcontroller Header	11	P2.0/ACLK
2	SOURCE	DGND	-	-
3	DRAIN	MAG10	24	MAG_RESET
		30 kΩ Resistor to 5.0 V Supply Rail	-	-

Table B.8: MOSFET Reset Circuit Pin Connections

B.5. Analog Multiplexer Pin Connections

Pin	Name	Connected To	Pin	Name
1	A0	Microcontroller Header	7	P1.6/TA1
2	S1	MAG10	42	XACCEL
3	GND	DGND	-	-
4	S3	MAG10	9	XMAG
5	EN	Microcontroller Header	6	P1.5/TA0
6	VDD	5.0 V Supply Rail 1.0 µF Capacitor to PGND	-	-
7	S4	MAG10	7	TEMPX
8	D	Op-Amp Attenuator	3	1IN+
9	S2	MAG10	2	XRATE
10	A1	Microcontroller Header	8	P1.7/TA2

Table B.9: Analog Multiplexer 1 Pin Connections

Pin	Name	Connected To	Pin	Name
1	A0	Microcontroller Header	7	P1.6/TA1
2	S1	MAG10	40	YACCEL
3	GND	DGND	-	-
4	S3	MAG10	10	YMAG
5	EN	Microcontroller Header	6	P1.5/TA0
6	VDD	5.0 V Supply Rail 1.0 µF Capacitor to PGND	-	-
7	S4	MAG10	8	TEMPY
8	D	Op-Amp Attenuator	5	2IN+
9	S2	MAG10	44	YRATE
10	A1	Microcontroller Header	8	P1.7/TA2

Table B.10: Analog Multiplexer 2 Pin Connections

Pin	Name	Connected To	Pin	Name
1	A0	Microcontroller Header	7	P1.6/TA1
2	S1	MAG10	41	ZACCEL
3	GND	DGND	-	-
4	S3	MAG10	11	ZMAG
5	EN	Microcontroller Header	6	P1.5/TA0
6	VDD	5.0 V Supply Rail 1.0 µF Capacitor to PGND	-	-
7	S4	MAG10	5	TEMPZ
8	D	Op-Amp	10	3IN+
9	S2	MAG10	4	ZRATE
10	A1	Microcontroller Header	8	P1.7/TA2

Table B.11: Analog Multiplexer 3 Pin Connections

B.6. Operational Amplifier/Voltage Attenuator Pin Connections

Pin	Name	Connected To	Pin	Name
1	1OUT	Voltage Divider to Microcontroller Header	56	P6.5/A5 (56)
2	1IN-	Op-Amp	1	1OUT (1)
3	1IN+	MUX1	8	D (8)
4	VCC+	5.0 V Supply Rail 1.0 µF Capacitor to PGND	-	-
5	2IN+	MUX2	8	D (8)
6	2IN-	Op-Amp	7	2OUT (7)
7	2OUT	Voltage Divider to Microcontroller Header	55	P6.4/A4 (55)
8	3OUT	Voltage Divider to Microcontroller Header	54	P6.3/A3 (54)
9	3IN-	Op-Amp	8	3OUT (8)
10	3IN+	MUX3	8	D (8)
11	VCC-	AGND	-	-
12	4IN+	Pressure Sensor	4	VOUT (4)
13	4IN-	Op-Amp	14	4OUT (14)
14	4OUT	Voltage Divider to Microcontroller Header	57	P6.6/A6 (57)

Table B.12: Op-Amp Attenuator Pin Connections

B.7. Embedded Wireless Module Pin Connections

Pin	Name	Connected To	Pin	Name
1	GND	DGND	-	-
2	TSI	NC	-	-
3	DVDD	3.3 V HC Supply Rail 1.0 µF Capacitor to PGND	-	-
4	DVDD	3.3 V HC Supply Rail 1.0µF Capacitor to PGND	-	-
5	V2.5	30 kΩ Resistor to Wireless Module	11	G3/FACRES
6	RFU	NC	-	-
7	/RESET	3.3V HC Supply Rail	-	-
8	/TSS	NC	-	-
9	G6	1 MΩ Resistor to DGND	-	-
10	TSO	NC	-	-
11	G3/FACRES	30 kΩ Resistor to Wireless Module Button to DGND	5	V2.5
12	F5/SS	Bus Switch 1	11	4B
13	G5	1 MΩ Resistor to DGND	-	-
14	G4	1 MΩ Resistor to DGND	-	-
15	VSS	DGND	-	-
16	VSS	DGND	-	-
17	G2	1 MΩ Resistor to DGND	-	-
18	F4/SCLK	Bus Switch 1	4	1B
19	G1	1 MΩ Resistor to DGND	-	-
20	TSCK	NC	-	-
21	G7	1 MΩ Resistor to DGND	-	-
22	G0/INT	Bus Switch 2	4	1B
23	F6/CONNECT	Bus Switch 2	13	4B
24	F7/SDI	Bus Switch 1	7	2B
25	F0/POST	Bus Switch 2	7	2B
26	F3/WLAN_STAT	1 MΩ Resistor to DGND	-	-
27	F2/LINK	Bus Switch 2	10	3B
28	F1/SDO	Bus Switch 1	10	3B
29	E6	1 MΩ Resistor to DGND	-	-
30	E5	1 MΩ Resistor to DGND	-	-
31	E7	1 MΩ Resistor to DGND	-	-
32	E4	1 MΩ Resistor to DGND	-	-
33	DVDD	3.3 V HC Supply Rail 1.0µF Capacitor to PGND	-	-
34	DVDD	3.3 V HC Supply Rail 1.0µF Capacitor to PGND	-	-
35	/RF_LED	680Ω Resistor to Red LED to 3.3 V LC Supply Rail	-	-
36	VSS	DGND	-	-

Table B.13: Embedded Wireless Module Pin Connections

B.8. Bus Switch Pin Connections

Pin	Name	Connected To	Pin	Name
1	NC	NC	-	-
2	OE1	Microcontroller Header	17	P2.6/ADCLK
3	1A	Microcontroller Header	44	P5.3/UCLK1
4	1B	Wireless Module	18	F4/SCLK
5	OE2	Microcontroller Header	17	P2.6/ADCLK
6	2A	Microcontroller Header	42	P5.1/SIMO
7	2B	Wireless Module	24	F7/SDI
8	GND	DGND	-	-
9	NC	NC	-	-
10	3B	Wireless Module	28	F1/SDO
11	3A	Microcontroller Header	43	P5.2/SOMI
12	OE3	Microcontroller Header	17	P2.6/ADCLK
13	4B	Wireless Module	12	F5/SS
14	4A	Microcontroller Header	15	P2.4/CA1
15	OE4	Microcontroller Header	17	P2.6/ADCLK
16	VCC	5.0 V Supply Rail 1.0 µF CAP to PGND	-	-

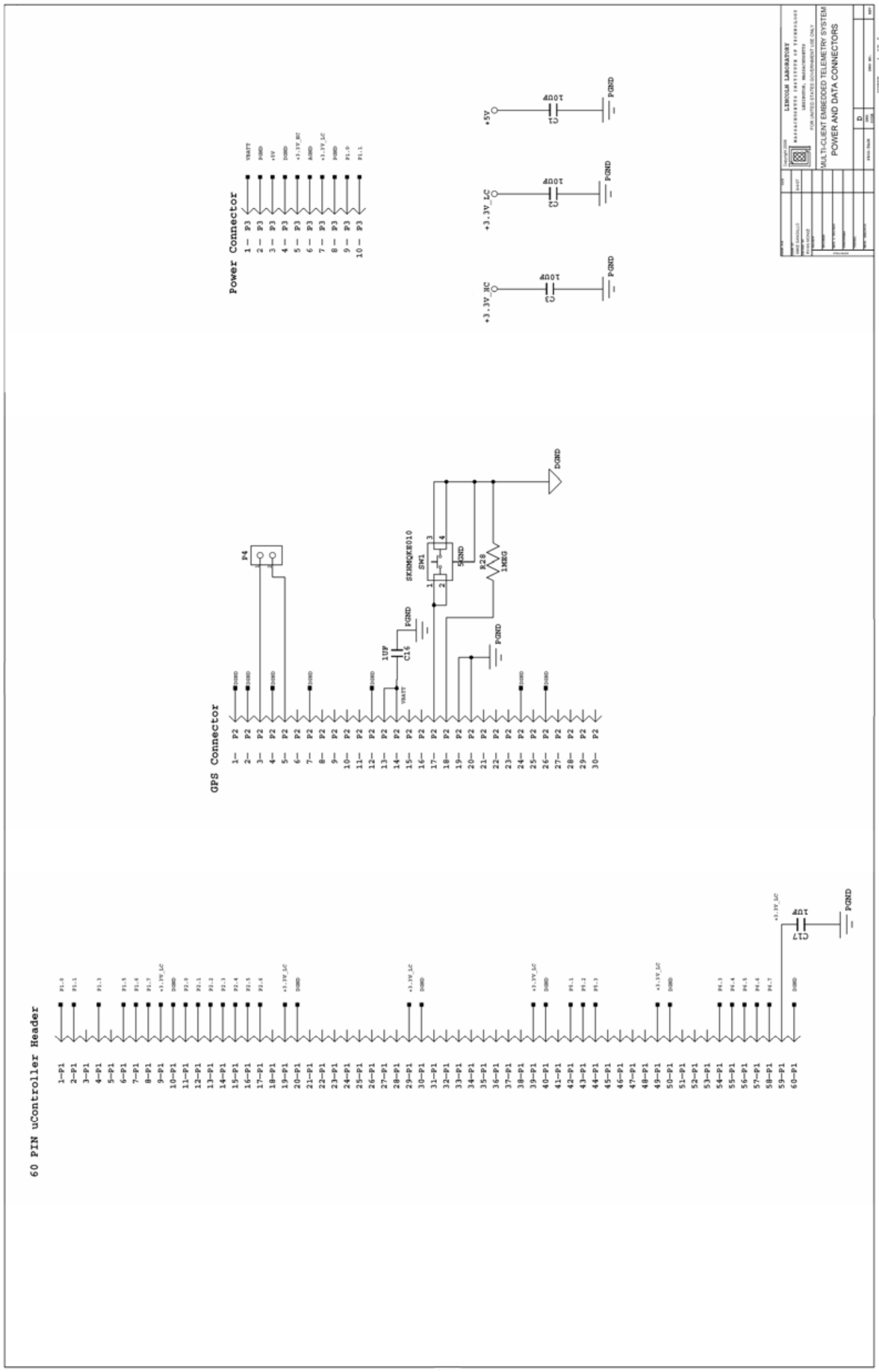
Table B.14: Bus Switch 1 Pin Connections

Pin	Name	Connected To	Pin	Name
1	NC	NC	-	-
2	OE1	Microcontroller Header	17	P2.6/ADCLK
3	1A	Microcontroller Header	16	P2.5/ROSC
4	1B	Wireless Module	22	G0/INT
5	OE2	Microcontroller Header	17	P2.6/ADCLK
6	2A	Microcontroller Header	12	P2.1/TACLK
7	2B	Wireless Module	25	F0/POST
8	GND	DGND	-	-
9	NC	NC	-	-
10	3B	Wireless Module	27	F2/RF_LINK
11	3A	Microcontroller Header	13	P2.2/CAOUT
12	OE3	Microcontroller Header	17	P2.6/ADCLK
13	4B	Wireless Module	23	F6/CONNECT
14	4A	Microcontroller Header	14	P2.3/CA0
15	OE4	Microcontroller Header	17	P2.6/ADCLK
16	VCC	5.0 V Supply Rail 1.0 µF CAP to PGND	-	-

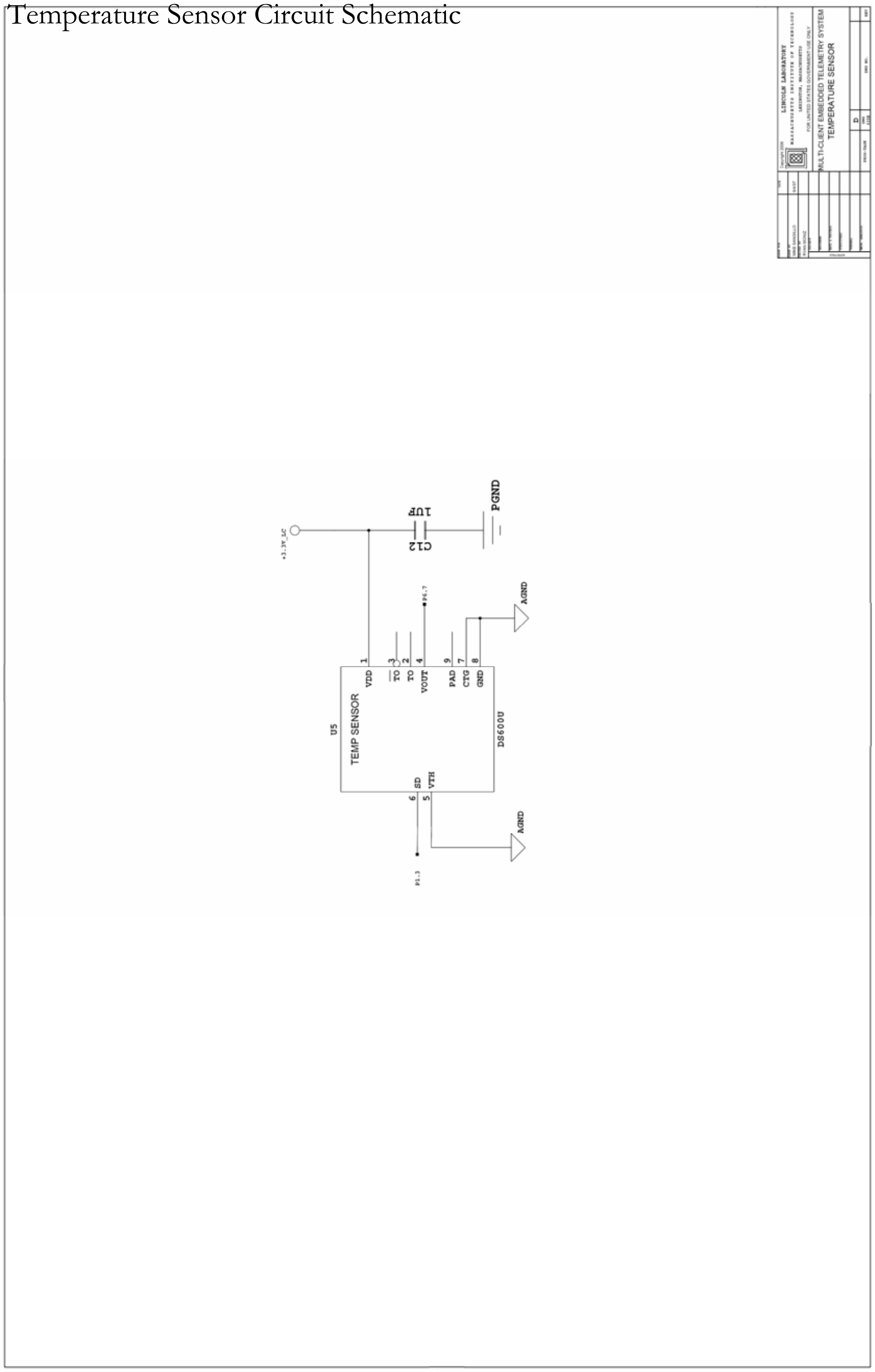
Table B.15: Bus Switch 2 Pin Connections

Appendix C: Circuit Schematics

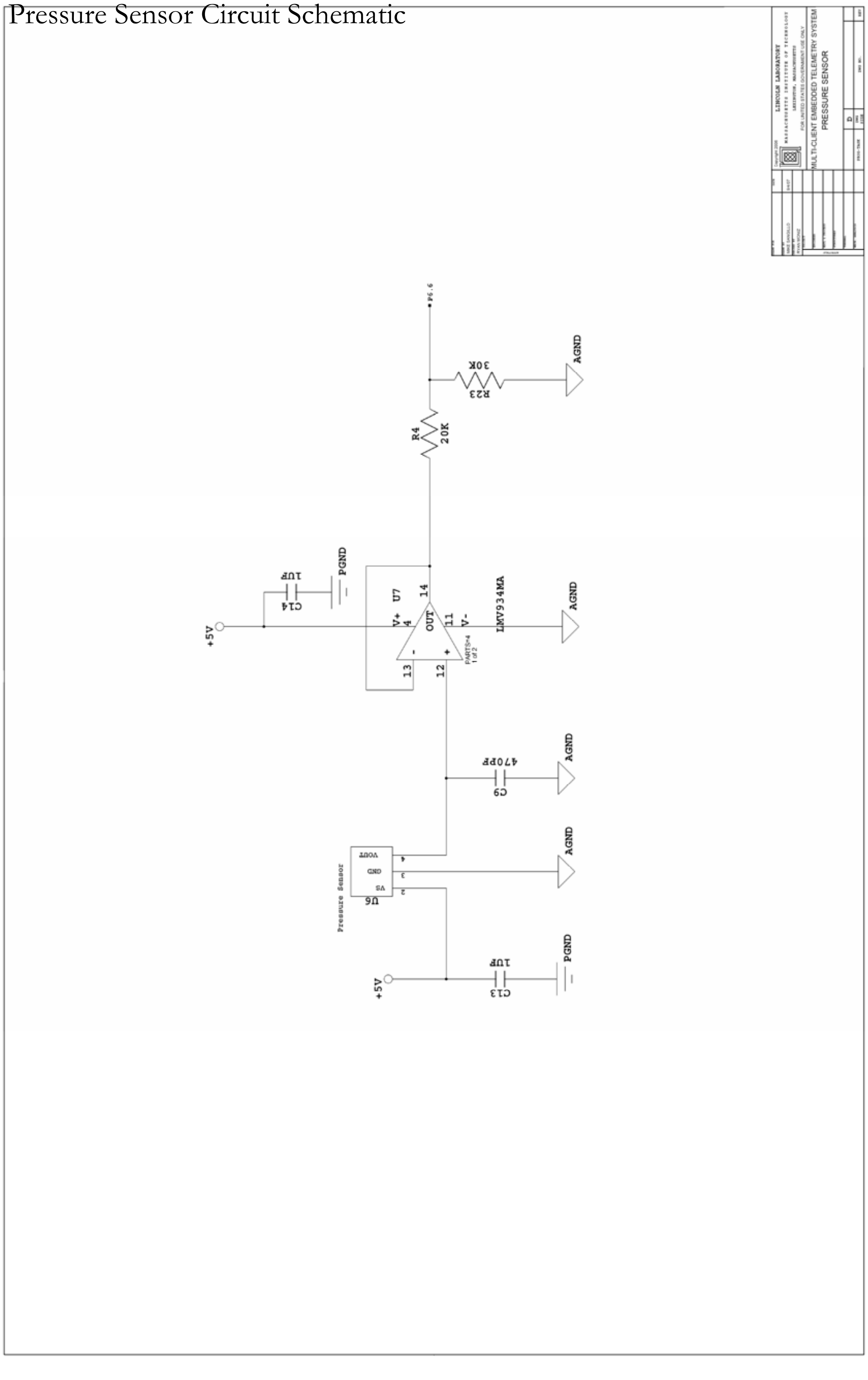
C.1. Miscellaneous Header and Connector Circuit Schematics



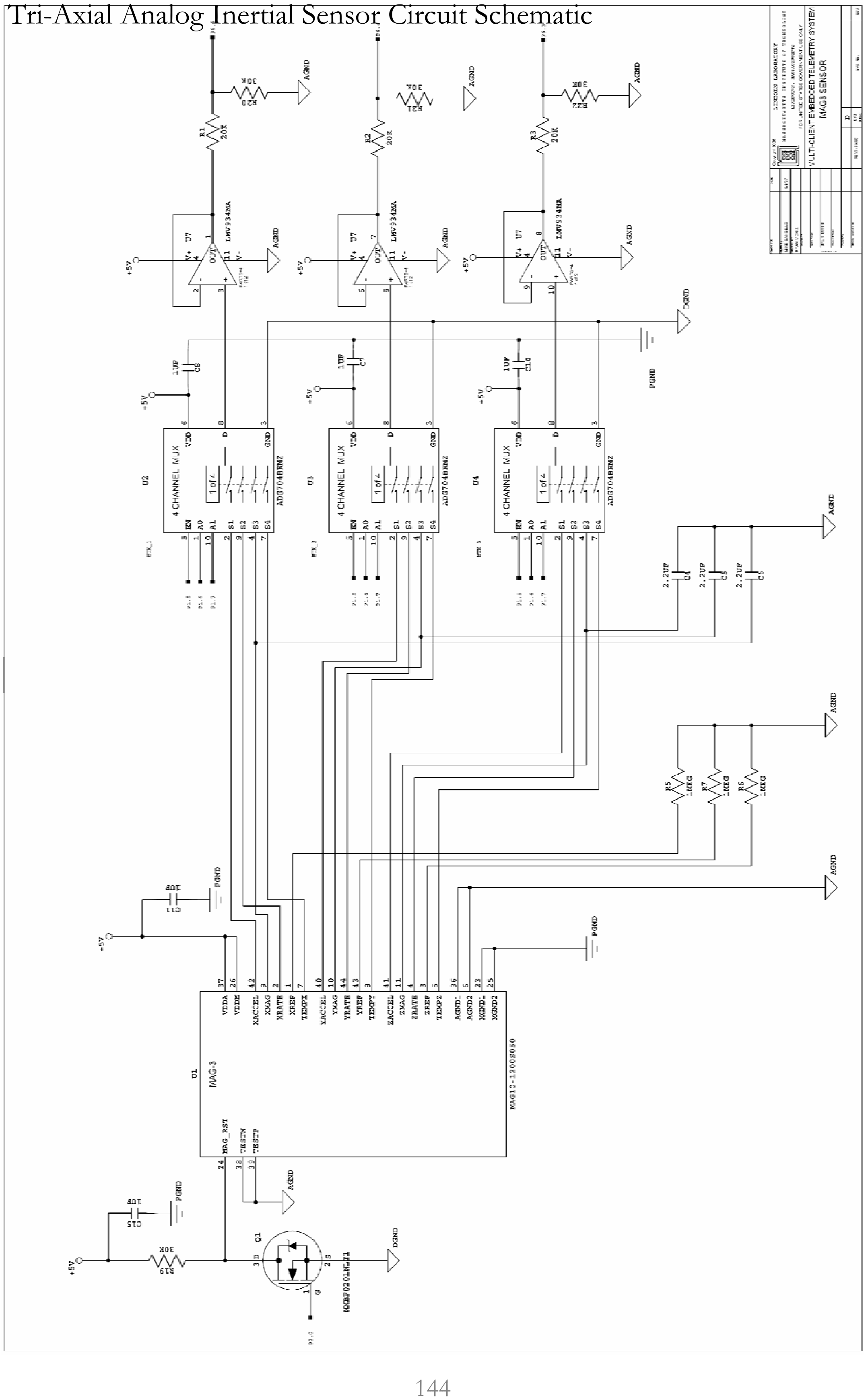
C.2. Temperature Sensor Circuit Schematic



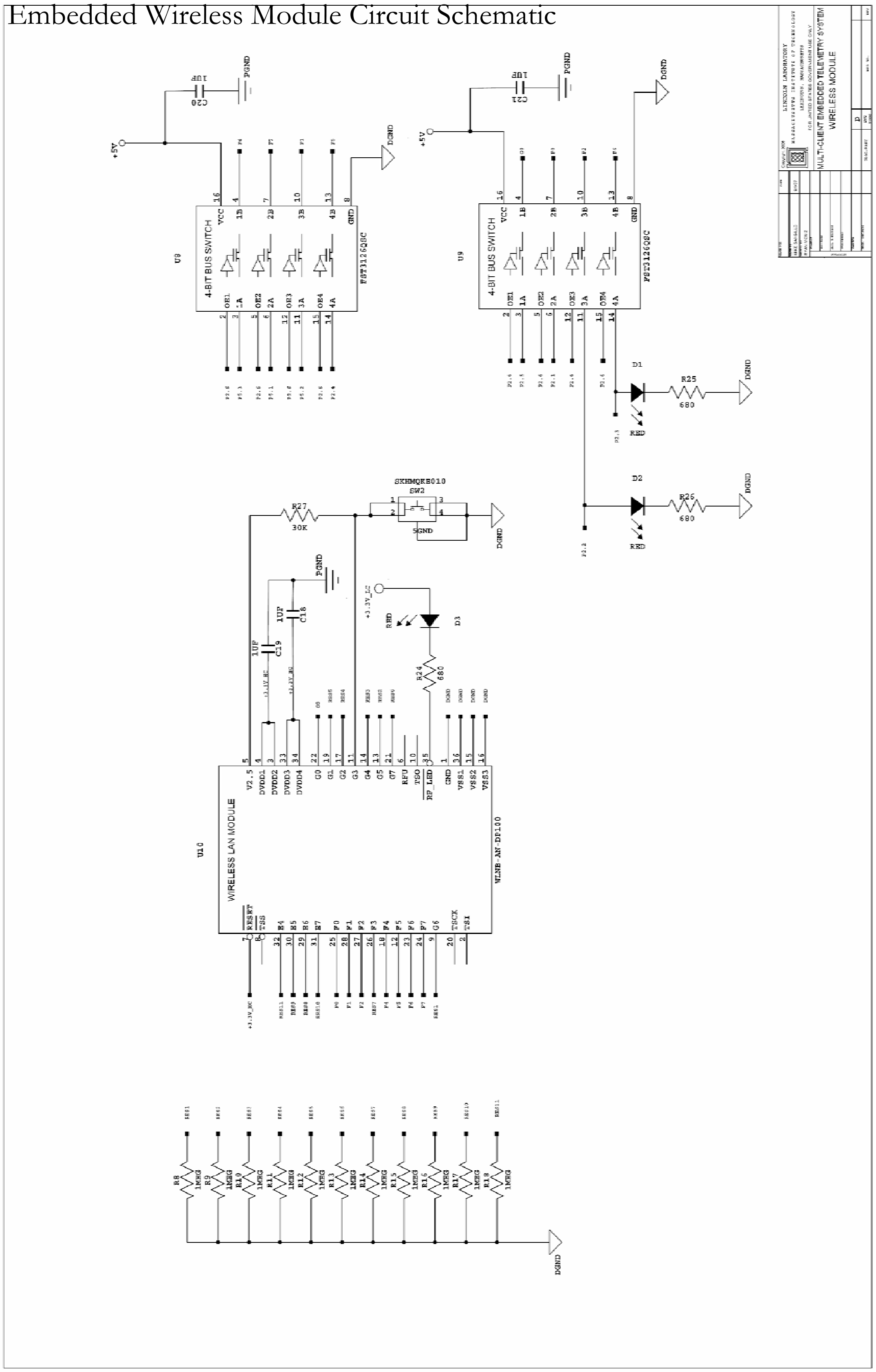
C.3. Pressure Sensor Circuit Schematic



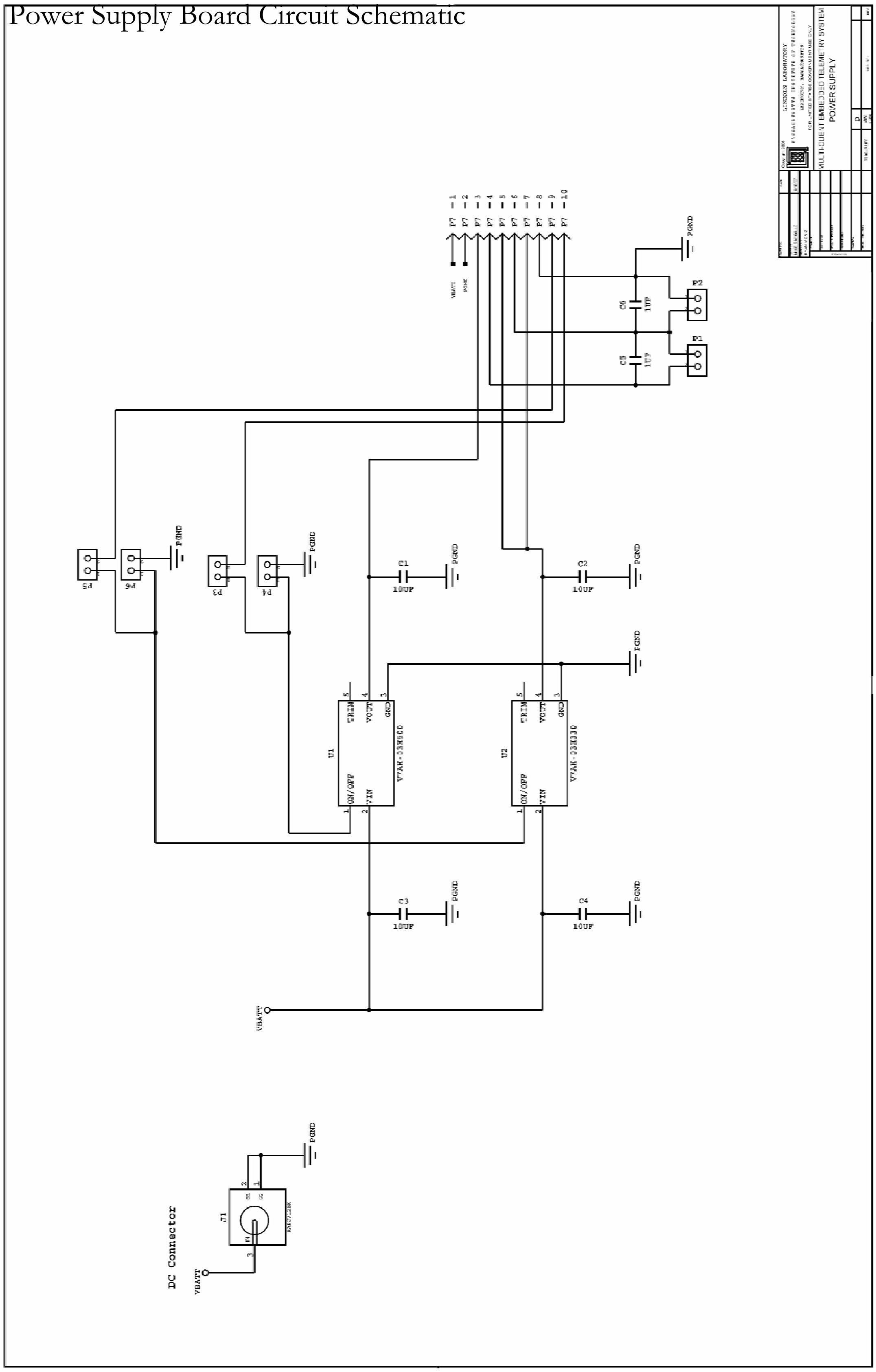
C.4. Tri-Axial Analog Inertial Sensor Circuit Schematic



C.5. Embedded Wireless Module Circuit Schematic



C.6. Power Supply Board Circuit Schematic



Appendix D: Printed Circuit Board Layouts

D.1. Sensor Board Printed Circuit Board Layout

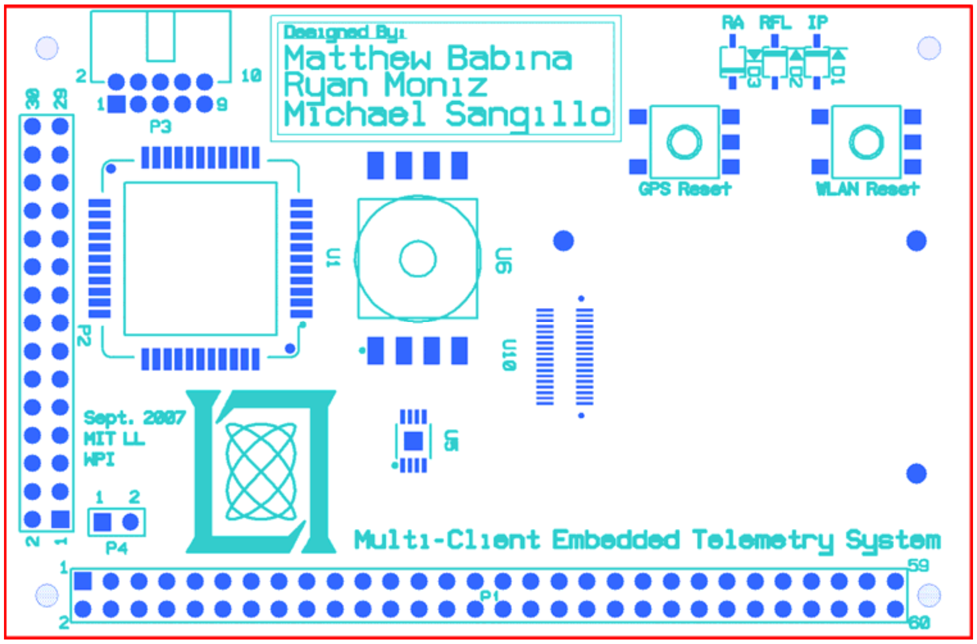


Figure D.1: Top Sensor Board Silk Screen

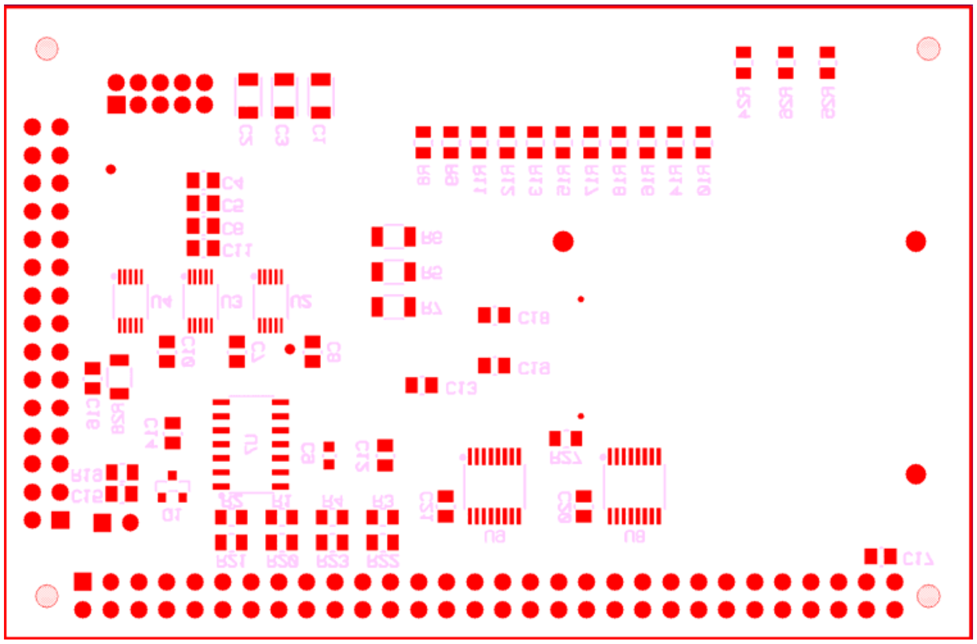
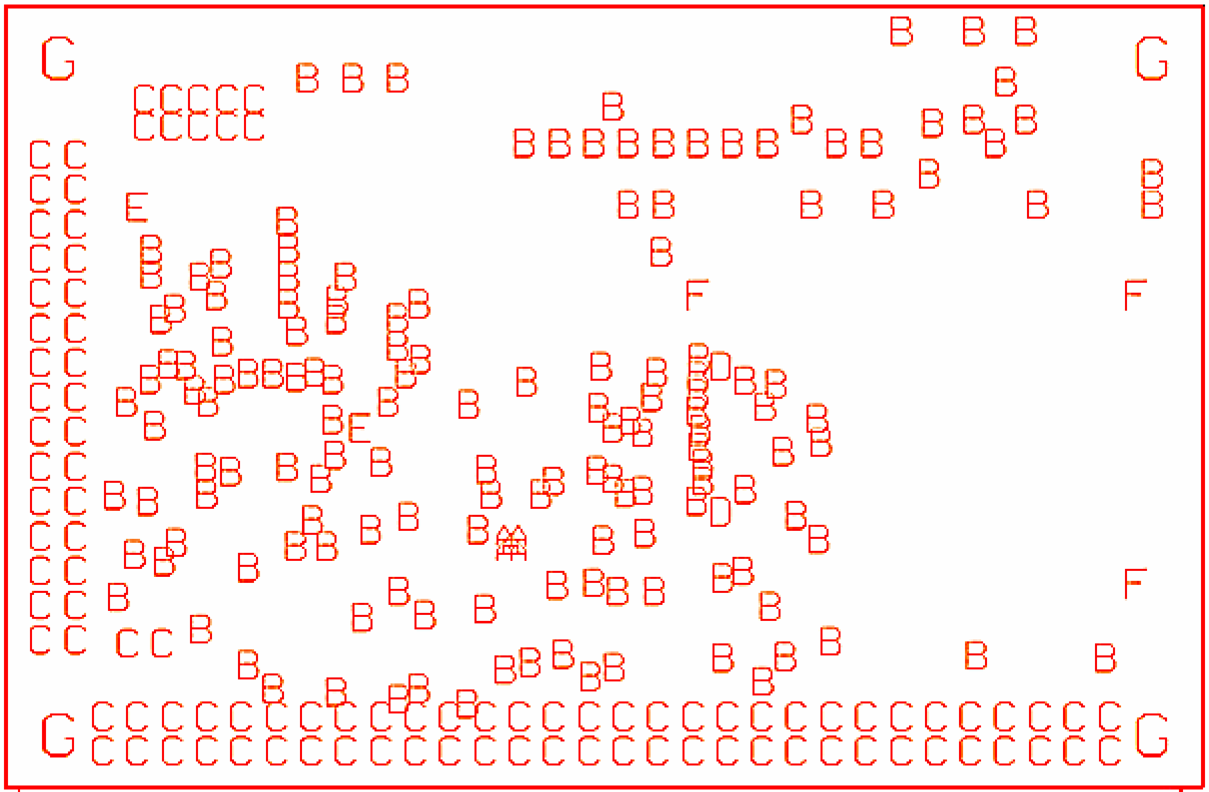


Figure D.2: Bottom Sensor Board Silk Screen



Through Holes All Drills (unless specified) +/- 0.003 (in)				
Symbol	Diameter (in)	Tolerance (in)	Plated	Quantity
A	0.013	+/- 0.003	Yes	5
B	0.016		Yes	154
C	0.035	+/- 0.003	Yes	102
D	0.024	+0.003 / -0.000	No	2
E	0.046	+/- 0.003	No	2
F	0.087		No	3
G	0.125	+/- 0.004	No	4

Figure D.3: Sensor Board Drill Holes

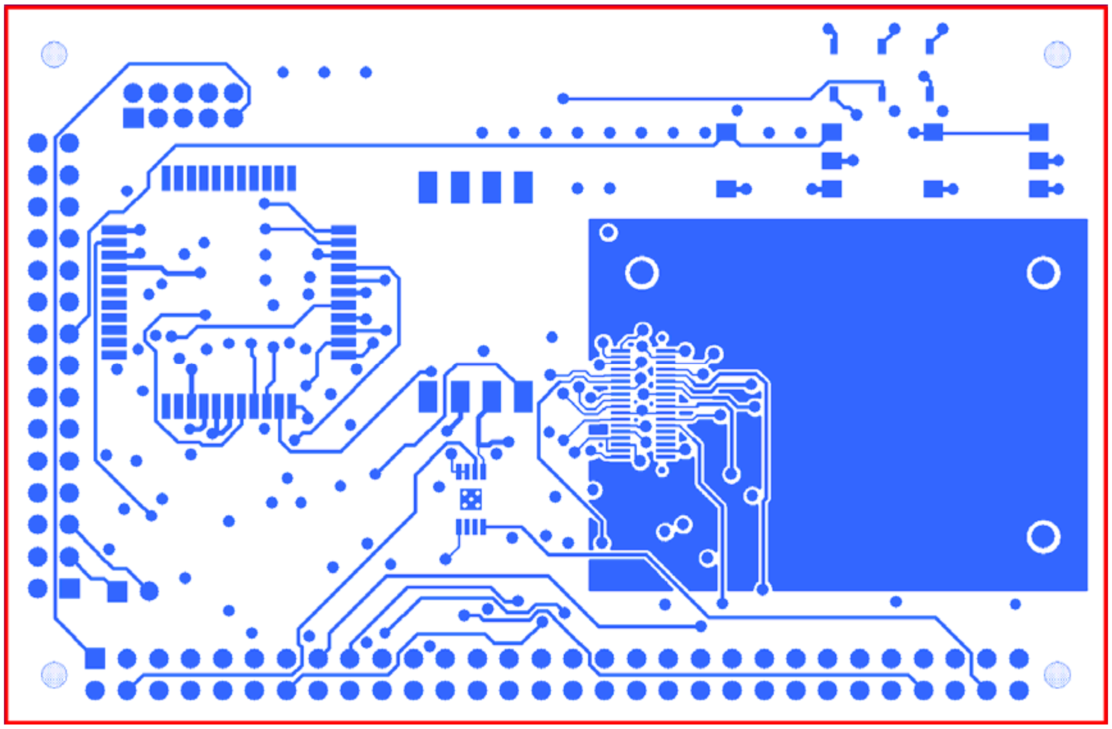


Figure D.4: First Sensor PCB Layer

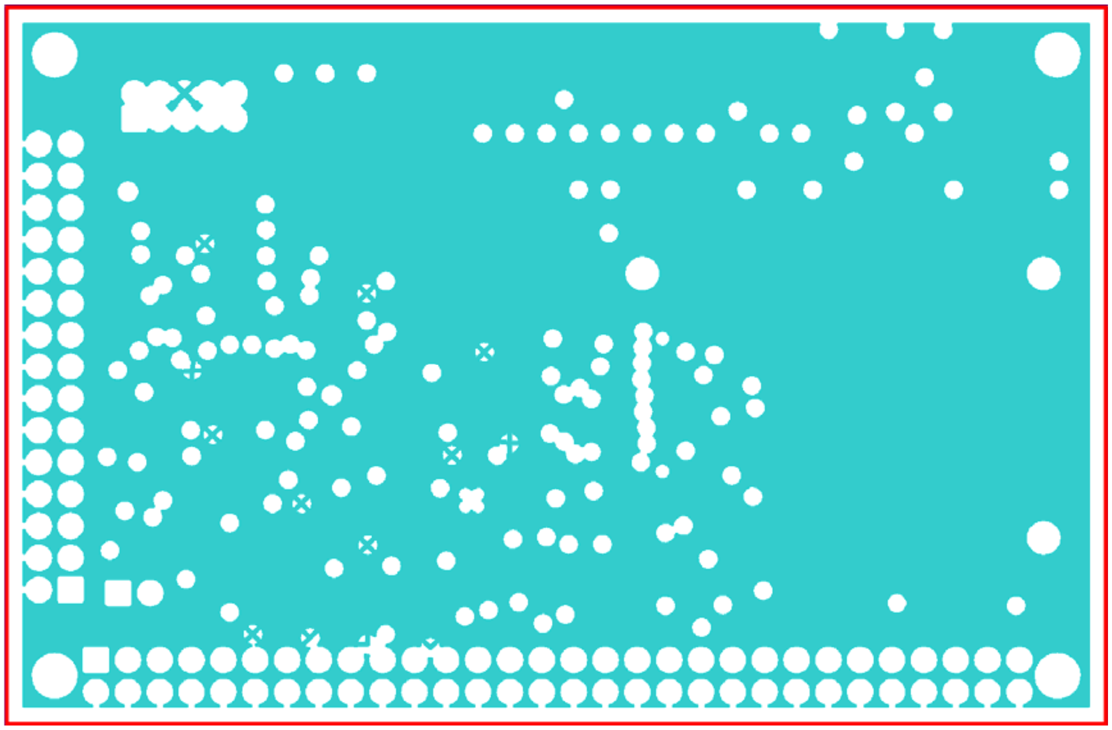


Figure D.5: Second Sensor PCB Layer (Analog Ground Plane)

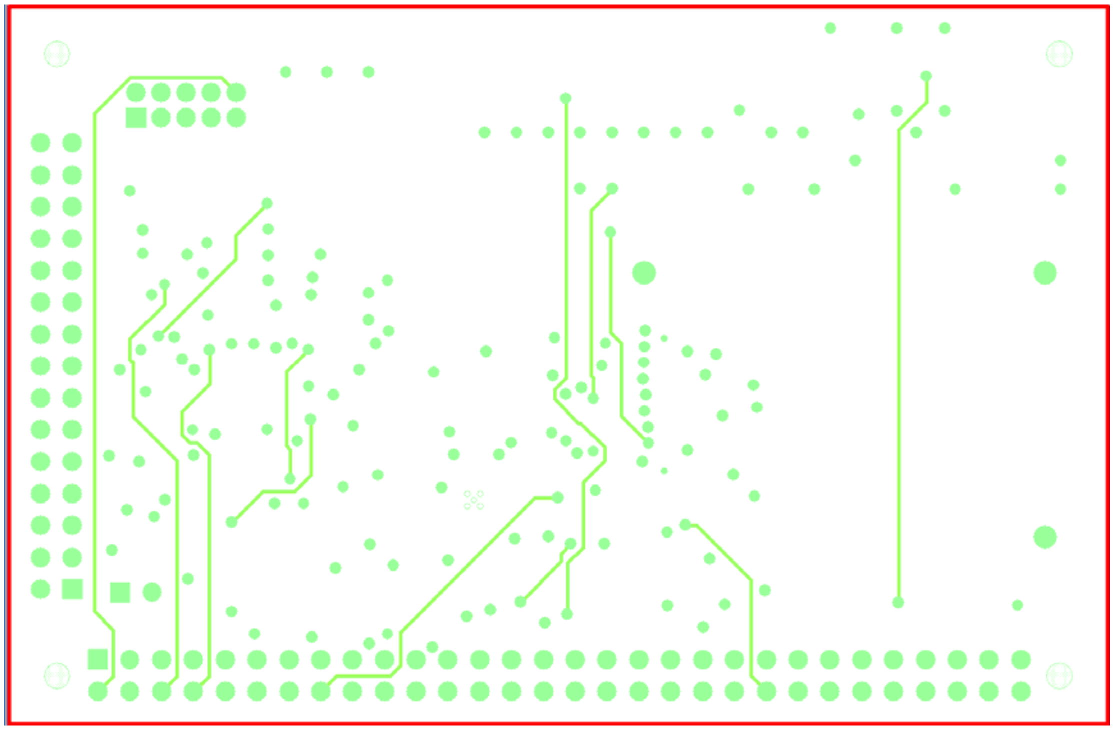


Figure D.6: Third Sensor PCB Layer

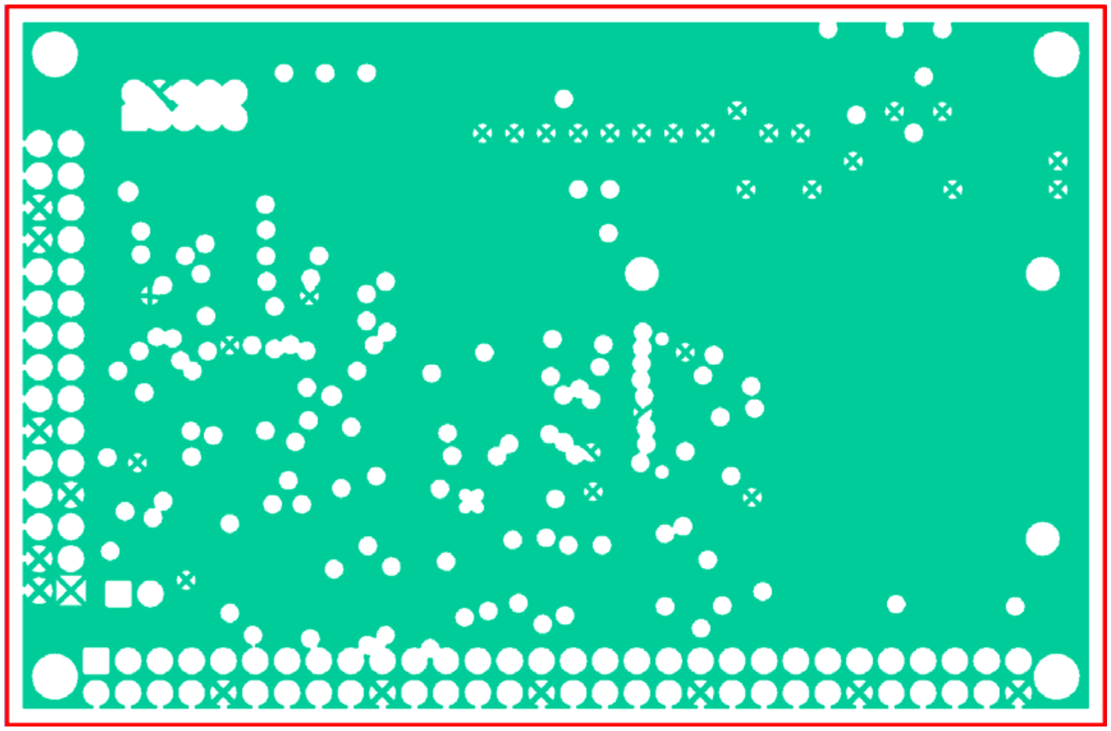


Figure D.7: Fourth Sensor PCB Layer (Digital Ground Plane)

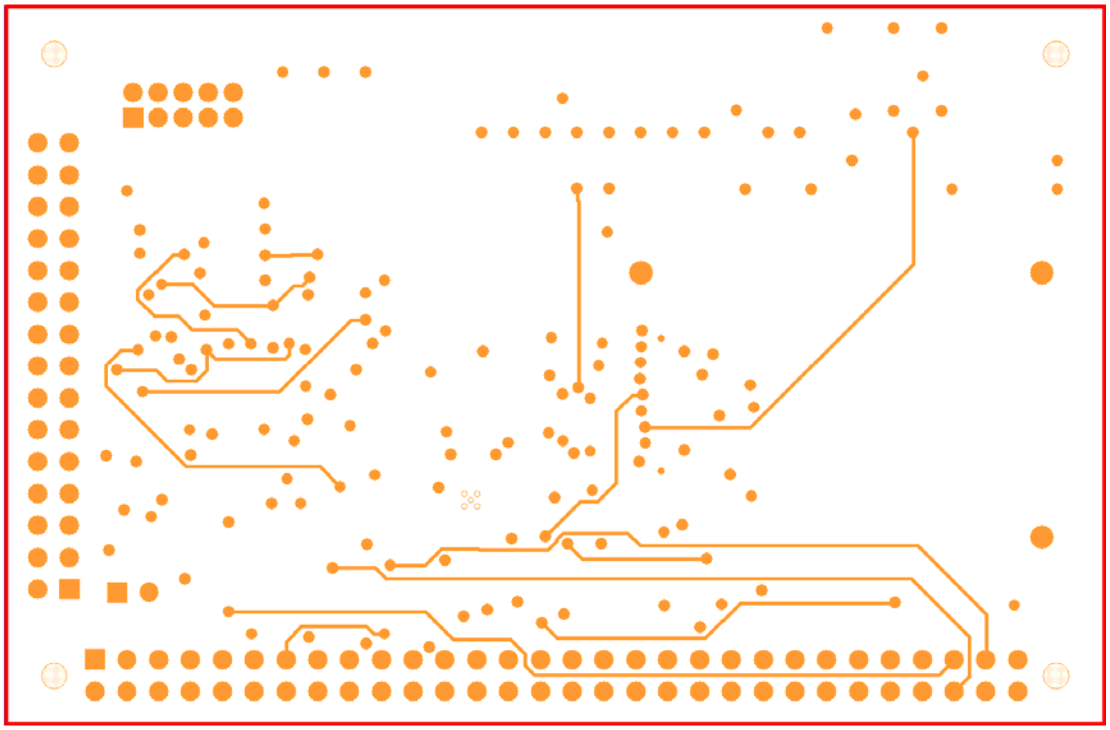


Figure D.8: Fifth Sensor PCB Layer

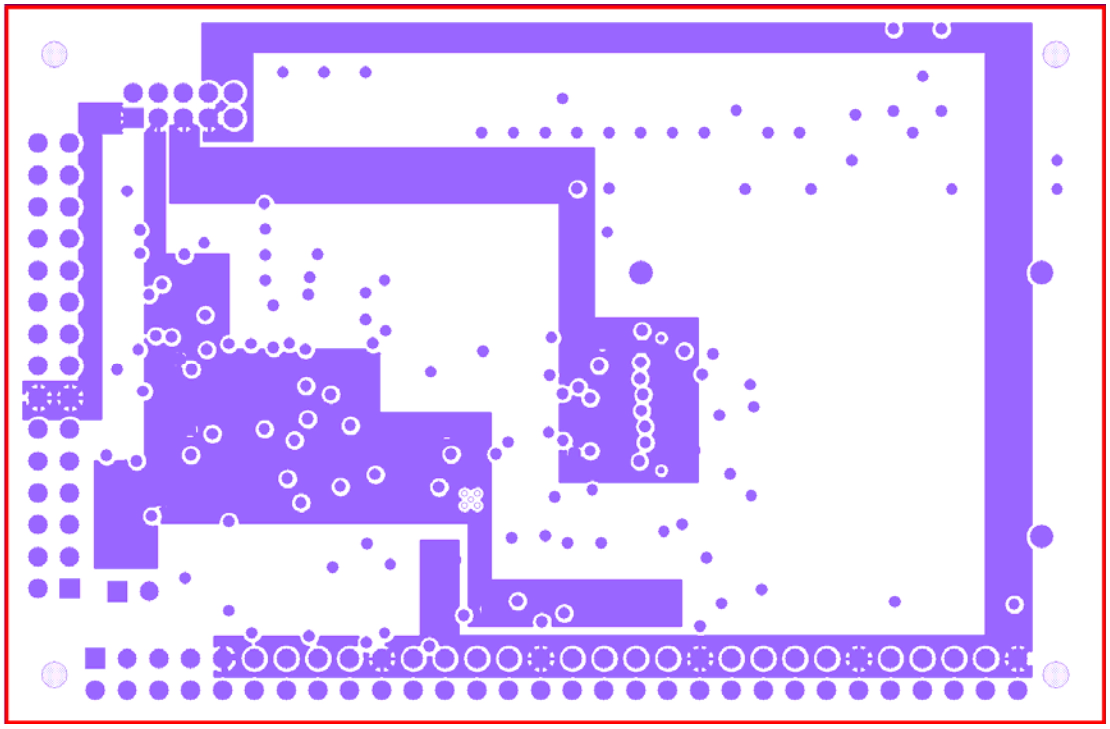


Figure D.9: Sixth Sensor PCB Layer (Power Supply Plane)

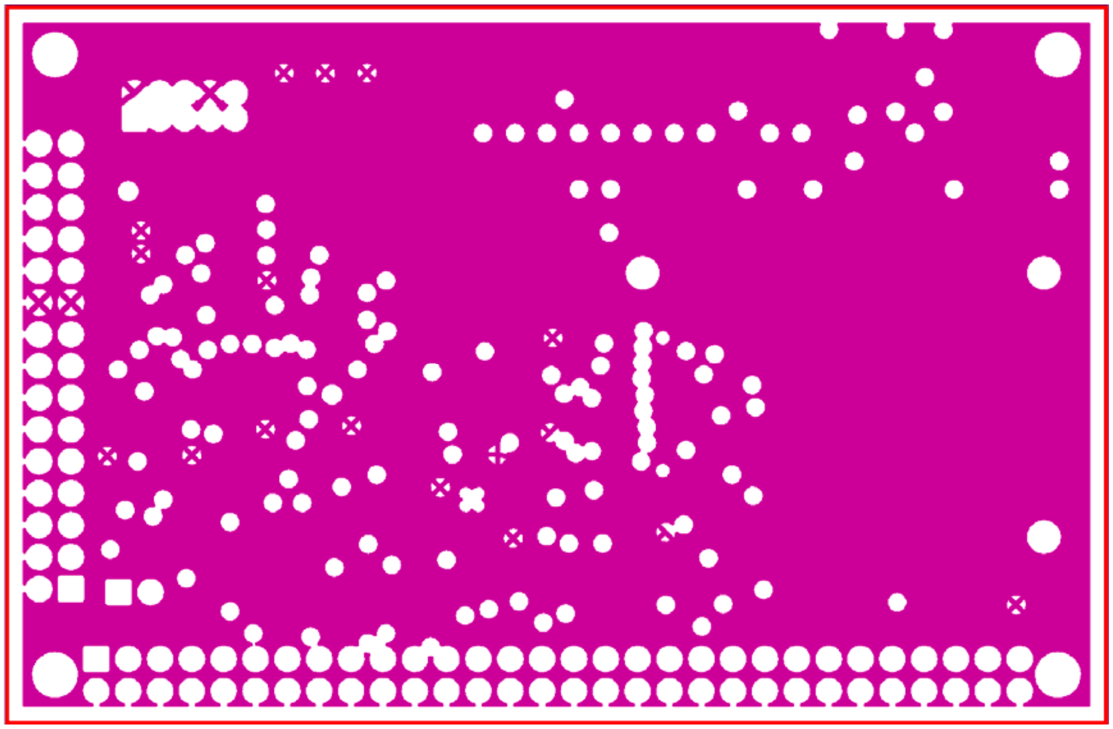


Figure D.10: Seventh Sensor PCB Layer (Power Ground Plane)

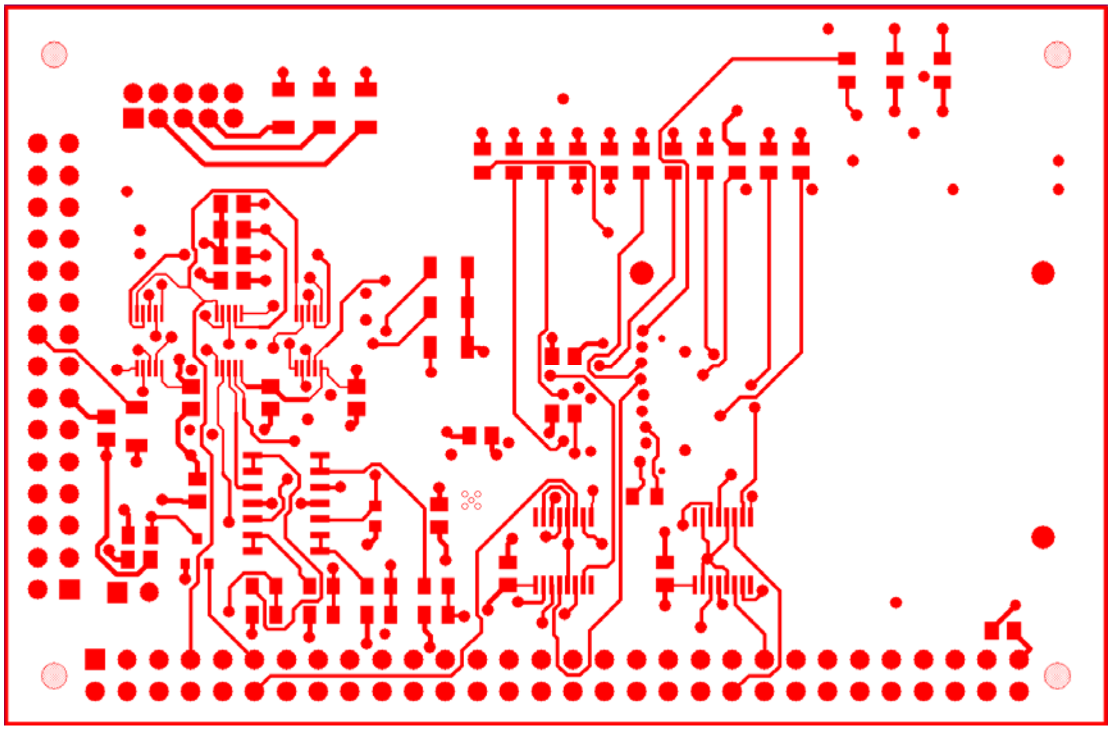


Figure D.11: Eighth Sensor PCB Layer

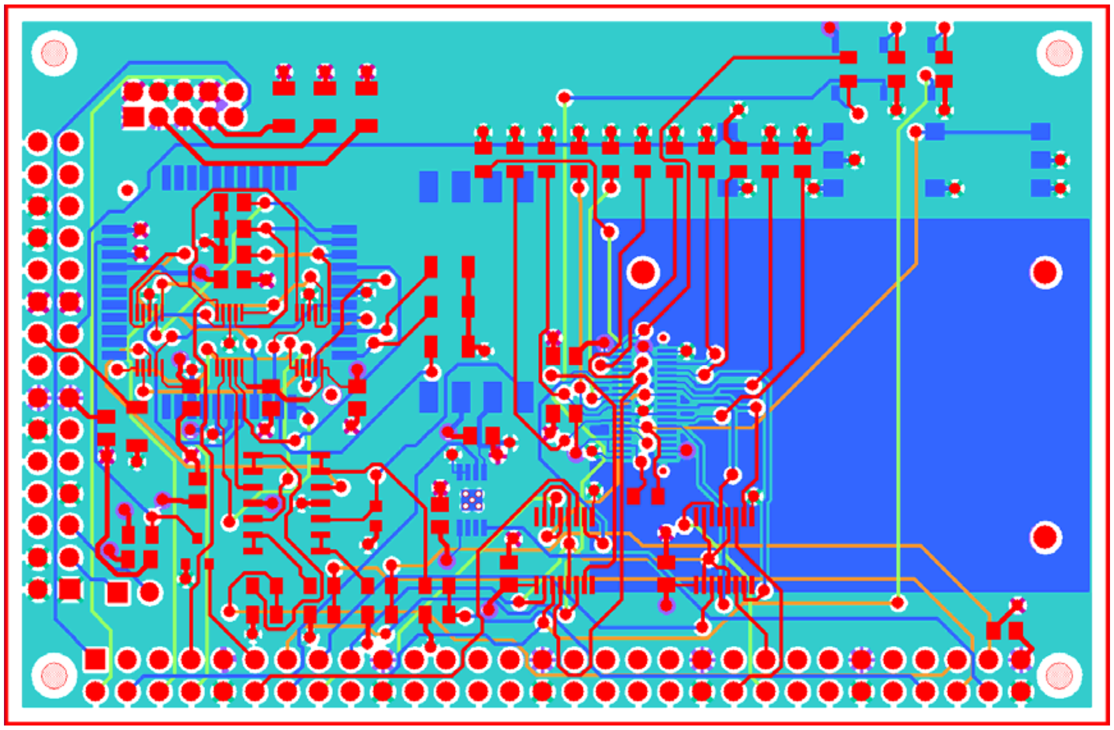


Figure D.12: All Sensor PCB Layers

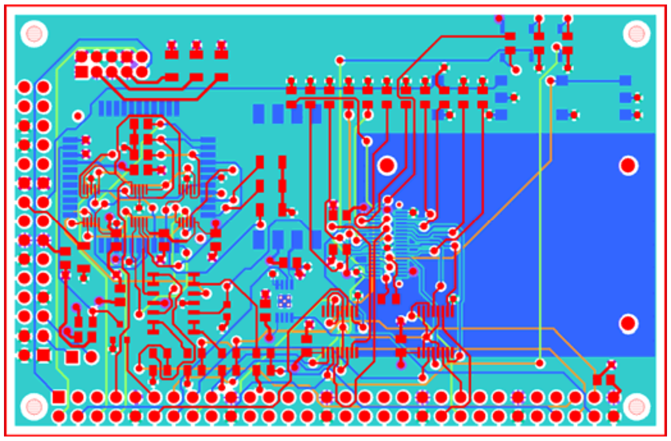


Figure D.13: All Sensor PCB Layers (Actual Size)

D.2. Power Supply Printed Circuit Board Layout

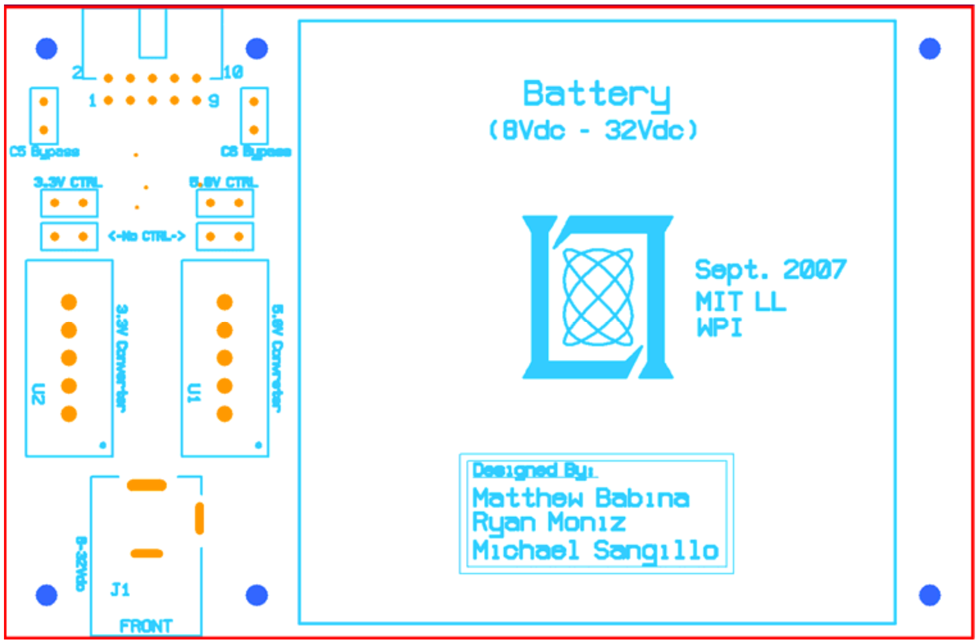


Figure D.14: Top Power Supply Board Silk Screen

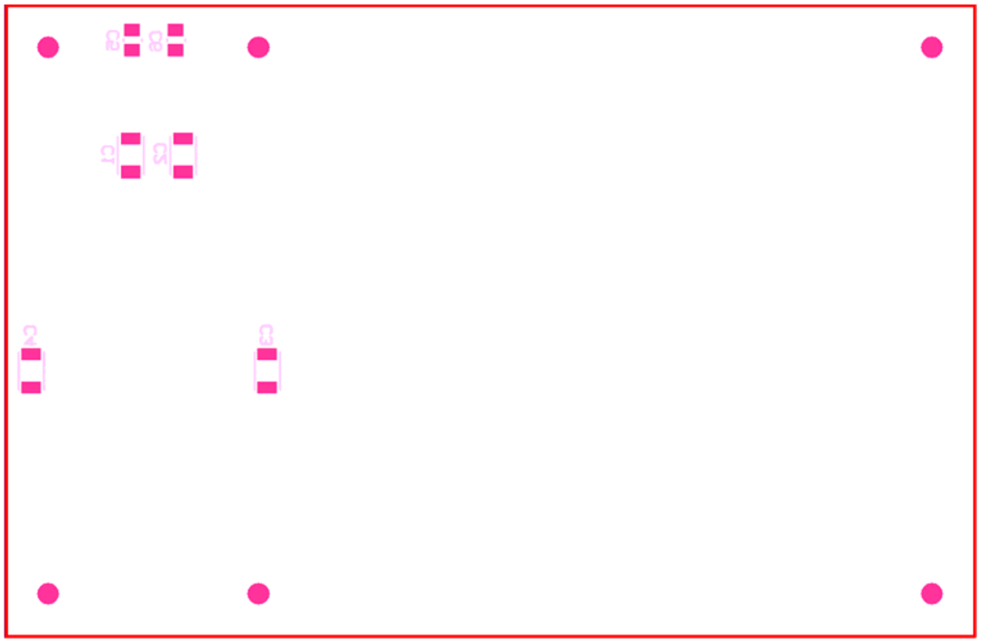
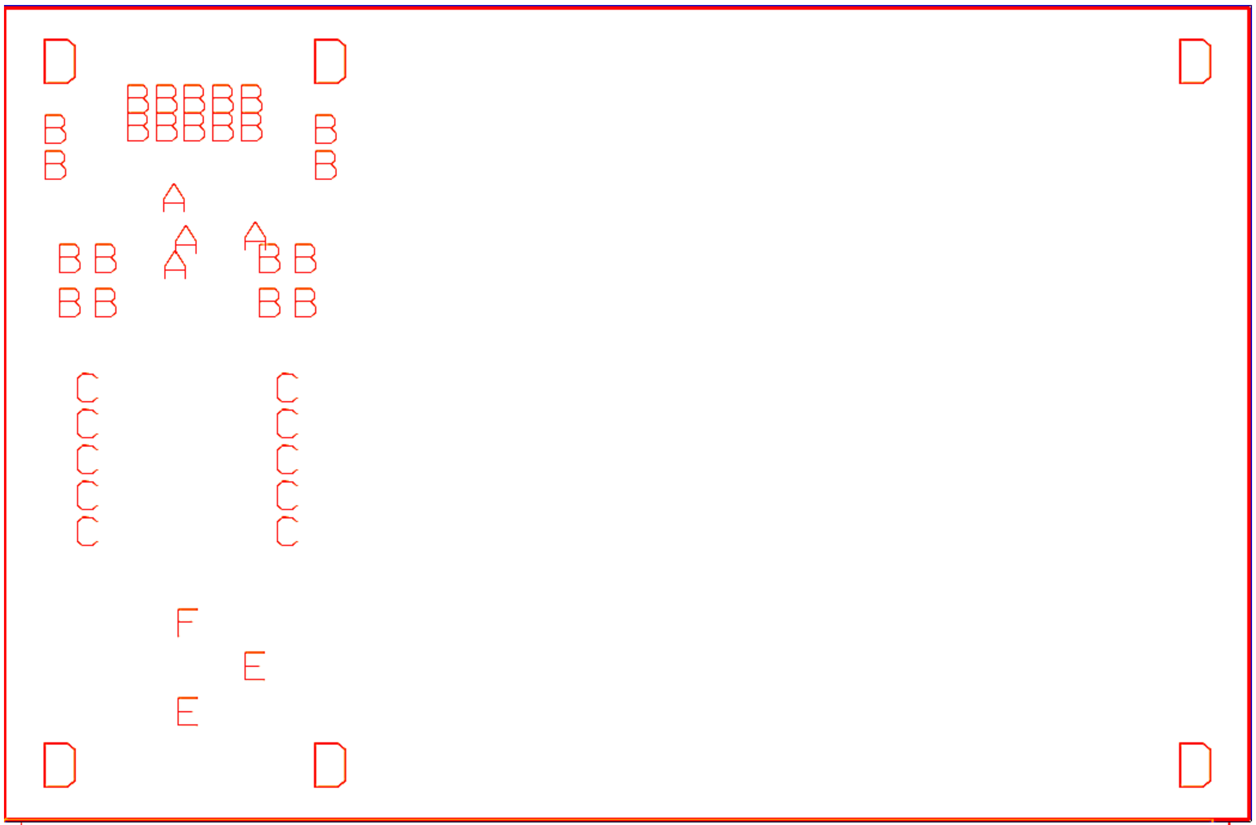


Figure D.15: Bottom Power Supply Board Silk Screen



Through Holes
All Drills (unless specified) +/- 0.003 (in)

Symbol	Diameter (in)	Tolerance (in)	Plated	Quantity
A	0.016		Yes	4
B	0.035	+/- 0.003	Yes	22
C	0.062	+/- 0.003	Yes	10
D	0.125	+/- 0.004	No	6
E	0.031		Yes	2
F	0.039		Yes	1

Figure D.16: Power Supply Board Drill Holes

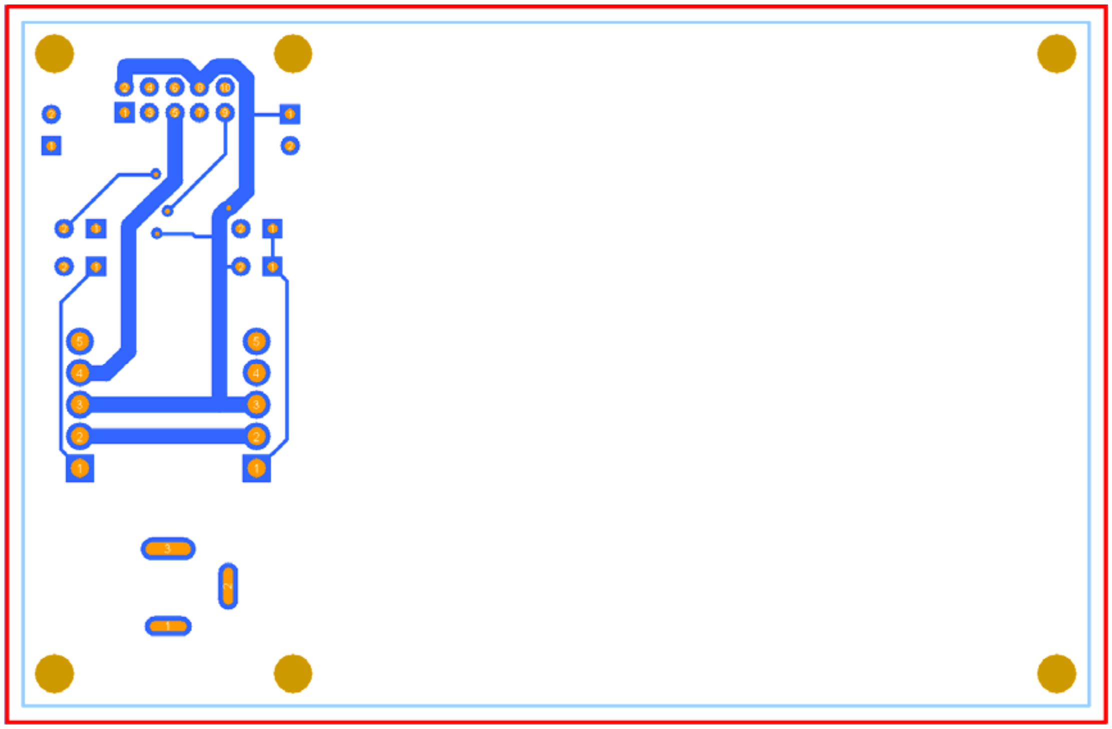


Figure D.17: First Power Supply PCB Layer

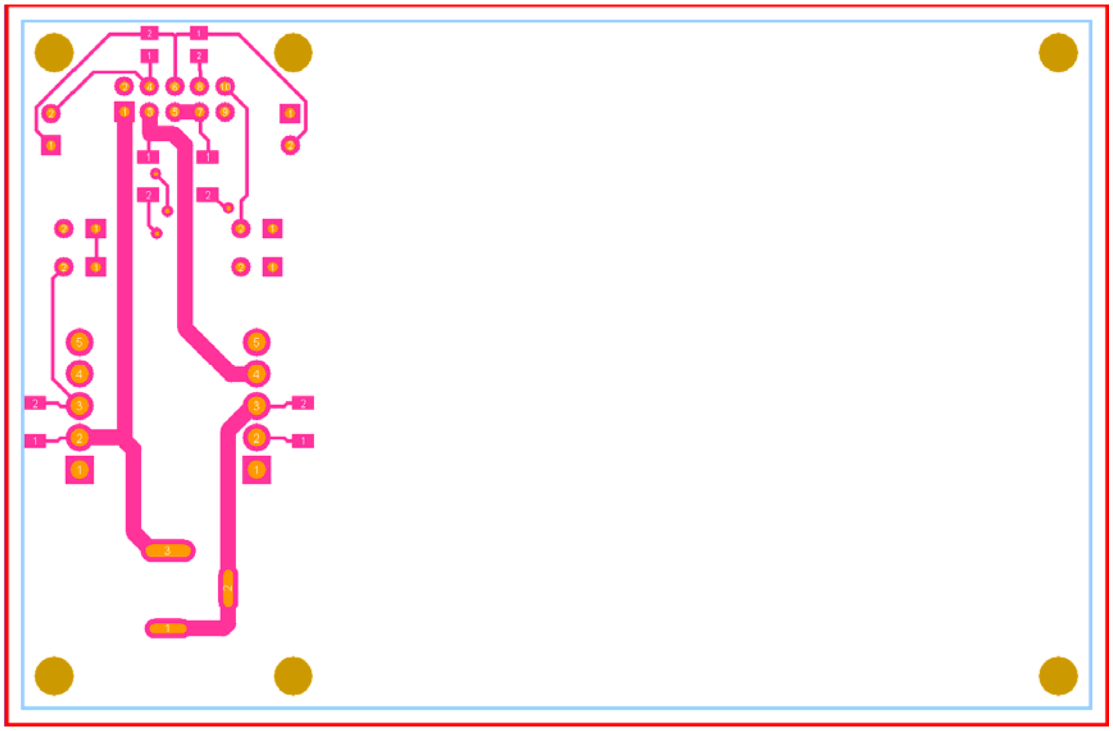


Figure D.18: Second Power Supply PCB Layer

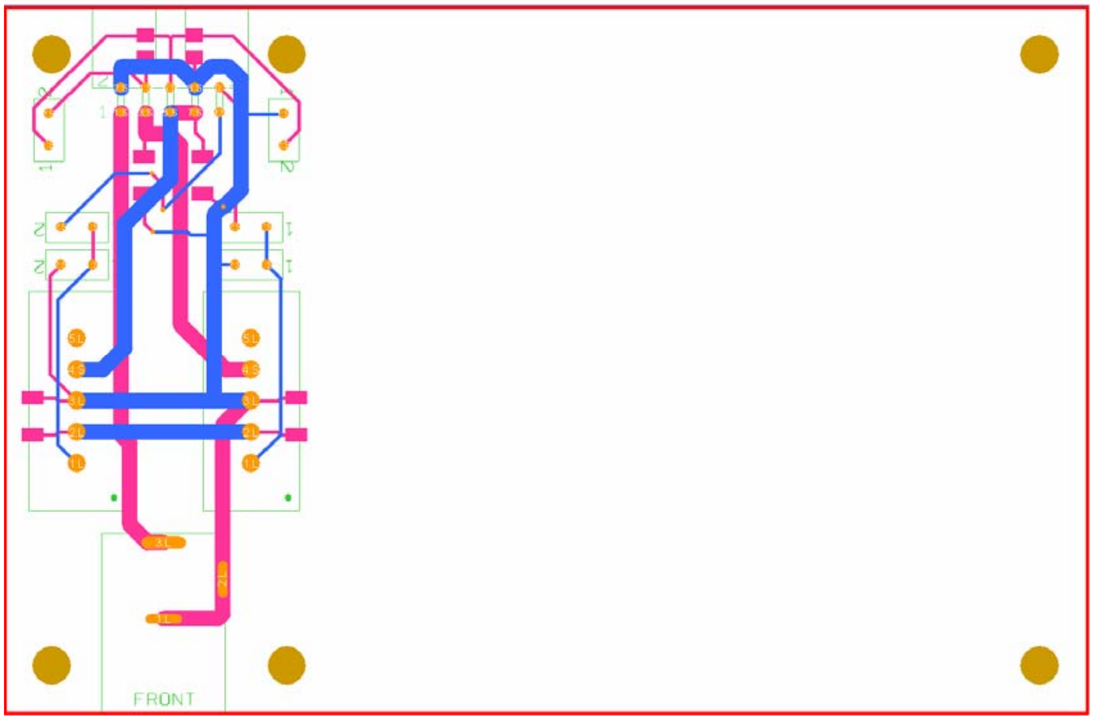


Figure D.19: All Power Supply PCB Layers

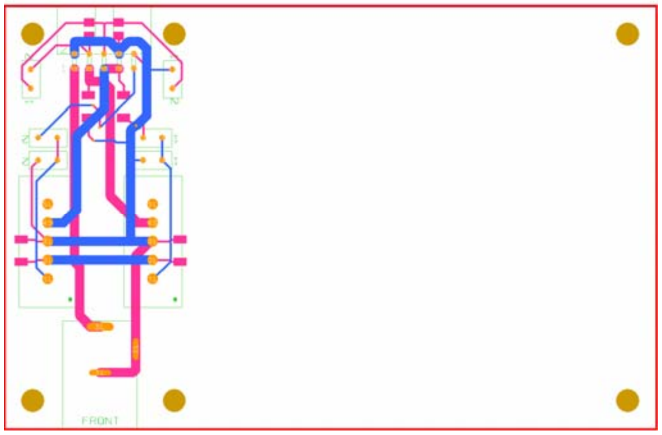


Figure D.20: All Power Supply PCB Layers (Actual Size)

D.3. Bill of Materials

Reference	Quantity	Manufacturer	Part Number	Description
C1-3	3	Panasonic©	ECJ3YB1E106M	10 μF SMT Capacitor
C4-6	3	Panasonic©	ECJ2FB1E225K	2.2 μF SMT Capacitor
C7-8; C10-21	14	AVX Corp©	08053D105KAT2A	1.0 μF SMT Capacitor
C9	1	AVX Corp©	06035A471JAT2A	470 pF SMT Capacitor
D1-3	3	CML Technologies©	CMD28-21SRC	SMT Clear Red LED
P1	1	Tyco International©	3-87215-0	60-Pin Male Header
P2	1	Molex©	90131-0135	30-Pin Male Header
P3	1	Molex©	87833-1021	10-Pin Right-Angle Male Header
P4	1	Tyco International©	87220-2	2-Pin Male Header
Q1	1	ON Semiconductor ©	MMBF0201NLT1	N-Channel MOSFET Transistor
R1-4	4	Panasonic©	ERA6YEB203V	0.1% Tolerant 20 kΩ SMT Resistor
R5-7; R28	4	Panasonic©	ERJ8ENF1004V	1.0 MΩ Resistor
R8-18	11	Susumu©	RR1220P-105-D	1.0 MΩ Resistor
R19-23; R27	6	Panasonic©	ERA6AEB303V	0.1% Tolerant 30 kΩ SMT Resistor
R24-26	3	Panasonic©	ERJ6GEYJ681V	680 Ω SMT Resistor
SW1-2	2	Alps©	SKHMQKE010	Push Button DPST NO Switch
U1	1	MemSense©	MAG10-1200S050	Tri-Axial Analog Inertial Sensor
U2-4	3	Analog Devices©	ADG704BRMZ	4:1 CMOS Analog Multiplexer
U5	1	Maxim©	DS600U	Analog-Output Temperature Sensor
U6	1	Motorola©	MPXA4350A6U	Analog-Output Pressure Sensor
U7	1	National©	LMV934MA	4-Channel Operational Amplifier
U8-9	2	Fairchild©	FST3126QSC	4-Bit Tri-State Bus Switch
U10	1	Quatech© Inc.	WLNB-AN-DP100	Airborne Embedded Wireless Module
	69			

Table D.1: Sensor Board Bill of Materials

Reference	Quantity	Manufacturer	Part Number	Description
C1-4	4	Panasonic©	ECJ3YB1E106M	10 μF Ceramic Capacitor
C5-6	2	AVX Corporation©	08053D105KAT2A	1.0 μF Ceramic Capacitor
J1	1	Switchcraft© Inc.	RAPC712BK	2.5x5.5 mm Right-Angle DC Barrel Jack
P1-6	6	Tyco International©	87220-2	2-Pin Male Header
P7	1	Molex©	87833-1021	10-Pin Right-Angle Male Header
U1	1	Bel Fuse© Inc.	V7AH-03H500	5.0V, 3.0A DC/DC Switching Regulator
U2	1	Bel Fuse© Inc.	V7AH-03H330	3.3V, 3.0A DC/DC Switching Regulator
	16			

Table D.2: Power Supply Board Bill of Materials

Appendix E: MCETS Client Firmware

E.1. Texas Instruments© MSP430 Assembly Instruction Set

Mnemonic		**Description**		**V**	**N**	**Z**	**C**
ADC(.B)†	dst	Add C to destination	dst + C → dst	*	*	*	*
ADD(.B)	src, dst	Add source to destination	src + dst → dst	*	*	*	*
ADDC(.B)	src, dst	Add source and C to destination	src + dst + C → dst	*	*	*	*
AND(.B)	src, dst	AND source and destination	src .and. dst → dst	0	*	*	*
BIC(.B)	src, dst	Clear bits in destination	(.not.src) .and. dst → dst	–	–	–	–
BIS(.B)	src, dst	Set bits in destination	src .or. dst → dst	–	–	–	–
BIT(.B)	src, dst	Test bits in destination	src .and. dst	0	*	*	*
BR†	dst	Branch to destination	dst → PC	–	–	–	–
CALL	dst	Call destination	PC + 2 → stack, dst → PC	–	–	–	–
CLR(.B)†	dst	Clear destination	0 → dst	–	–	–	–
CLRC†		Clear C	0 → C	–	–	–	0
CLRN†		Clear N	0 → N	–	0	–	–
CLRZ†		Clear Z	0 → Z	–	–	0	–
CMP(.B)	src, dst	Compare source and destination	dst – src	*	*	*	*
DADC(.B)†	dst	Add C decimally to destination	dst + C → dst (decimally)	*	*	*	*
DADD(.B)	src, dst	Add source and C decimally to destination	src + dst + C → dst (decimally)	*	*	*	*
DEC(.B)†	dst	Decrement destination	dst – 1 → dst	*	*	*	*
DECD(.B)†	dst	Double-decrement destination	dst – 2 → dst	*	*	*	*
DINT†		Disable interrupts	0 → GIE	–	–	–	–
EINT†		Enable interrupts	1 → GIE	–	–	–	–
INC(.B)†	dst	Increment destination	dst + 1 → dst	*	*	*	*
INCD(.B)†	dst	Double increment destination	dst + 2 → dst	*	*	*	*
INV(.B)†	dst	Invert destination	.not.dst → dst	*	*	*	*
JC/JHS	label	Jump if C set / Jump if higher or same		–	–	–	–
JEQ/JZ	label	Jump if equal / Jump if Z set		–	–	–	–
JGE	label	Jump if greater or equal		–	–	–	–
JL	label	Jump if less		–	–	–	–
JMP	label	Jump	PC + 2 × offset → PC	–	–	–	–
JN	label	Jump if N set		–	–	–	–
JNC/JLO	label	Jump if C not set / Jump if lower		–	–	–	–
JNE/JNZ	label	Jump if not equal / Jump if Z not set		–	–	–	–

MOV(.B)	src, dst	Move source to destination	src → dst	–	–	–	–
NOP†		No operation		–	–	–	–
POP(.B)†	dst	Pop item from stack to destination	@SP → dst, SP + 2 → SP	–	–	–	–
PUSH(.B)	src	Push source onto stack	SP − 2 → SP, src → @SP	–	–	–	–
RET†		Return from subroutine	@SP → PC, SP + 2 → SP	–	–	–	–
RETI		Return from interrupt		*	*	*	*
RLA(.B)†	dst	Rotate left arithmetically		*	*	*	*
RLC(.B)†	dst	Rotate left through C		*	*	*	*
RRA(.B)	dst	Rotate right arithmetically		0	*	*	*
RRC(.B)	dst	Rotate right through C		*	*	*	*
SBC(.B)†	dst	Subtract not(C) from destination	dst + 0xFFFF + C → dst	*	*	*	*
SETC†		Set C	1 → C	–	–	–	1
SETN†		Set N	1 → N	–	1	–	–
SETZ†		Set Z	1 → Z	–	–	1	–
SUB(.B)	src, dst	Subtract source from destination	dst + not.src + 1 → dst	*	*	*	*
SUBC(.B)	src, dst	Subtract source and not(C) from destination	dst + not.src + C → dst	*	*	*	*
SWPB	dst	Swap bytes		–	–	–	–
SXT	dst	Extend sign		0	*	*	*
TST(.B)†	dst	Test destination	dst + 0xFFFF + 1	0	*	*	1
XOR(.B)	src, dst	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

† Emulated Instruction

E.2. Packet Format for Data Transmitted from the Server to the Clients

16-bit "Wake-Up" Subpacket															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Module ID Number (Last 8 bits of IP Address)								Data Acquisition Rate (Hertz)							

16-bit "Length of Data Acquisition" Subpacket															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Length of Data Acquisition (seconds) (Minimum Length of Time: 1 second) (Maximum Length of Time: 65,535 seconds (18 hours, 12 minutes, and 15 seconds)) (Acquire Data Indefinitely: 0 seconds)															

16-bit "What Data" Subpacket							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
Temperature	Pressure	Acceleration	Angular Rate	Magnetic Field Strength	Reserved	Reserved	Reserved
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Cartesian Position	Geodetic Position	Cartesian Velocity	Geodetic Velocity	Receiver Time	Reserved	Reserved	Reserved

E.3. Packet Format for Data Transmitted from the Clients to the Server

16-bit "Data Status" Subpacket

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Module ID Number (Last 8 bits of IP Address)								Number of Bytes to Transmit (Integer Ranging from 6 to 130)							

16-bit Temperature Data Subpacket
(If Requested)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Temperature Data											

16-bit Pressure Data Subpacket
(If Requested)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Pressure Data											

16-bit *x*-Axis Acceleration Data Subpacket
(If Acceleration is Requested)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	*x*-Axis Acceleration Data											

16-bit *y*-Axis Acceleration Data Subpacket
(If Acceleration is Requested)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	*y*-Axis Acceleration Data											

16-bit *z*-Axis Acceleration Data Subpacket
(If Acceleration is Requested)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	*z*-Axis Acceleration Data											

16-bit *x*-Axis Angular Rate Data Sub-Packet
(If Angular Rate is Requested)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	*x*-Axis Angular Rate Data											

16-bit *y*-Axis Angular Rate Data Subpacket
(If Angular Rate is Requested)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	0	0	0	*y*-Axis Angular Rate Data											

16-bit *z*-Axis Angular Rate Data Subpacket
(If Angular Rate is Requested)

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	*z*-Axis Angular Rate Data											

16-bit *x*-Axis Magnetic Field Data Subpacket
(If Magnetic Field Data is Requested)

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	*x*-Axis Magnetic Field Data											

16-bit *y*-Axis Magnetic Field Data Subpacket
(If Magnetic Field Data is Requested)

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	*y*-Axis Magnetic Field Data											

16-bit *z*-Axis Magnetic Field Data Subpacket
(If Magnetic Field Data is Requested)

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	*z*-Axis Magnetic Field Data											

16-bit *x*-Axis Internal Temperature Data Subpacket
(If Acceleration, Angular Rate, or Magnetic Field Data is Requested)

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	*x*-Axis Internal Temperature Data											

16-bit *y*-Axis Internal Temperature Data Subpacket
(If Acceleration, Angular Rate, or Magnetic Field Data is Requested)

15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	*y*-Axis Internal Temperature Data											

16-bit *z*-Axis Internal Temperature Data Subpacket
(If Acceleration, Angular Rate, or Magnetic Field Data is Requested)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	*z*-Axis Internal Temperature Data											

64-bit *x*-Axis Cartesian Position Data Subpacket
(If Cartesian Position is Requested)

63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**
Sign	Exponent											Mantissa			
47	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**

Mantissa (continued)															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Mantissa (continued)															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

64-bit *y*-Axis Cartesian Position Data Subpacket

(If Cartesian Position is Requested)

63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**
Sign	Exponent											Mantissa			
47	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Mantissa (continued)															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Mantissa (continued)															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

64-bit *z*-Axis Cartesian Position Data Subpacket

(If Cartesian Position is Requested)

63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**
Sign	Exponent											Mantissa			
47	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Mantissa (continued)															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Mantissa (continued)															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

64-bit Latitude Data Subpacket

(If Geodetic Position is Requested)

63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**
Sign	Exponent											Mantissa			
47	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Mantissa (continued)															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Mantissa (continued)															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

64-bit Longitude Data Subpacket

(If Geodetic Position is Requested)

63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**
Sign	Exponent											Mantissa			
47	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**

Mantissa (continued)															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Mantissa (continued)															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

64-bit Altitude Data Subpacket

(If Geodetic Position is Requested)

63	**62**	**61**	**60**	**59**	**58**	**57**	**56**	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**
Sign	Exponent											Mantissa			
47	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
Mantissa (continued)															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Mantissa (continued)															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

32-bit *x*-Axis Cartesian Velocity Data Subpacket

(If Cartesian Velocity is Requested)

31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Sign	Exponent								Mantissa						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

32-bit *y*-Axis Cartesian Velocity Data Subpacket

(If Cartesian Velocity is Requested)

31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Sign	Exponent								Mantissa						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

32-bit *z*-Axis Cartesian Velocity Data Subpacket

(If Cartesian Velocity is Requested)

31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Sign	Exponent								Mantissa						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

32-bit Northing Velocity Data Subpacket

(If Geodetic Velocity is Requested)

31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Sign	Exponent								Mantissa						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

32-bit Easting Velocity Data Subpacket
(If Geodetic Velocity is Requested)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Sign	Exponent								Mantissa						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

32-bit Altitude Velocity Data Subpacket
(If Geodetic Velocity is Requested)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Sign	Exponent								Mantissa						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

32-bit HDOP Data Subpacket
(If Position or Velocity is Requested)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Sign	Exponent								Mantissa						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

32-bit VDOP Data Subpacket
(If Position or Velocity is Requested)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Sign	Exponent								Mantissa						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

32-bit TDOP Data Subpacket
(If Position or Velocity is Requested)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Sign	Exponent								Mantissa						
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mantissa (continued)															

24-bit Satellite Statistics Data Sub-Packet
(If Position or Velocity is Requested)

23	22	21	20	19	18	17	16
Number of GPS Satellites Locked							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Number of GPS Satellites Available								Number of GPS Satellites Used in Positioning							

72-bit Receiver Time Data Subpacket (If Receiver Time is Requested)															
71	**70**	**69**	**68**	**67**	**66**	**65**	**64**	**63**	**62**	**61**	**60**	**59**	**58**	**57**	**56**
Year (1 – 65,534)															
55	**54**	**53**	**52**	**51**	**50**	**49**	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**
Month (1 – 12)								Day (1 – 31)							
38		**38**		**37**		**36**		**35**		**34**		**33**		**32**	
Receiver Reference Time (0 = GPS; 1 = UTC USNO; 2 = GLONASS; 3 = UTC SU)															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Receiver Time (milliseconds) (0 – 86,400,000)															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Receiver Time (continued)															

32-bit Battery Voltage Data Subpacket															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Battery Voltage Character 4								Battery Voltage Character 3							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Battery Voltage Character 2								Battery Voltage Character 1							

16-bit “Module Status” Subpacket															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Data Acquisition Rate (Hertz)								0	Acquisition Complete	0	Transmission Complete	0	Oscillator Fault Flag	0	1

E.4. Data Format for Standard GRIL Output Messages

Field	Value	Bytes	Format
Message ID	PO (0x504F)	2	ASCII Character
Length of Message Body	30 (0x01E)	3	ASCII Character
x-Axis Cartesian Position (m)	–	8	Double-Precision Floating Point
y-Axis Cartesian Position (m)	–	8	Double-Precision Floating Point
z-Axis Cartesian Position (m)	–	8	Double-Precision Floating Point
Position Spherical Error Probable	–	4	Single-Precision Floating Point
Solution Type	–	1	Unsigned Integer
Checksum	0x51	1	Unsigned Integer

Table E.1: GRIL Cartesian Position Output Message

Field	Value	Bytes	Format
Message ID	PG (0x5047)	2	ASCII Character
Length of Message Body	0x01E (30)	3	ASCII Character
Latitude (rad)	?	8	Double-Precision Floating Point
Longitude (rad)	?	8	Double -Precision Floating Point
Altitude (m)	?	8	Double -Precision Floating Point
Position Spherical Error Probable	?	4	Single-Precision Floating Point
Solution Type	?	1	Unsigned Integer
Checksum	0x51	1	Unsigned Integer

Table E.2: GRIL Geodetic Position Output Message

Field	Value	Bytes	Format
Message ID	VE (0x5645)	2	ASCII Character
Length of Message Body	18 (0x012)	3	ASCII Character
x-Axis Cartesian Velocity (m/s)	–	4	Single-Precision Floating Point
y-Axis Cartesian Velocity (m/s)	–	4	Single-Precision Floating Point
z-Axis Cartesian Velocity (m/s)	–	4	Single-Precision Floating Point
Velocity Spherical Error Probable	–	4	Single-Precision Floating Point
Solution Type	–	1	Unsigned Integer
Checksum	0x51	1	Unsigned Integer

Table E.3: GRIL Cartesian Velocity Output Message

Field	Value	Bytes	Format
Message ID	VG (0x5647)	2	ASCII Character
Length of Message Body	18 (0x012)	3	ASCII Character
Northing Velocity (m/s)	–	4	Single-Precision Floating Point
Easting Velocity (m/s)	–	4	Single-Precision Floating Point
Height Velocity (m/s)	–	4	Single-Precision Floating Point
Velocity Spherical Error Probable	–	4	Single-Precision Floating Point
Solution Type	–	1	Unsigned Integer
Checksum	0x51	1	Unsigned Integer

Table E.4: GRIL Geodetic Velocity Output Message

Field	Value	Bytes	Format
Message ID	DP (0x4450)	2	ASCII Character
Length of Message Body	14 (0x0E)	3	ASCII Character
HDOP	–	4	Single-Precision Floating Point
VDOP	–	4	Single-Precision Floating Point
TDOP	–	4	Single-Precision Floating Point
Solution Type	–	1	Unsigned Integer
Checksum	0x92	1	Unsigned Integer

Note: $\mathrm{GDOP} = \sqrt{\mathrm{HDOP}^2 + \mathrm{VDOP}^2 + \mathrm{TDOP}^2}$

$\mathrm{PDOP} = \sqrt{\mathrm{HDOP}^2 + \mathrm{VDOP}^2}$

Table E.5: GRIL Dilution of Precision Output Message

Field	Value	Bytes	Format
Message ID	PS (0x5053)	2	ASCII Character
Length of Message Body	9 (0x09)	3	ASCII Character
Solution Type	–	1	Unsigned Integer
Number of GPS Satellites Locked	–	1	Unsigned Integer
Number of GPS Satellites Available	–	1	Unsigned Integer
Number of GLONASS Satellites Locked	–	1	Unsigned Integer
Number of GLONASS Satellites Available	–	1	Unsigned Integer

Number of GPS Satellites Used in Positioning	–	1	Unsigned Integer
Number of GLONASS Satellites Used in Positioning	–	1	Unsigned Integer
Field	**Value**	**Bytes**	**Format**
Ambiguity Fixing Progress Indicator	–	1	Unsigned Integer
Checksum	0x49	1	Unsigned Integer

Table E.6: GRIL Satellite Statistics Output Message

Field	**Value**	**Bytes**	**Format**
Message ID	RD (0x5244)	2	ASCII Character
Length of Message Body	6 (0x06)	3	ASCII Character
Year	1 – 65,534	2	Unsigned Integer
Month	1 – 12	1	Unsigned Integer
Day	1 – 31	1	Unsigned Integer
Receiver Reference Time	0 – GPS 1 – UTC USNO 2 – GLONASS 3 – UTC SU	1	Unsigned Integer
Checksum	0x26	1	Unsigned Integer

Table E.7: GRIL Receiver Date Output Message

Field	**Value**	**Bytes**	**Format**
Message ID	~~ (0x7E7E)	2	ASCII Character
Length of Message Body	5 (0x05)	3	ASCII Character
Receiver Time (ms)	0 – 86400000	4	Unsigned Integer
Checksum	0x90	1	Unsigned Integer

Table E.8: GRIL Receiver Time Output Message

E.5. MCETS Client Firmware (Assembly Language)

```
; Multi-Client Embedded Telemetry System (MCETS) Client Firmware
; Matthew Babina
; Ryan T. Moniz
; Michael Sangillo
; MIT Lincoln Laboratory

#include "msp430x16x.h"

module_id   DEFINE   0x1E                                   ; 8-Bit module ID number (0x00 to 0xFF)
load_time   DEFINE   1000                                   ; Sensor load time [milliseconds]
data_rate   DEFINE   10                                     ; Default data rate [Hz]
data_time   DEFINE   60                                     ; Default length of data acquisition [seconds]
what_data   DEFINE   0xF8F8                                 ; Default "what data" subpacket (0xF8F8 = all data transmitted)
auto_load   DEFINE   0x0                                    ; MCETS load method (0x0 = low power; 0x1 = data acquisition)

LF          DEFINE   0x0A                                   ; ASCII line-feed character

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME     main                                   ; Start of the main module

            ORG      0A000h                                 ; Setup the GPS GRIL messages in Flash memory
set_br:     DC8      "set,dev/ser/a/rate,460800"            ; GRIL message to set serial port A's baud rate to 460800 bps
set_stop:   DC8      "set,dev/ser/a/stops,2"                ; GRIL message to configure serial port A for two stop bits
set_par:    DC8      "set,dev/ser/a/parity,odd"             ; GRIL message to configure serial port A for odd parity
res_GPS:    DC8      "init,/dev/nvm/a"                      ; GRIL message to reset and reboot the GPS receiver
GPS_lpm:    DC8      "set,lpm,on"                           ; GRIL message to put the GPS receiver into low-power mode
GPS_sleep:  DC8      "set,sleep,on"                         ; GRIL message to put the GPS receiver to sleep
car_pos:    DC8      "out,,jps/PO"                          ; GRIL message for Cartesian position
geo_pos:    DC8      "out,,jps/VE"                          ; GRIL message for Cartesian velocity
car_vel:    DC8      "out,,jps/PG"                          ; GRIL message for Geodetic position
geo_vel:    DC8      "out,,jps/VG"                          ; GRIL message for Geodetic velocity
DOP:        DC8      "out,,jps/DP"                          ; GRIL message for dilution of precision
pos_stats:  DC8      "out,,jps/PS"                          ; GRIL message for position statistics
rec_date:   DC8      "out,,jps/RD"                          ; GRIL message for receiver date
rec_time:   DC8      "out,,jps/RT"                          ; GRIL message for receiver time
bat_volt:   DC8      "print,pwr/board"                      ; GRIL message for battery voltage

            ORG      0FFFEh                                 ; Setup the reset interrupt vector (POR and PUC)
            DW       init_sys

            ORG      04000h                                 ; Set the program counter to the beginning of the Flash (code) memory
init_sys:
            MOV      #03900h, SP                            ; Set the stack pointer to the top of RAM
            MOV      #(WDTPW+WDTHOLD), &WDTCTL_             ; Turn the watchdog timer off
            EINT                                            ; Enable global interrupts

defaults:   MOV      #(module_id*00100h), R4                ; Store the module ID number in the "data status" subpacket (R4, bits 15-8)
            MOV      #(data_rate*00100h+1), R5              ; Store the (default) data rate in the "module status" subpacket (R5, bits 15-8)
            MOV      #data_rate, R6                         ; Store the (default) data rate in R6
            MOV      #data_time, R7                         ; Store the (default) length of data acquisition in R7
            MOV      #what_data, R8                         ; Store the (default) "what data" subpacket in R8

            MOV.B    #000FFh, &P1DIR_                       ; Set all I/O pins to output mode (conserves power)
            MOV.B    #000FFh, &P2DIR_
            MOV.B    #000FFh, &P3DIR_
            MOV.B    #000FFh, &P4DIR_
            MOV.B    #000FFh, &P5DIR_
            MOV.B    #000FFh, &P6DIR_

set_IO:     BIC.B    #(BIT1+BIT3), &P1SEL_                  ; Select I/O for P1.1 (5.0V supply control) and P1.3 (temperature sensor control)
            BIC.B    #(BIT5+BIT6+BIT7), &P1SEL_             ; Select I/O for P1.5-7 (analog multiplexer controls)
            BIC.B    #(BIT0+BIT1), &P2SEL_                  ; Select I/O for P2.0 (magnetometer reset circuit) and P2.1 (ewm load status)
            BIC.B    #(BIT2+BIT3), &P2SEL_                  ; Select I/O for P2.2 (ewm peer connection status) and P2.3 (ewm IP connection status)
            BIC.B    #(BIT4+BIT5), &P2SEL_                  ; Select I/O for P2.4 (ewm data framing) and P2.5 (ewm data availability)
            BIC.B    #BIT6, &P2SEL_                         ; Select I/O for P2.6 (bus switch control)
            BIS.B    #(BIT4+BIT5), &P3SEL_                  ; Select UTXD0 and URXD0 for P3.4 and P3.5
            BIS.B    #(BIT1+BIT2+BIT3), &P5SEL_             ; Select SIMO1, SOMI1, and UCLK1 for P5.1, P5.2, and P5.3
            BIC.B    #BIT0, &P6SEL_                         ; Select I/O for P6.0 (oscillator fault LED)
            BIS.B    #(BIT3+BIT4+BIT5+BIT6+BIT7), &P6SEL_   ; Select ADC12 for P6.3-7

            BIC.B    #(BIT1+BIT2+BIT3+BIT5), &P2DIR_        ; Set P2.1, P2.2, P2.3, and P2.5 to input mode
            BIC.B    #BIT5, &P3DIR_                         ; Set P3.5 to input mode
            BIC.B    #BIT2, &P5DIR_                         ; Set P5.2 to input mode
            BIC.B    #(BIT3+BIT4+BIT5+BIT6+BIT7), &P6DIR_   ; Set P6.3-7 to input mode

            BIS.B    #(BIT1+BIT3), &P1OUT_                  ; Turn the 5.0V supply and temperature sensor off
            BIC.B    #(BIT5+BIT6+BIT7), &P1OUT_             ; Turn the analog multiplexers off
            BIC.B    #BIT0, &P2OUT_                         ; Turn the magnetometer reset circuit off
            BIS.B    #BIT4, &P2OUT_                         ; Turn data framing off
            BIC.B    #BIT6, &P2OUT_                         ; Turn the bus switches off
            BIS.B    #BIT0, &P6OUT_                         ; Turn the oscillator fault LED off

set_ewm:    BIS.B    #BIT6, &P2OUT_                         ; Turn the bus switches on
            CLR      R13
            BIT.B    #BIT1, &P2IN_                          ; Determine if the embedded wireless module (ewm) failed to load (P2.1 = 0x0)
            JZ       ewm_noLO
            BIT.B    #BIT2, &P2IN_                          ; Determine if the ewm is not connected to an access point (P2.2 = 0x0)
            JZ       ewm_noAP
            BIT.B    #BIT3, &P2IN_                          ; Determine if the ewm does not have an IP connection (P2.3 = 0x0)
            JZ       ewm_noIP
            JMP      ewm_yesLO                              ; Continue configuring if the ewm successfully connected to the network
ewm_noLO:   BIC.B    #BIT1, &P2IES_                         ; Enable P2.1 is issue an interrupt if the ewm loads successfully (low-to-high)
            BIC.B    #BIT1, &P2IFG_
            BIS.B   #BIT1, &P2IE_
            BIS     #00400h, R13                                ; Note the system is in LPM4
            BIS     #LPM4, SR                                   ; Enter LPM4 (CPU, MCLK, SMCLK, ACLK, and DCO disabled)
            JMP     set_ewm
ewm_noAP:   BIC.B   #BIT2, &P2IES_                              ; Enable P2.2 to issue an interrupt if the ewm connects to an access point (low-to-high)
            BIC.B   #BIT2, &P2IFG_
            BIS.B   #BIT2, &P2IE_
            BIS     #00400h, R13                                ; Note the system is in LPM4
            BIS     #LPM4, SR                                   ; Enter LPM4 (CPU, MCLK, SMCLK, ACLK, and DCO disabled)
            JMP     set_ewm
ewm_noIP:   BIC.B   #BIT3, &P2IES_                              ; Enable P2.3 to issue an interrupt if the ewm makes an IP connection (low-to-high)
            BIC.B   #BIT3, &P2IFG_
            BIS.B   #BIT3, &P2IE_
            BIS     #00400h, R13                                ; Note the system is in LPM4
            BIS     #LPM4, SR                                   ; Enter LPM4 (CPU, MCLK, SMCLK, ACLK, and DCO disabled)
            JMP     set_ewm
ewm_yesLO:  BIS.B   #BIT1, &P2IES_                              ; Enable P2.1 to issue an interrupt if the ewm fails to load (high-to-low)
            BIC.B   #BIT1, &P2IFG_
            BIS.B   #BIT1, &P2IE_
ewm_yesAP:  BIS.B   #BIT2, &P2IES_                              ; Enable P2.2 to issue an interrupt if the ewm loses its AP connection (high-to-low)
            BIC.B   #BIT2, &P2IFG_
            BIS.B   #BIT2, &P2IE_
ewm_yesIP:  BIS.B   #BIT3, &P2IES_                              ; Enable P2.3 to issue an interrupt if the ewm loses its IP connection (high-to-low)
            BIC.B   #BIT3, &P2IFG_
            BIS.B   #BIT3, &P2IE_

setup_GPS:  NOP

setup_SPI:  MOV.B   #SWRST, &U1CTL_                             ; Hold the SPI peripheral for configuration
            BIS.B   #(CHAR+SYNC+MM), &U1CTL_                    ; Configure the SPI peripheral as a master
            MOV.B   #(SSEL1+STC), &U1TCTL_                      ; Set the SPI peripheral master clock to 1.5 MHz (SMCLK/2)
            CLR     &U1BR1_
            MOV     #00002h, &U1BR0_
            CLR     &U1MCTL_
            BIC.B   #BIT5, &P2IES_                              ; Enable P2.5 to issue an interrupt if the ewm receives data from the server (low-to-high)
            BIC.B   #BIT5, &P2IFG_
            BIS.B   #BIT5, &P2IE_

            CLR     R15
            CMP     #auto_load, R15
            JNZ     main                                        ; Automatically start acquiring data if auto_load = 0x1
ent_LPM3:   BIS.B   #BIT0, &P6OUT_                              ; Turn the oscillator fault LED off
            BIS     #00003h, R13                                ; Note the system is in LPM3
            BIS     #LPM3, SR                                   ; Enter LPM3 (only ACLK active) and wait to be queried by the server

main:
            BIC.B   #BIT0, &P6OUT_                              ; Turn the LED on to indicate an oscillator fault
            BIS.B   #SELM_0, &BCSCTL2_                          ; Select the digitally controlled oscillator for MCLK (~800 kHz)
            BIC.B   #XT2OFF, &BCSCTL1_                          ; Turn the high-frequency XT2 oscillator on (~6.00 MHz)
tst_XT2CLK: BIC.B   #OFIFG, &IFG1_                              ; Clear the oscillator fault flag
            MOV     #000FFh, R15                                ; Allow time for the oscillator fault flag to set
CLK_D       DEC     R15
            JNZ     CLK_D
            BIT.B   #OFIFG, &IFG1_                              ; Wait for XT2CLK to start (i.e., no oscillator fault)
            JNZ     tst_XT2CLK
            BIS.B   #SELM_2, &BCSCTL2_                          ; Select XT2CLK for MCLK (~6.00 MHz)
            BIS.B   #(SELS+DIVS_1), &BCSCTL2_                   ; Select XT2CLK (divided by 2) for SMCLK (~3.0 MHz)
            BIS.B   #(OFIE+ACCVIE), &IE1_                       ; Enable the oscillator fault and Flash access violation interrupts
            BIC     #BIT2, R5                                   ; Clear the oscillator fault flag in the module status subpacket (R5, bit 2)
            BIS.B   #BIT0, &P6OUT_                              ; Turn the oscillator fault LED off

init_sens:
            BIS     #TACLR, &TACTL_                             ; Clear the Timer A counter
            MOV     #(load_time*4096/1000), &TACCR0_            ; Set the Timer A capture value to the system load time
            MOV     #CCIE, &TACCTL0_                            ; Enable Timer A to issue an interrupt when the timer counts to the value in TACCR0
            MOV     #(TASSEL_1+ID_3+MC_1+TAIE), &TACTL_         ; Select ACLK (divided by 8) for the Timer A clock, and start the timer in up mode

init_5V:    BIT     #(BITB+BITC+BITD+BITE), R8                  ; Determine if the 5.0V supply is needed
            JZ      init_temp
            BIC.B   #BIT1, &P1OUT_                              ; Turn the 5.0V supply on
init_temp:  BIT     #BITF, R8                                   ; Determine if temperature data was requested
            JZ      init_tas
            BIC.B   #BIT3, &P1OUT_                              ; Turn the temperature sensor on
init_tas:   BIT     #(BITB+BITC+BITD), R8                       ; Determine if the tri-axial inertial sensor is needed
            JZ      init_mag
            BIS.B   #BIT5, &P1OUT_                              ; Enable the analog multiplexers
init_mag:   BIT     #BITB, R8                                   ; Determine if magnetic field data was requested
            JZ      init_ADC
            BIS.B   #BIT0, &P2OUT_                              ; Start the magnetometer reset circuit
init_ADC:   BIT     #(BITB+BITC+BITD+BITE+BITF), R8             ; Determine if any analog sensor data was requested
            JZ      init_UART0
            BIS.B   #INCH_7, &ADC12MCTL0_                       ; Select the temperature sensor (P6.7) for ADC12-0 (3.3V & 0.0V references)
            BIS.B   #INCH_6, &ADC12MCTL1_                       ; Select the pressure sensor (P6.6) for ADC12-1 (3.3V & 0.0V references)
            BIS.B   #INCH_5, &ADC12MCTL2_                       ; Select the x-axis (P6.5) for ADC12-2 (3.3V & 0.0V references)
            BIS.B   #INCH_4, &ADC12MCTL3_                       ; Select the y-axis (P6.4) for ADC12-3 (3.3V & 0.0V references)
            BIS.B   #(INCH_3+EOS), &ADC12MCTL4_                 ; Select the z-axis (P6.3) for ADC12-4 (3.3V & 0.0V references)
init_UART0: MOV.B   #SWRST, &U0CTL_                             ; Hold UART0 for configuration
            BIS.B   #(PENA+SPB+CHAR), &U0CTL_                   ; Configure UART0 for odd parity, two stop bits, and 8-bit data length
            BIS.B   #SSEL1, &U0TCTL_                            ; Select SMCLK for the baud rate generator
            CLR.B   &U0BR1_                                     ; Set the baud rate to 115200 bps
            MOV.B   #0001Ah, &U0BR0_
            CLR.B   &U0MCTL_
sens_D:     BIS     #00003h, R13                                ; Note the system is in LPM3
```

```
            BIS      #LPM3, SR                                        ; Enter LPM3 and wait for the sensors to initialize
            BIC.B    #BIT0, &P2OUT_                                   ; Turn the magnetometer reset circuit off

acq_data:
set_lda:    MOV      R7, R9                                           ; Move the length of data acquisition into R9 (counter)
lim_dr:     CMP      #00065h, R10                                     ; Force the data rate to be less than 101 Hz (predefined maximum data rate)
            JL       set_dr
            MOV      #00064h, R10
set_dr:     MOV      R6, R10                                          ; Move the data rate into R10 (counter)

init_TB:    BIS      #TBCLR, &TBCTL_                                  ; Clear the Timer B counter
            MOV      #0B000h, R15                                     ; Move the start address of the period lookup table to R15
            ADD      R6, R15                                          ; Seek to the desired location in the period lookup table
            ADD      R6, R15
            DECD     R15
            MOV      @R15, &TBCCR0_                                   ; Set the Timer B capture value to the time from the period lookup table
            MOV      #CCIE, &TBCCTL0_                                 ; Enable Timer B to issue an interrupt when the timer counts to the value in TBCCR0
            MOV      #(TBSSEL_1+MC_1+TBIE), &TBCTL_                   ; Select ACLK for the Timer B clock, and start the timer in up mode

set_dapf:   BIS      #BIT6, R5                                        ; Set the data acquisition pending flag (indicates data is currently being acquired)
            BIC      #000FFh, R4                                      ; Clear the number of bytes to transmit in the "data status" subpacket
            INCD     R4                                               ; Add two transmitted bytes for the "data status" subpacket
            BIC      #0FF00h, R5                                      ; Clear the old data rate in the "module status" subpacket
            SWPB     R6
            BIS      R6, R5                                           ; Store the new data rate in the "module status" subpacket
            SWPB     R6
            MOV      #01106h, R11                                     ; Move the start of data acquisition RAM to R11

acq_temp:   BIT      #BITF, R8                                        ; Determine if temperature data was requested
            JZ       acq_pres
            MOV      #ADC12MEM0_, &DMA0SA_                            ; Set the DMA-0 source address to ADC12-0
            MOV      R11, &DMA0DA_                                    ; Set the DMA-0 destination address to the value in R11
            MOV      #00001h, &DMA0SZ_                                ; Configure DMA-0 to transfer two bytes of data
            BIC      #DMA0TSEL_15, &DMACTL0_                          ; Tigger DMA-0 when the DMAREQ bit is set
            MOV      #DMAEN, &DMA0CTL_                                ; Enable DMA-0 for single transfer
ADC12_D0:   BIT      #0FFFFh, &ADC12IE_                               ; Wait for all ADC12 conversions to finish
            JNZ      ADC12_D0
            MOV      #(SHP+ADC12SSEL_2+CSTARTADD_0), &ADC12CTL1_      ; Select MCLK and setup the ADC12 to sample the temperature sensor
            MOV      #BIT0, &ADC12IE_                                 ; Enable ADC12-0 to issue an interrupt request
            MOV      #(ADC12ON+ENC+ADC12SC+SHT0_3), &ADC12CTL0_       ; Turn the ADC12 on and start sampling the temperature data
            MOV      #BIT0, R12                                       ; Note that DMA-0 needs to be triggered
            INCD     R11                                              ; Increase the data desination address by two bytes
            INCD     R4                                               ; Add two transmitted bytes to the "data status" subpacket

acq_pres:   BIT      #BITE, R8                                        ; Determine if pressure data was requested
            JZ       acq_accel
            MOV      #ADC12MEM1_, &DMA1SA_                            ; Set the DMA-1 source address to ADC12-1
            MOV      R11, &DMA1DA_                                    ; Set the DMA-1 destination address to the value in R11
            MOV      #00001h, &DMA1SZ_                                ; Configure DMA-1 to transfer two bytes of data
            BIC      #DMA1TSEL_15, &DMACTL0_                          ; Tigger DMA-1 when the DMAREQ bit is set
            MOV      #DMAEN, &DMA1CTL_                                ; Enable DMA-1 for single transfer
ADC12_D1:   BIT      #0FFFFh, &ADC12IE_                               ; Wait for all ADC12 conversions to finish
            JNZ      ADC12_D1
            MOV      #(SHP+ADC12SSEL_2+CSTARTADD_1), &ADC12CTL1_      ; Select MCLK and setup the ADC12 to sample the pressure sensor
            MOV      #BIT1, &ADC12IE_                                 ; Enable ADC12-1 to issue an interrupt request
            MOV      #(ADC12ON+ENC+ADC12SC+SHT0_3), &ADC12CTL0_       ; Turn the ADC12 on and start sampling the pressure data
            MOV      #BIT1, R12                                       ; Note that DMA-1 needs to be triggered
            INCD     R11                                              ; Increase the data desination address by two bytes
            INCD     R4                                               ; Add two transmitted bytes to the "data status" subpacket

acq_accel:  BIT      #BITD, R8                                        ; Determine if acceleration data was requested
            JZ       acq_rate
            MOV      #ADC12MEM2_, &DMA2SA_                            ; Set the DMA-2 source address to ADC12-2
            MOV      R11, &DMA2DA_                                    ; Set the DMA-2 destination address to the value in R11
            MOV      #00003h, &DMA2SZ_                                ; Configure DMA-2 to transfer six bytes of data
            BIC      #DMA2TSEL_15, &DMACTL0_                          ; Tigger DMA-2 when the DMAREQ bit is set
            MOV      #(DMADT_1+DMADSTINCR_3+DMASRCINCR_3+DMAEN), &DMA2CTL_   ; Enable DMA-2 for block transfer
ADC12_D2:   BIT      #0FFFFh, &ADC12IE_                               ; Wait for all ADC12 conversions to finish
            JNZ      ADC12_D2
            MOV      #(SHP+ADC12SSEL_2+CSTARTADD_2+CONSEQ_1), &ADC12CTL1_   ; Select MCLK and setup the ADC12 to sample the three axes of the accelerometer
            MOV      #BIT4, &ADC12IE_                                 ; Enable ADC12-4 to issue an interrupt request
            BIC.B    #(BIT6+BIT7), &P1OUT_                            ; Select acceleration data on the analog multiplexers (A1|A0|EN = 0|0|1)
            NOP                                                       ; Wait for the analog multiplexers to switch
            MOV      #(MSC+ADC12ON+ENC+ADC12SC+SHT0_3), &ADC12CTL0_   ; Turn the ADC12 on, and start sampling the tri-axial acceleration data
            MOV      #BIT2, R12                                       ; Note that DMA-2 needs to be triggered
            ADD      #00006h, R11                                     ; Increase the data desination address by six bytes
            ADD      #00006h, R4                                      ; Add six transmitted bytes to the "data status" subpacket

acq_rate:   BIT      #BITC, R8                                        ; Determine if angular rate data was requested
            JZ       acq_mag
            MOV      #ADC12MEM2_, &DMA0SA_                            ; Set the DMA-0 source address to ADC12-2
            MOV      R11, &DMA0DA_                                    ; Set the DMA-0 destination address to the value in R11
            MOV      #00003h, &DMA0SZ_                                ; Configure DMA-0 to transfer six bytes of data
            BIC      #DMA0TSEL_15, &DMACTL0_                          ; Tigger DMA-0 when the DMAREQ bit is set
            MOV      #(DMADT_1+DMADSTINCR_3+DMASRCINCR_3+DMAEN), &DMA0CTL_   ; Enable DMA-0 for block transfer
ADC12_D3:   BIT      #0FFFFh, &ADC12IE_                               ; Wait for all ADC12 conversions to finish
            JNZ      ADC12_D3
            MOV      #(SHP+ADC12SSEL_2+CSTARTADD_2+CONSEQ_1), &ADC12CTL1_   ; Select MCLK and setup the ADC12 to sample the three axes of the gyroscope
            MOV      #BIT4, &ADC12IE_                                 ; Enable ADC12-4 to issue an interrupt request
            BIC.B    #BIT7, &P1OUT_                                   ; Select angular rate data on the analog multiplexers (A1|A0|EN = 0|1|1)
            BIS.B    #BIT6, &P1OUT_
            NOP                                                       ; Wait for the analog multiplexers to switch
            MOV      #(MSC+ADC12ON+ENC+ADC12SC+SHT0_3), &ADC12CTL0_   ; Turn the ADC12 on, and start sampling the tri-axial angular rate data
            MOV     #BIT0, R12                                                  ; Note that DMA-0 needs to be triggered
            ADD     #00006h, R11                                                ; Increase the data desination address by six bytes
            ADD     #00006h, R4                                                 ; Add six transmitted bytes to the "data status" subpacket

acq_mag:    BIT     #BITB, R8                                                   ; Determine if magnetic data was requested
            JZ      acq_itemp
            MOV     #ADC12MEM2_, &DMA1SA_                                       ; Set the DMA-1 source address to ADC12-2
            MOV     R11, &DMA1DA_                                               ; Set the DMA-1 destination address to the value in R11
            MOV     #00003h, &DMA1SZ_                                           ; Configure DMA-1 to transfer six bytes of data
            BIC     #DMA1TSEL_15, &DMACTL0_                                     ; Tigger DMA-1 when the DMAREQ bit is set
            MOV     #(DMADT_1+DMADSTINCR_3+DMASRCINCR_3+DMAEN), &DMA1CTL_       ; Enable DMA-1 for block transfer
ADC12_D4:   BIT     #0FFFFh, &ADC12IE_                                          ; Wait for all ADC12 conversions to finish
            JNZ     ADC12_D4
            MOV     #(SHP+ADC12SSEL_2+CSTARTADD_2+CONSEQ_1), &ADC12CTL1_        ; Select MCLK and setup the ADC12 to sample the three axes of the magnetometer
            MOV     #BIT4, &ADC12IE_                                            ; Enable ADC12-4 to issue an interrupt request
            BIS.B   #BIT7, &P1OUT_                                              ; Select magnetic field data on the analog multiplexers (A1|A0|EN = 1|0|1)
            BIC.B   #BIT6, &P1OUT_
            NOP                                                                 ; Wait for the analog multiplexers to switch
            MOV     #(MSC+ADC12ON+ENC+ADC12SC+SHT0_3), &ADC12CTL0_              ; Turn the ADC12 on, and start sampling the tri-axial magnetic field data
            MOV     #BIT1, R12                                                  ; Note that DMA-1 needs to be triggered
            ADD     #00006h, R11                                                ; Increase the data desination address by six bytes
            ADD     #00006h, R4                                                 ; Add six transmitted bytes to the "data status" subpacket

acq_itemp:  BIT     #(BITB+BITC+BITD), R8                                       ; Determine if internal temperature is needed
            JZ      prep_GPS
            MOV     #ADC12MEM2_, &DMA2SA_                                       ; Set the DMA-2 source address to ADC12-2
            MOV     R11, &DMA2DA_                                               ; Set the DMA-2 destination address to the value in R11
            MOV     #00003h, &DMA2SZ_                                           ; Configure DMA-2 to transfer six bytes of data
            BIC     #DMA2TSEL_15, &DMACTL0_                                     ; Tigger DMA-2 when the DMAREQ bit is set
            MOV     #(DMADT_1+DMADSTINCR_3+DMASRCINCR_3+DMAEN), &DMA2CTL_       ; Enable DMA-2 for block transfer
ADC12_D5:   BIT     #0FFFFh, &ADC12IE_                                          ; Wait for all ADC12 conversions to finish
            JNZ     ADC12_D5
            MOV     #(SHP+ADC12SSEL_2+CSTARTADD_2+CONSEQ_1), &ADC12CTL1_        ; Select MCLK and setup the ADC12 to sample the three axes of the internal temperature sensor
            MOV     #BIT4, &ADC12IE_                                            ; Enable ADC12-4 to issue an interrupt request
            BIS.B   #(BIT6+BIT7), &P1OUT_                                       ; Select internal temperature data on the analog multiplexers (A1|A0|EN = 1|1|1)
            NOP                                                                 ; Wait for the analog multiplexers to switch
            MOV     #(MSC+ADC12ON+ENC+ADC12SC+SHT0_3), &ADC12CTL0_              ; Turn the ADC12 on, and start sampling the tri-axial internal temperature sensor data
            MOV     #BIT2, R12                                                  ; Note that DMA-2 needs to be tiggered
            ADD     #00006h, R11                                                ; Increase the data desination address by six bytes
            ADD     #00006h, R4                                                 ; Add six transmitted bytes to the "data status" subpacket

prep_GPS:   BIT     #0FFFFh, &ADC12IE_                                          ; Wait for all ADC12 conversions to finish
            JNZ     prep_GPS
            MOV     #U0TXBUF_, &DMA0DA_                                         ; Set the DMA-0 destination address to the UART0 transmit buffer
            MOV     #U0RXBUF_, &DMA1SA_                                         ; Set the DMA-1 source address to the UART0 receive buffer
            MOV     #02000h, &DMA1DA_                                           ; Set the DMA-1 destination address to the GPS dump address
            MOV     #02005h, &DMA2SA_                                           ; Set the DMA-2 source address to the start of data in the GPS dump address
            MOV     #DMA0TSEL_4, DMACTL0_                                       ; Configure DMA-0 to transfer when the UART0 transmitter is available
            BIS     #DMA1TSEL_3, DMACTL0_                                       ; Configure DMA-1 to transfer when the UART0 receiver receives new data
            BIS     #DMA2TSEL_0, DMACTL0_                                       ; Configure DMA-2 to transfer when the DMAREQ bit is set
            MOV     #(DMADT_1+DMASRCINCR_3+DMASBDB), &DMA0CTL_                  ; Configure DMA-0 for (byte) block transfer
            MOV     #(DMADT_1+DMADSTINCR_3+DMASBDB), &DMA1CTL_                  ; Configure DMA-1 for (byte) block transfer
            MOV     #(DMADT_1+DMASRCINCR_3+DMADSTINCR_3+DMASBDB), &DMA2CTL_     ; Configure DMA-2 for (byte) block transfer
            MOV     #CCIE, &TACCTL0_                                            ; Enable Timer A to issue an interrupt when the timer counts to the value in TACCR0

acq_cpos:   BIT     #BIT7, R8                                                   ; Determine if Cartesian position data was requested
            JZ      acq_gpos
            MOV     #00007h, R14                                                ; Note the system is acquiring Cartesian position data
            MOV     #car_pos, &DMA0SA_                                          ; Set the DMA-0 source address to the location of the car_pos GRIL message
            MOV     R11, &DMA2DA_                                               ; Set the DMA-2 destination address to the value in R11
            MOV     #0000Bh, &DMA0SZ_                                           ; Configure DMA-0 to transfer eleven bytes of data
            MOV     #00023h, &DMA1SZ_                                           ; Configure DMA-1 to transfer thirty-five bytes of data
            MOV     #00018h, &DMA2SZ_                                           ; Configure DMA-2 to transfer twenty-four bytes of data
            BIS     #(DMAEN+DMAIE), &DMA0CTL_                                   ; Enable DMA-0 to issue an interrupt request
            BIS     #(DMAEN+DMAIE), &DMA1CTL_                                   ; Enable DMA-1 to issue an interrupt request
            BIS     #(DMAEN+DMAIE), &DMA2CTL_                                   ; Enable DMA-2 to issue an interrupt request

            BIS     #TACLR, &TACTL_                                             ; Clear the Timer A counter
            MOV     #00BB8h, &TACCR0_                                           ; Set the Timer A capture value to 3000 (1 ms)
            MOV     #(TASSEL_2+MC_1+TAIE), &TACTL_                              ; Select SMCLK for the Timer A clock, and start the timer in up mode

            BIS.B   #(UTXE0+URXE0), &ME1_                                       ; Enable the UART0 transmitter and receiver
            BIC.B   #SWRST, &U0CTL_                                             ; Release UART0 for operation
            ADD     #00018h, R11                                                ; Increase the data destination address by eighteen bytes
            ADD     #00018h, R4                                                 ; Add eighteen transmitted bytes to the "data status" subpacket
GPS_d0:     BIT     #(UTXE0+URXE0), &ME1_                                       ; Wait for the Cartesian position data to be acquired
            JNZ     GPS_d0

acq_gpos:   BIT     #BIT6, R8                                                   ; Determine if Geodetic position data was requested
            JZ      acq_cvel
            MOV     #00008h, R14                                                ; Note the system is acquiring Geodetic position data
            MOV     #geo_pos, &DMA0SA_                                          ; Set the DMA-0 source address to the location of the geo_pos GRIL message
            MOV     R11, &DMA2DA_                                               ; Set the DMA-2 destination address to the value in R11
            MOV     #0000Bh, &DMA0SZ_                                           ; Configure DMA-0 to transfer eleven bytes of data
            MOV     #00023h, &DMA1SZ_                                           ; Configure DMA-1 to transfer thirty-five bytes of data
            MOV     #00018h, &DMA2SZ_                                           ; Configure DMA-2 to transfer twenty-four bytes of data
            BIS     #(DMAEN+DMAIE), &DMA0CTL_                                   ; Enable DMA-0 to issue an interrupt request
            BIS     #(DMAEN+DMAIE), &DMA1CTL_                                   ; Enable DMA-1 to issue an interrupt request
            BIS     #(DMAEN+DMAIE), &DMA2CTL_                                   ; Enable DMA-2 to issue an interrupt request

            BIS     #TACLR, &TACTL_                                             ; Clear the Timer A counter
            MOV     #00BB8h, &TACCR0_                                           ; Set the Timer A capture value to 3000 (1 ms)
            MOV     #(TASSEL_2+MC_1+TAIE), &TACTL_                              ; Select SMCLK for the Timer A clock, and start the timer in up mode
            BIS.B     #(UTXE0+URXE0), &ME1_                           ; Enable the UART0 transmitter and receiver
            BIC.B     #SWRST, &U0CTL_                                 ; Release UART0 for operation
            ADD       #00018h, R11                                    ; Increase the data destination address by eighteen bytes
            ADD       #00018h, R4                                     ; Add eighteen transmitted bytes to the "data status" subpacket
GPS_d1:     BIT       #(UTXE0+URXE0), &ME1_                           ; Wait for the Geodetic position data to be acquired
            JNZ       GPS_d1

acq_cvel:   BIT       #BIT5, R8                                       ; Determine if Cartesian velocity data was requested
            JZ        acq_gvel
            MOV       #00009h, R14                                    ; Note the system is acquiring Cartesian velocity data
            MOV       #car_vel, &DMA0SA_                              ; Set the DMA-0 source address to the location of the car_vel GRIL message
            MOV       R11, &DMA2DA_                                   ; Set the DMA-2 destination address to the value in R11
            MOV       #0000Bh, &DMA0SZ_                               ; Configure DMA-0 to transfer eleven bytes of data
            MOV       #00017h, &DMA1SZ_                               ; Configure DMA-1 to transfer twenty-three bytes of data
            MOV       #0000Ch, &DMA2SZ_                               ; Configure DMA-2 to transfer twelve bytes of data
            BIS       #(DMAEN+DMAIE), &DMA0CTL_                       ; Enable DMA-0 to issue an interrupt request
            BIS       #(DMAEN+DMAIE), &DMA1CTL_                       ; Enable DMA-1 to issue an interrupt request
            BIS       #(DMAEN+DMAIE), &DMA2CTL_                       ; Enable DMA-2 to issue an interrupt request

            BIS       #TACLR, &TACTL_                                 ; Clear the Timer A counter
            MOV       #00BB8h, &TACCR0_                               ; Set the Timer A capture value to 3000 (1 ms)
            MOV       #(TASSEL_2+MC_1+TAIE), &TACTL_                  ; Select SMCLK for the Timer A clock, and start the timer in up mode

            BIS.B     #(UTXE0+URXE0), &ME1_                           ; Enable the UART0 transmitter and receiver
            BIC.B     #SWRST, &U0CTL_                                 ; Release UART0 for operation
            ADD       #0000Ch, R11                                    ; Increase the data destination address by twelve bytes
            ADD       #0000Ch, R4                                     ; Add twelve transmitted bytes to the "data status" subpacket
GPS_d2:     BIT       #(UTXE0+URXE0), &ME1_                           ; Wait for the Cartesian velocity data to be acquired
            JNZ       GPS_d2

acq_gvel:   BIT       #BIT4, R8                                       ; Determine if Geodetic velocity data was requested
            JZ        acq_dop
            MOV       #0000Ah, R14                                    ; Note the system is acquiring Geodetic velocity data
            MOV       #geo_vel, &DMA0SA_                              ; Set the DMA-0 source address to the location of the geo_vel GRIL message
            MOV       R11, &DMA2DA_                                   ; Set the DMA-2 destination address to the value in R11
            MOV       #0000Bh, &DMA0SZ_                               ; Configure DMA-0 to transfer eleven bytes of data
            MOV       #00017h, &DMA1SZ_                               ; Configure DMA-1 to transfer twenty-three bytes of data
            MOV       #0000Ch, &DMA2SZ_                               ; Configure DMA-2 to transfer twelve bytes of data
            BIS       #(DMAEN+DMAIE), &DMA0CTL_                       ; Enable DMA-0 to issue an interrupt request
            BIS       #(DMAEN+DMAIE), &DMA1CTL_                       ; Enable DMA-1 to issue an interrupt request
            BIS       #(DMAEN+DMAIE), &DMA2CTL_                       ; Enable DMA-2 to issue an interrupt request

            BIS       #TACLR, &TACTL_                                 ; Clear the Timer A counter
            MOV       #00BB8h, &TACCR0_                               ; Set the Timer A capture value to 3000 (1 ms)
            MOV       #(TASSEL_2+MC_1+TAIE), &TACTL_                  ; Select SMCLK for the Timer A clock, and start the timer in up mode

            BIS.B     #(UTXE0+URXE0), &ME1_                           ; Enable the UART0 transmitter and receiver
            BIC.B     #SWRST, &U0CTL_                                 ; Release UART0 for operation
            ADD       #0000Ch, R11                                    ; Increase the data destination address by twelve bytes
            ADD       #0000Ch, R4                                     ; Add twelve transmitted bytes to the "data status" subpacket
GPS_d3:     BIT       #(UTXE0+URXE0), &ME1_                           ; Wait for the Geodetic velocity data to be acquired
            JNZ       GPS_d3

acq_dop:    BIT       #(BIT7+BIT6+BIT5+BIT4), R8                      ; Determine if position or velocity data was requested (dilution of precision)
            JZ        acq_stats
            MOV       #0000Bh, R14                                    ; Note the system is acquiring dilution of precision data
            MOV       #DOP, &DMA0SA_                                  ; Set the DMA-0 source address to the location of the DOP GRIL message
            MOV       R11, &DMA2DA_                                   ; Set the DMA-2 destination address to the value in R11
            MOV       #0000Bh, &DMA0SZ_                               ; Configure DMA-0 to transfer eleven bytes of data
            MOV       #00013h, &DMA1SZ_                               ; Configure DMA-1 to transfer ninteen bytes of data
            MOV       #0000Ch, &DMA2SZ_                               ; Configure DMA-2 to transfer twelve bytes of data
            BIS       #(DMAEN+DMAIE), &DMA0CTL_                       ; Enable DMA-0 to issue an interrupt request
            BIS       #(DMAEN+DMAIE), &DMA1CTL_                       ; Enable DMA-1 to issue an interrupt request
            BIS       #(DMAEN+DMAIE), &DMA2CTL_                       ; Enable DMA-2 to issue an interrupt request

            BIS       #TACLR, &TACTL_                                 ; Clear the Timer A counter
            MOV       #00BB8h, &TACCR0_                               ; Set the Timer A capture value to 3000 (1 ms)
            MOV       #(TASSEL_2+MC_1+TAIE), &TACTL_                  ; Select SMCLK for the Timer A clock, and start the timer in up mode

            BIS.B     #(UTXE0+URXE0), &ME1_                           ; Enable the UART0 transmitter and receiver
            BIC.B     #SWRST, &U0CTL_                                 ; Release UART0 for operation
            ADD       #0000Ch, R11                                    ; Increase the data destination address by twelve bytes
            ADD       #0000Ch, R4                                     ; Add twelve transmitted bytes to the "data status" subpacket
GPS_d4:     BIT       #(UTXE0+URXE0), &ME1_                           ; Wait for the dilution of precision data to be acquired
            JNZ       GPS_d4

acq_stats:  BIT       #(BIT7+BIT6+BIT5+BIT4), R8                      ; Determine if position or velocity data was requested (satellite statistics)
            JZ        acq_date
            MOV       #0000Ch, R14                                    ; Note the system is acquiring satellite statistic data
            MOV       #pos_stats, &DMA0SA_                            ; Set the DMA-0 source address to the location of the pos_stats GRIL message
            MOV       R11, &DMA2DA_                                   ; Set the DMA-2 destination address to the value in R11
            MOV       #0000Bh, &DMA0SZ_                               ; Configure DMA-0 to transfer eleven bytes of data
            MOV       #0000Eh, &DMA1SZ_                               ; Configure DMA-1 to transfer fourteen bytes of data
            MOV       #00003h, &DMA2SZ_                               ; Configure DMA-2 to transfer three bytes of data
            BIS       #(DMAEN+DMAIE), &DMA0CTL_                       ; Enable DMA-0 to issue an interrupt request
            BIS       #(DMAEN+DMAIE), &DMA1CTL_                       ; Enable DMA-1 to issue an interrupt request
            BIS       #(DMAEN+DMAIE), &DMA2CTL_                       ; Enable DMA-2 to issue an interrupt request

            BIS       #TACLR, &TACTL_                                 ; Clear the Timer A counter
            MOV       #00BB8h, &TACCR0_                               ; Set the Timer A capture value to 3000 (1 ms)
            MOV       #(TASSEL_2+MC_1+TAIE), &TACTL_                  ; Select SMCLK for the Timer A clock, and start the timer in up mode
            BIS.B   #(UTXE0+URXE0), &ME1_                   ; Enable the UART0 transmitter and receiver
            BIC.B   #SWRST, &U0CTL_                         ; Release UART0 for operation
            ADD     #00003h, R11                            ; Increase the data destination address by three bytes
            ADD     #00003h, R4                             ; Add three transmitted bytes to the "data status" subpacket
GPS_d5:     BIT     #(UTXE0+URXE0), &ME1_                   ; Wait for the satellite statistic data to be acquired
            JNZ     GPS_d5

acq_date:   BIT     #BIT3, R8                               ; Determine if GPS receiver time was requested
            JZ      acq_bat
            MOV     #0000Dh, R14                            ; Note the system is acquiring the GPS receiver date
            MOV     #rec_date, &DMA0SA_                     ; Set the DMA-0 source address to the location of the rec_date GRIL message
            MOV     R11, &DMA2DA_                           ; Set the DMA-2 destination address to the value in R11
            MOV     #0000Bh, &DMA0SZ_                       ; Configure DMA-0 to transfer eleven bytes of data
            MOV     #0000Bh, &DMA1SZ_                       ; Configure DMA-1 to transfer eleven bytes of data
            MOV     #00005h, &DMA2SZ_                       ; Configure DMA-2 to transfer five bytes of data
            BIS     #(DMAEN+DMAIE), &DMA0CTL_               ; Enable DMA-0 to issue an interrupt request
            BIS     #(DMAEN+DMAIE), &DMA1CTL_               ; Enable DMA-1 to issue an interrupt request
            BIS     #(DMAEN+DMAIE), &DMA2CTL_               ; Enable DMA-2 to issue an interrupt request

            BIS     #TACLR, &TACTL_                         ; Clear the Timer A counter
            MOV     #00BB8h, &TACCR0_                       ; Set the Timer A capture value to 3000 (1 ms)
            MOV     #(TASSEL_2+MC_1+TAIE), &TACTL_          ; Select SMCLK for the Timer A clock, and start the timer in up mode

            BIS.B   #(UTXE0+URXE0), &ME1_                   ; Enable the UART0 transmitter and receiver
            BIC.B   #SWRST, &U0CTL_                         ; Release UART0 for operation
            ADD     #00005h, R11                            ; Increase the data destination address by five bytes
            ADD     #00005h, R4                             ; Add five transmitted bytes to the "data status" subpacket
GPS_d6:     BIT     #(UTXE0+URXE0), &ME1_                   ; Wait for the GPS date to be acquired
            JNZ     GPS_d6

acq_time:   MOV     #0000Eh, R14                            ; Note the system is acquiring the GPS receiver time
            MOV     #rec_time, &DMA0SA_                     ; Set the DMA-0 source address to the location of the rec_time GRIL message
            MOV     R11, &DMA2DA_                           ; Set the DMA-2 destination address to the value in R11
            MOV     #0000Bh, &DMA0SZ_                       ; Configure DMA-0 to transfer eleven bytes of data
            MOV     #0000Ah, &DMA1SZ_                       ; Configure DMA-1 to transfer ten bytes of data
            MOV     #00004h, &DMA2SZ_                       ; Configure DMA-2 to transfer four bytes of data
            BIS     #(DMAEN+DMAIE), &DMA0CTL_               ; Enable DMA-0 to issue an interrupt request
            BIS     #(DMAEN+DMAIE), &DMA1CTL_               ; Enable DMA-1 to issue an interrupt request
            BIS     #(DMAEN+DMAIE), &DMA2CTL_               ; Enable DMA-2 to issue an interrupt request

            BIS     #TACLR, &TACTL_                         ; Clear the Timer A counter
            MOV     #00BB8h, &TACCR0_                       ; Set the Timer A capture value to 3000 (1 ms)
            MOV     #(TASSEL_2+MC_1+TAIE), &TACTL_          ; Select SMCLK for the Timer A clock, and start the timer in up mode

            BIS.B   #(UTXE0+URXE0), &ME1_                   ; Enable the UART0 transmitter and receiver
            BIC.B   #SWRST, &U0CTL_                         ; Release UART0 for operation
            ADD     #00004h, R11                            ; Increase the data destination address by four bytes
            ADD     #00004h, R4                             ; Add four transmitted bytes to the "data status" subpacket
GPS_d7:     BIT     #(UTXE0+URXE0), &ME1_                   ; Wait for the GPS time to be acquired
            JNZ     GPS_d7

acq_bat:    MOV     #0000Fh, R14                            ; Note the system is acquiring the battery voltage
            MOV     #bat_volt, &DMA0SA_                     ; Set the DMA-0 source address to the location of the bat_volt GRIL message
            MOV     #02006h, &DMA2SA_                       ; Set the DMA-2 source address to the start of data in the GPS dump address
            MOV     R11, &DMA2DA_                           ; Set the DMA-2 destination address to the value in R11
            MOV     #0000Fh, &DMA0SZ_                       ; Configure DMA-0 to transfer fifteen bytes of data
            MOV     #0000Ah, &DMA1SZ_                       ; Configure DMA-1 to transfer ten bytes of data
            MOV     #00004h, &DMA2SZ_                       ; Configure DMA-2 to transfer four bytes of data
            BIS     #(DMAEN+DMAIE), &DMA0CTL_               ; Enable DMA-0 to issue an interrupt request
            BIS     #(DMAEN+DMAIE), &DMA1CTL_               ; Enable DMA-1 to issue an interrupt request
            BIS     #(DMAEN+DMAIE), &DMA2CTL_               ; Enable DMA-2 to issue an interrupt request

            BIS     #TACLR, &TACTL_                         ; Clear the Timer A counter
            MOV     #00BB8h, &TACCR0_                       ; Set the Timer A capture value to 3000 (1 ms)
            MOV     #(TASSEL_2+MC_1+TAIE), &TACTL_          ; Select SMCLK for the Timer A clock, and start the timer in up mode

            BIS.B   #(UTXE0+URXE0), &ME1_                   ; Enable the UART0 transmitter and receiver
            BIC.B   #SWRST, &U0CTL_                         ; Release UART0 for operation
            ADD     #00004h, R11                            ; Increase the data destination address by four bytes
            ADD     #00004h, R4                             ; Add four transmitted bytes to the "data status" subpacket
GPS_d8:     BIT     #(UTXE0+URXE0), &ME1_                   ; Wait for the battery voltage to be acquired
            JNZ     GPS_d8
clr_dapf:   BIC     #BIT6, R5                               ; Clear the data acquisition pending flag (indicates data was successfully acquired)

tx_data:
            MOV     #08000h, &01100h                        ; Move the ewm write command to the top of data acquisition RAM
            MOV     #00000h, &01102h
            MOV     R4, &01104h                             ; Move the "data status" subpacket to data acquisition RAM
            MOV     R5, 0(R11)                              ; Move the "module status" subpacket to the bottom of data acquisition RAM
            INCD    R11                                     ; Increase the data destination address by two bytes for the "module status" subpacket
            ADD     #00005h, R4                             ; Add five transmitted bytes for the ewm write message and the "data status" subpacket

            CMP     #00001h, R9                             ; Append the "data status" terminator if this is the last packet to transmit
            JNZ     set_dtpf
            CMP     #000001h, R10
            JNZ     set_dtpf
            BIC     #000FFh, R4                             ; Clear the number of bytes to transmit for the "data status" terminator
            MOV     R4, 0(R11)                              ; Move the "module status" terminator to the bottom of data acquisition RAM
            INCD    R11                                     ; Increase the data destination address by two bytes for the "data status" terminator
            INCD    R4                                      ; Add two transmitted bytes for the "data status" terminator
set_dtpf:   BIS     #BIT4, R5                               ; Set the data transmission pending flag (indicates data is currently being transmitted)

config_tx:  MOV     #01101h, &DMA0SA_                       ; Set the DMA-0 source address to the top of data acquisition RAM
```

```
            MOV       #U1TXBUF_, &DMA0DA_                                    ; Set the DMA-0 destination address to the SPI peripheral transmitter
            MOV.B     R4, &DMA0SZ_                                           ; Configure DMA-0 to transfer one entire packet of data
            MOV       #DMA0TSEL_10, DMACTL0_                                 ; Trigger DMA-0 when the SPI peripheral transmitter is available
            MOV       #(DMADT_1+DMASRCINCR_3+DMASBDB+DMAEN), &DMA0CTL_       ; Enable DMA-0 for (byte) block transfer

tx_start:   BIS.B     #USPIE1, &ME2_                                         ; Enable the SPI peripheral
            BIC.B     #BIT4, &P2OUT_                                         ; Start a data transmission frame
            BIC.B     #SWRST, &U1CTL_                                        ; Release the SPI peripheral for operation
tx_done:    BIT       #DMAEN, &DMA0CTL_                                      ; Wait for the DMA transfer of data to finish
            JNZ       tx_done
            BIT       #TXEPT, &U1TCTL_                                       ; Wait for the SPI transfer of data to finish
            JZ        tx_done
            BIS.B     #BIT4, &P2OUT_                                         ; End the data transmission frame
            BIS.B     #SWRST, &U1CTL_                                        ; Hold the SPI peripheral
            BIC.B     #USPIE1, &ME2_                                         ; Disable the SPI peripheral

clr_dtpf:   BIC       #BIT4, R5                                              ; Clear the data transmission pending flag (indicates data was successfully transmitted)
            BIC       #00003h, R13                                           ; Note the system is in LPM3
            BIS       #LPM3, SR                                              ; Enter LPM3 and wait for the data acquisition period to end

more_data:
            DEC       R10                                                    ; Decrement the data rate counter
            JNZ       jmp_TB                                                 ; If needed, go back and acquire more data
            CMP       #00000h, R7                                            ; Determine if acquiring data indefinetly (i.e., data_time = R7 = 0)
            JZ        jmp_lim_dr                                             ; If acquiring data forever, jump back and restore the data rate counter
            DEC       R9                                                     ; Decrement the length of data acquisition counter
            JNZ       jmp_lim_dr                                             ; If needed, jump back and restore the data rate counter
            BR        #defaults                                              ; Restore system defaults and wait for another message from the server
jmp_TB:     BR        #init_TB
jmp_lim_dr: BR        #lim_dr

            ENDMOD                                                           ; End of the main module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      NMI_ISR                                                ; Start of the NMI_ISR module

            ORG       0FFFCh                                                 ; Setup the non-maskable interrupt vector
            DW        NMI_ISR

            RSEG      CODE                                                   ; Make the ISR code relocatable
NMI_ISR:
            BIT.B     #OFIFG, &IFG1_                                         ; Check for an oscillator fault
            JNZ       OSC_FAULT
            BIT.B     #ACCVIFG, &FCTL3_                                      ; Check for a Flash access violation
            JNZ       FLASH_VIOL
            BIT.B     #NMIIFG, &IFG1_                                        ; Check for an edge on the RST/NMI pin
            JNZ       RST_NMI
            RETI                                                             ; Return from the interrupt if no remaining flags are set
OSC_FAULT:
            BIC.B     #OFIFG, &IFG1_                                         ; Reset the oscillator fault flag
            BIS       #BIT6, R5                                              ; Indicate an oscillator fault in the data status subpacket (R5, bit 2)
            BIC.B     #BIT0, &P6OUT_                                         ; Turn the oscillator fault LED on
            JMP       NMI_ISR                                                ; Check for remaining non-maskable interrupts
FLASH_VIOL:
            MOV       #04000h, PC                                            ; Force a PUC (system should never reach this point)
RST_NMI:
            MOV       #04000h, PC                                            ; Force a PUC (system should never reach this point)

            ENDMOD                                                           ; End of the NMI_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      TBH_ISR                                                ; Start of the TBH_ISR module (highest priority)

            ORG       0FFFAh                                                 ; Setup the Timer B interrupt vector
            DW        TBH_ISR

            RSEG      CODE                                                   ; Make the ISR code relocatable
TBH_ISR:
            BIC       #MC_1, &TBCTL_                                         ; Turn Timer B off to conserve power
            CMP       #00403h, R13                                           ; Determine if the system is also in LPM4
            JZ        ent_LPM4
            BIC       #LPM3, 0(SP)                                           ; Enter active power mode (i.e., exit LPM3)
            BIC       #00003h, R13                                           ; Note the system is no longer in LPM3
            RETI
ent_LPM4:   BIS       #OSCOFF, 0(SP)                                         ; Enter LPM4 (i.e., exit LPM3)
            BIC       #00003h, R13                                           ; Note the system is no longer in LPM3
            RETI

            ENDMOD                                                           ; End of the TBH_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      TBL_ISR                                                ; Start of the TBL_ISR module (lowest priority)

            ORG       0FFF8h                                                 ; Setup the Timer B interrupt vector
            DW        TBL_ISR

            RSEG      CODE                                                   ; Make the ISR code relocatable
TBL_ISR:
            MOV       #04000h, PC                                            ; Force a PUC (system should never reach this point)

            ENDMOD                                                           ; End of the TBL_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
            NAME      COMP_ISR                                         ; Start of the COMP_ISR module

            ORG       0FFF6h                                           ; Setup the comparator interrupt vector
            DW        COMP_ISR

            RSEG      CODE                                             ; Make the ISR code relocatable
COMP_ISR:
            MOV       #04000h, PC                                      ; Force a PUC (system should never reach this point)

            ENDMOD                                                     ; End of the COMP_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      WDT_ISR                                          ; Start of the WDT_ISR module

            ORG       0FFF4h                                           ; Setup the watchdog timer interrupt vector
            DW        WDT_ISR

            RSEG      CODE                                             ; Make the ISR code relocatable
WDT_ISR:
            MOV       #04000h, PC                                      ; Force a PUC (system should never reach this point)

            ENDMOD                                                     ; End of the WDT_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      URX0_ISR                                         ; Start of the URX0_ISR module

            ORG       0FFF2h                                           ; Setup the USART0 receive interrupt vector
            DW        URX0_ISR

            RSEG      CODE                                             ; Make the ISR code relocatable
URX0_ISR:
            MOV       #04000h, PC                                      ; Force a PUC (system should never reach this point)

            ENDMOD                                                     ; End of the URX0_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      UTX0_ISR                                         ; Start of the UTX0_ISR module

            ORG       0FFF0h                                           ; Setup the USART0 transmit interrupt vector
            DW        UTX0_ISR

            RSEG      CODE                                             ; Make the ISR code relocatable
UTX0_ISR:
            MOV       #04000h, PC                                      ; Force a PUC (system should never reach this point)

            ENDMOD                                                     ; End of the UTX0_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      ADC12_ISR                                        ; Start of the ADC12_ISR module

            ORG       0FFEEh                                           ; Setup the ADC12 interrupt vector
            DW        ADC12_ISR

            RSEG      CODE                                             ; Make the ISR code relocatable
ADC12_ISR:
            CLR       &ADC12CTL0_                                      ; Turn ADC12 off
            CLR       &ADC12IE_                                        ; Disable all ADC12 interrupts
            BIS       #Z, 0(SP)
            BIT       #BIT0, R12                                       ; Determine if DMA-0 needs to be triggered
            JNZ       EN_DMA0
            BIT       #BIT1, R12                                       ; Determine if DMA-1 needs to be triggered
            JNZ       EN_DMA1
            BIT       #BIT2, R12                                       ; Determine if DMA-2 needs to be triggered
            JNZ       EN_DMA2
            MOV       #04000h, PC                                      ; Force a PUC (system should never reach this point)
EN_DMA0:    BIS       #DMAREQ, &DMA0CTL_                               ; Trigger DMA-0
            RETI
EN_DMA1:    BIS       #DMAREQ, &DMA1CTL_                               ; Trigger DMA-1
            RETI
EN_DMA2:    BIS       #DMAREQ, &DMA2CTL_                               ; Trigger DMA-2
            RETI

            ENDMOD                                                     ; End of the ADC12_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      TAH_ISR                                          ; Start of the TAH_ISR module

            ORG       0FFECh                                           ; Setup the Timer A interrupt vector (highest priority)
            DW        TAH_ISR

            RSEG      CODE                                             ; Make the ISR code relocatable
TAH_ISR:
            BIC       #MC_1, &TACTL_                                   ; Turn Timer A off to conserve power
            BIT       #00003h, R13                                     ; Determine if Timer A is being used for GPS timeout
            JZ        TA_TOUT
            CMP       #00403h, R13                                     ; Determine if the system is also in LPM4
            JZ        ent_LPM4
            BIC       #LPM3, 0(SP)                                     ; Enter active power mode (i.e., exit LPM3)
            BIC       #00003h, R13                                     ; Note the system is no longer in LPM3
            RETI
ent_LPM4:   BIS       #OSCOFF, 0(SP)                                   ; Enter LPM4 (i.e., exit LPM3)
            BIC       #00003h, R13                                     ; Note the system is no longer in LPM3
            RETI
```

```
TA_TOUT:
            CLR.B     &02005h                                   ; Configure DMA-2 to transfer zeros instead of GPS data
            CLR       &02006h
            BIC       #DMASRCINCR_3, &DMA2CTL_                  ; Configure DMA-2 for (byte) block transfer of zeroes
            BIS       #DMAREQ, &DMA2CTL_                        ; Trigger DMA-2
            RETI

            ENDMOD                                              ; End of the TAH_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      TAL_ISR                                   ; Start of the TAL_ISR module

            ORG       0FFEAh                                    ; Setup the Timer A interrupt vector (lowest priority)
            DW        TAL_ISR

            RSEG      CODE                                      ; Make the ISR code relocatable
TAL_ISR:
            MOV       #04000h, PC                               ; Force a PUC (system should never reach this point)

            ENDMOD                                              ; End of the TAL_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      P1_ISR                                    ; Start of the P1_ISR module

            ORG       0FFE8h                                    ; Setup the port 1 interrupt vector
            DW        P1_ISR

            RSEG      CODE                                      ; Make the ISR code relocatable
P1_ISR:
            MOV       #04000h, PC                               ; Force a PUC (system should never reach this point)

            ENDMOD                                              ; End of the P1_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      URX1_ISR                                  ; Start of the URX1_ISR module

            ORG       0FFE6h                                    ; Setup the USART1 receive interrupt vector
            DW        URX1_ISR

            RSEG      CODE                                      ; Make the ISR code relocatable
URX1_ISR:
            MOV       #04000h, PC                               ; Force a PUC (system should never reach this point)

            ENDMOD                                              ; End of the URX1_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      UTX1_ISR                                  ; Start of the UTX1_ISR module

            ORG       0FFE4h                                    ; Setup the USART1 transmit interrupt vector
            DW        UTX1_ISR

            RSEG      CODE                                      ; Make the ISR code relocatable
UTX1_ISR:
            MOV       #04000h, PC                               ; Force a PUC (system should never reach this point)

            ENDMOD                                              ; End of the UTX1_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME      P2_ISR                                    ; Start of the P2_ISR module

            ORG       0FFE2h                                    ; Setup the port 2 interrupt vector
            DW        P2_ISR

            RSEG      CODE                                      ; Make the ISR code relocatable
P2_ISR:
            BIT.B     #BIT1, &P2IFG_                            ; Determine if P2.1 (ewm load status) issued an interrupt
            JNZ       P21_INT
            BIT.B     #BIT2, &P2IFG_                            ; Determine if P2.2 (ewm access point status) issued an interrupt
            JNZ       P22_INT
            BIT.B     #BIT3, &P2IFG_                            ; Determine if P2.3 (ewm IP connection status) issued an interrupt
            JNZ       P23_INT
            BIT.B     #BIT5, &P2IFG_                            ; Determine if P2.5 (ewm data available) issued an interrupt
            JNZ       P25_INT
            MOV       #04000h, PC                               ; Force a PUC (system should never reach this point)
P21_INT:
            BIC.B     #BIT1, &P2IFG_                            ; Clear the P2.1 interrupt flag
            BIT.B     #BIT1, &P2IES_                            ; Determine if the ewm just failed to load (high-to-low)
            JNZ       ewm_noLO
            BIS.B     #BIT1, &P2IES_                            ; If the ewm just loaded, trigger interrupt on high-to-low
            CMP       #00403h, R13                              ; Determine if the system is also in LPM3
            JZ        ent_LPM3
            JMP       ex_LPM4
ewm_noLO:   BIC.B     #BIT1, &P2IES_                            ; If the ewm just failed, trigger interrupt on low-to-high
            JMP       ent_LPM4
            RETI
P22_INT:
            BIC.B     #BIT2, &P2IFG_                            ; Clear the P2.2 interrupt flag
            BIT.B     #BIT2, &P2IES_                            ; Determine if the ewm just lost its access point connection (high-to-low)
            JNZ       ewm_noAP
            BIS.B     #BIT2, &P2IES_                            ; If the ewm just connected to an access point, trigger interrupt on high-to-low
            CMP       #00403h, R13                              ; Determine if the system is also in LPM3
            JZ        ent_LPM3
            JMP       ex_LPM4
```

```
ewm_noAP:  BIC.B    #BIT2, &P2IES_                                     ; If the ewm just lost its access point connection, trigger interrupt on low-to-high
           JMP      ent_LPM4
P23_INT:
           BIC.B    #BIT3, &P2IFG_                                     ; Clear the P2.3 interrupt flag
           BIT.B    #BIT3, &P2IES_                                     ; Determine if the ewm just lost its IP connection (high-to-low)
           JNZ      ewm_noIP
           BIS.B    #BIT3, &P2IES_                                     ; If the ewm just connected to an IP, trigger interrupt on high-to-low
           CMP      #00403h, R13                                       ; Determine if the system is also in LPM3
           JZ       ent_LPM3
           JMP      ex_LPM4
ewm_noIP:  BIC.B    #BIT3, &P2IES_                                     ; If the ewm just lost its IP connection, trigger interrupt on low-to-high
           JMP      ent_LPM4
P25_INT:
           BIC.B    #BIT5, &P2IFG_                                     ; Clear P2.5 interrupt flag
           PUSH     &DMACTL0_                                          ; Temporarily store the DMA-0 and DMA-1 settings
           PUSH     &DMA0CTL_
           PUSH     &DMA1CTL_
           PUSH     &DMA0SA_
           PUSH     &DMA0DA_
           PUSH     &DMA0SZ_
           PUSH     &DMA1SA_
           PUSH     &DMA1DA_
           PUSH     &DMA1SZ_

config_rx: CLR      R15                                                ; Move the ewm read command to R15
           MOV      R15, &DMA0SA_                                      ; Set the DMA-0 source address to R15 (ewm read command)
           MOV      #U1TXBUF_, &DMA0DA_                                ; Set the DMA-0 destination address to the SPI peripheral transmitter
           MOV      #00003h, &DMA1SZ_                                  ; Configure DMA-0 to transfer three bytes of data
           MOV      #U1RXBUF_, &DMA1SA_                                ; Set the DMA-1 source address to the SPI peripheral receiver
           MOV      #03001h, &DMA1DA_                                  ; Set the DMA-1 destination address to the data receive RAM
           MOV      #00009h, &DMA1SZ_                                  ; Configure DMA-1 to transfer one entire packet of data
           MOV      #(DMA0TSEL_10+DMA1TSEL_9), &DMACTL0_               ; Trigger DMA-0 and DMA-1 when the SPI peripheral is available
           MOV      #(DMADT_1+DMASBDB+DMAEN), &DMA0CTL_                ; Enable DMA-0 for (byte) block transfer
           MOV      #(DMADT_1+DMADSTINCR_3+DMASBDB+DMAEN), &DMA1CTL_   ; Enable DMA-1 for (byte) block transfer

rx_start:  BIS.B    #USPIE1, &ME2_                                     ; Enable the SPI peripheral
           BIC.B    #BIT4, &P2OUT_                                     ; Start a data transmission frame
           BIC.B    #SWRST, &U1CTL_                                    ; Release the SPI peripheral for operation
rx_done:   BIT      #DMAEN, &DMA1CTL_                                  ; Wait for the DMA transfer of data to finish
           JNZ      rx_done
           BIS.B    #BIT4, &P2OUT_                                     ; End the data transmission frame
           BIS.B    #SWRST, &U1CTL_                                    ; Hold the SPI peripheral
           BIC.B    #USPIE1, &ME2_                                     ; Disable the SPI peripheral

           CMP.B    #00006h, &03003h                                   ; Check for the correct number of transmitted bits
           JNZ      rst_DMA
           CMP.B    #module_id, &03004h                                ; Check for the correct module ID number
           JNZ      rst_DMA
           MOV.B    &03005h, R6                                        ; Store the requested data rate in R6
           MOV      &03006h, R7                                        ; Store the requested length of data acquisition in R7
           MOV      &03008h, R8                                        ; Store the requested "what data" in R8
rst_DMA:   POP      &DMA1SZ_                                           ; Restore the DMA-0 and DMA-1 settings
           POP      &DMA1DA_
           POP      &DMA1SA_
           POP      &DMA0SZ_
           POP      &DMA0DA_
           POP      &DMA0SA_
           POP      &DMA1CTL_
           POP      &DMA0CTL_
           POP      &DMACTL0_
           BIC      #LPM3, 0(SP)                                       ; Enter active power mode (i.e., exit LPM3)
           BIC      #00003h, R13                                       ; Note the system is no longer in LPM3
           RETI
ent_LPM3:  BIC      #OSCOFF, 0(SP)                                     ; Enter LPM3 (i.e., exit LPM4)
           BIC      #00400h, R13                                       ; Note the system is no longer in LPM4
           RETI
ent_LPM4:  BIS      #LPM4, 0(SP)                                       ; Enter LPM4
           BIS      #00400h, R13                                       ; Note the system is in LPM4
           RETI
ex_LPM4:   BIC      #LPM4, 0(SP)                                       ; Enter active power mode (i.e., exit LPM4)
           BIC      #00400h, R13                                       ; Note the system is no longer in LPM4
           RETI

           ENDMOD                                                      ; End of the P2_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
           NAME     DMA_ISR                                            ; Start of the DMA_ISR module

           ORG      0FFE0h                                             ; Setup the DMA interrupt vector
           DW       DMA_ISR

           RSEG     CODE                                               ; Make the ISR code relocatable
DMA_ISR:
           BIT      #DMAIFG, &DMA0CTL_                                 ; Determine if DMA-0 issued an interrupt
           JNZ      DMA0_ISR
           BIT      #DMAIFG, &DMA1CTL_                                 ; Determine if DMA-1 issued an interrupt
           JNZ      DMA1_ISR
           BIT      #DMAIFG, &DMA2CTL_                                 ; Determine if DMA-2 issued an interrupt
           JNZ      DMA2_ISR
           MOV      #04000h, PC                                        ; Force a PUC (system should never reach this point)
DMA0_ISR:
           BIC      #DMAIFG, &DMA0CTL_                                 ; Clear the DMA-0 interrupt flag
           BIT.B    #UTXIFG0, &IFG1_                                   ; Wait for the UART0 transmit buffer to be available
```

```
            JZ       DMA0_ISR
            MOV.B    #LF, &U0TXBUF_                          ; Transmit the line-feed character
            RETI
DMA1_ISR:
            BIC      #DMAIFG, &DMA1CTL_                      ; Clear the DMA-1 interrupt flag
            CMP      #00007h, R14                            ; Determine if currently acquiring Cartesian position
            JZ       acq_cpos
            CMP      #00008h, R14                            ; Determine if currently acquiring Geodetic position
            JZ       acq_gpos
            CMP      #00009h, R14                            ; Determine if currently acquiring Cartesian velocity
            JZ       acq_cvel
            CMP      #0000Ah, R14                            ; Determine if currently acquiring Geodetic velocity
            JZ       acq_gvel
            CMP      #0000Bh, R14                            ; Determine if currently acquiring dilution of precision
            JZ       acq_dop
            CMP      #0000Ch, R14                            ; Determine if currently acquiring satellite statistics data
            JZ       acq_sats
            CMP      #0000Dh, R14                            ; Determine if currently acquiring GPS receiver date
            JZ       acq_rdate
            CMP      #0000Eh, R14                            ; Determine if currently acquiring GPS receiver time
            JZ       acq_rtime
            CMP      #0000Fh, R14                            ; Determine if currently acquiring the battery voltage
            JZ       acq_vbat
acq_cpos:   CMP      #04F50h, &02000h                        ; Check the message ID number (PO)
            JZ       gps_crct
            RETI
acq_gpos:   CMP      #04750h, &02000h                        ; Check the message ID number (PG)
            JZ       gps_crct
            RETI
acq_cvel:   CMP      #04556h, &02000h                        ; Check the message ID number (VE)
            JZ       gps_crct
            RETI
acq_gvel:   CMP      #04756h, &02000h                        ; Check the message ID number (VG)
            JZ       gps_crct
            RETI
acq_dop:    CMP      #05044h, &02000h                        ; Check the message ID number (DP)
            JZ       gps_crct
            RETI
acq_sats:   CMP      #05350h, &02000h                        ; Check the message ID number (PS)
            JZ       gps_crct
            RETI
acq_rdate:  CMP      #04452h, &02000h                        ; Check the message ID number (RD)
            JZ       gps_crct
            RETI
acq_rtime:  CMP      #07E7Eh, &02000h                        ; Check the message ID number (~~)
            JZ       gps_crct
            RETI
acq_vbat:   CMP      #04552h, &02000h                        ; Check the message ID number (RE)
            JNZ      gps_error
            CMP.B    #00035h, &02004h                        ; Determine if the battery voltage is four characters long
            JNZ      vbat_3ch
            JMP      gps_crct
vbat_3ch:   MOV.B    #00030h, &02004h                        ; If the battery voltage is three characters long, fix DMA-2
            DEC      &DMA2SA_                                ; Decrease the DMA-2 source address to account for one less character
            JMP      gps_crct
gps_crct:   BIS      #DMAREQ, &DMA2CTL_                      ; Trigger DMA-2
            RETI
gps_error:  RETI
DMA2_ISR:
            BIC      #DMAIFG, &DMA2CTL_                      ; Clear the DMA-2 interrupt flag
            BIC      #MC_1, &TACTL_                          ; Turn Timer A off to conserve power
            BIS.B    #SWRST, &U0CTL_                         ; Hold UART0
            BIC.B    #(UTXE0+URXE0), &ME1_                   ; Disable the UART0 receiver and transmitter
            RETI

            ENDMOD                                           ; End of the DMA_ISR module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            NAME     prd_lkup                                ; Start of the prd_lkup module

            ORG      0B000h                                  ; Setup the period lookup table in Flash memory
            DW       32768                                   ; Clock ticks for a data rate of 1 Hz
            DW       16384                                   ; Clock ticks for a data rate of 2 Hz
            DW       10923                                   ; Clock ticks for a data rate of 3 Hz
            DW       8192                                    ; Clock ticks for a data rate of 4 Hz
            DW       6554                                    ; Clock ticks for a data rate of 5 Hz
            DW       5461                                    ; Clock ticks for a data rate of 6 Hz
            DW       4681                                    ; Clock ticks for a data rate of 7 Hz
            DW       4096                                    ; Clock ticks for a data rate of 8 Hz
            DW       3641                                    ; Clock ticks for a data rate of 9 Hz
            DW       3277                                    ; Clock ticks for a data rate of 10 Hz
            DW       2979                                    ; Clock ticks for a data rate of 11 Hz
            DW       2731                                    ; Clock ticks for a data rate of 12 Hz
            DW       2521                                    ; Clock ticks for a data rate of 13 Hz
            DW       2341                                    ; Clock ticks for a data rate of 14 Hz
            DW       2185                                    ; Clock ticks for a data rate of 15 Hz
            DW       2048                                    ; Clock ticks for a data rate of 16 Hz
            DW       1928                                    ; Clock ticks for a data rate of 17 Hz
            DW       1820                                    ; Clock ticks for a data rate of 18 Hz
            DW       1725                                    ; Clock ticks for a data rate of 19 Hz
            DW       1638                                    ; Clock ticks for a data rate of 20 Hz
            DW       1560                                    ; Clock ticks for a data rate of 21 Hz
            DW       1489                                    ; Clock ticks for a data rate of 22 Hz
```

```
        DW      1425                                            ; Clock ticks for a data rate of 23 Hz
        DW      1365                                            ; Clock ticks for a data rate of 24 Hz
        DW      1311                                            ; Clock ticks for a data rate of 25 Hz
        DW      1260                                            ; Clock ticks for a data rate of 26 Hz
        DW      1214                                            ; Clock ticks for a data rate of 27 Hz
        DW      1170                                            ; Clock ticks for a data rate of 28 Hz
        DW      1130                                            ; Clock ticks for a data rate of 29 Hz
        DW      1092                                            ; Clock ticks for a data rate of 30 Hz
        DW      1057                                            ; Clock ticks for a data rate of 31 Hz
        DW      1024                                            ; Clock ticks for a data rate of 32 Hz
        DW      993                                             ; Clock ticks for a data rate of 33 Hz
        DW      964                                             ; Clock ticks for a data rate of 34 Hz
        DW      936                                             ; Clock ticks for a data rate of 35 Hz
        DW      910                                             ; Clock ticks for a data rate of 36 Hz
        DW      886                                             ; Clock ticks for a data rate of 37 Hz
        DW      862                                             ; Clock ticks for a data rate of 38 Hz
        DW      840                                             ; Clock ticks for a data rate of 39 Hz
        DW      819                                             ; Clock ticks for a data rate of 40 Hz
        DW      799                                             ; Clock ticks for a data rate of 41 Hz
        DW      780                                             ; Clock ticks for a data rate of 42 Hz
        DW      762                                             ; Clock ticks for a data rate of 43 Hz
        DW      745                                             ; Clock ticks for a data rate of 44 Hz
        DW      728                                             ; Clock ticks for a data rate of 45 Hz
        DW      712                                             ; Clock ticks for a data rate of 46 Hz
        DW      697                                             ; Clock ticks for a data rate of 47 Hz
        DW      683                                             ; Clock ticks for a data rate of 48 Hz
        DW      669                                             ; Clock ticks for a data rate of 49 Hz
        DW      655                                             ; Clock ticks for a data rate of 50 Hz
        DW      643                                             ; Clock ticks for a data rate of 51 Hz
        DW      630                                             ; Clock ticks for a data rate of 52 Hz
        DW      618                                             ; Clock ticks for a data rate of 53 Hz
        DW      607                                             ; Clock ticks for a data rate of 54 Hz
        DW      596                                             ; Clock ticks for a data rate of 55 Hz
        DW      585                                             ; Clock ticks for a data rate of 56 Hz
        DW      575                                             ; Clock ticks for a data rate of 57 Hz
        DW      565                                             ; Clock ticks for a data rate of 58 Hz
        DW      555                                             ; Clock ticks for a data rate of 59 Hz
        DW      546                                             ; Clock ticks for a data rate of 60 Hz
        DW      537                                             ; Clock ticks for a data rate of 61 Hz
        DW      529                                             ; Clock ticks for a data rate of 62 Hz
        DW      520                                             ; Clock ticks for a data rate of 63 Hz
        DW      512                                             ; Clock ticks for a data rate of 64 Hz
        DW      504                                             ; Clock ticks for a data rate of 65 Hz
        DW      496                                             ; Clock ticks for a data rate of 66 Hz
        DW      489                                             ; Clock ticks for a data rate of 67 Hz
        DW      482                                             ; Clock ticks for a data rate of 68 Hz
        DW      475                                             ; Clock ticks for a data rate of 69 Hz
        DW      468                                             ; Clock ticks for a data rate of 70 Hz
        DW      462                                             ; Clock ticks for a data rate of 71 Hz
        DW      455                                             ; Clock ticks for a data rate of 72 Hz
        DW      449                                             ; Clock ticks for a data rate of 73 Hz
        DW      443                                             ; Clock ticks for a data rate of 74 Hz
        DW      437                                             ; Clock ticks for a data rate of 75 Hz
        DW      431                                             ; Clock ticks for a data rate of 76 Hz
        DW      426                                             ; Clock ticks for a data rate of 77 Hz
        DW      420                                             ; Clock ticks for a data rate of 78 Hz
        DW      415                                             ; Clock ticks for a data rate of 79 Hz
        DW      410                                             ; Clock ticks for a data rate of 80 Hz
        DW      405                                             ; Clock ticks for a data rate of 81 Hz
        DW      400                                             ; Clock ticks for a data rate of 82 Hz
        DW      395                                             ; Clock ticks for a data rate of 83 Hz
        DW      390                                             ; Clock ticks for a data rate of 84 Hz
        DW      386                                             ; Clock ticks for a data rate of 85 Hz
        DW      381                                             ; Clock ticks for a data rate of 86 Hz
        DW      377                                             ; Clock ticks for a data rate of 87 Hz
        DW      372                                             ; Clock ticks for a data rate of 88 Hz
        DW      368                                             ; Clock ticks for a data rate of 89 Hz
        DW      364                                             ; Clock ticks for a data rate of 90 Hz
        DW      360                                             ; Clock ticks for a data rate of 91 Hz
        DW      356                                             ; Clock ticks for a data rate of 92 Hz
        DW      352                                             ; Clock ticks for a data rate of 93 Hz
        DW      349                                             ; Clock ticks for a data rate of 94 Hz
        DW      345                                             ; Clock ticks for a data rate of 95 Hz
        DW      341                                             ; Clock ticks for a data rate of 96 Hz
        DW      338                                             ; Clock ticks for a data rate of 97 Hz
        DW      334                                             ; Clock ticks for a data rate of 98 Hz
        DW      331                                             ; Clock ticks for a data rate of 99 Hz
        DW      328                                             ; Clock ticks for a data rate of 100 Hz

        ENDMOD                                                  ; End of the prd_lkup module

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        END                                                     ; End of MCETS firmware
```

Appendix F: MCETS Server Software

F.1. MCETS Main Figure Functions (MATLAB Language)

```
% MCETS_Main   M-file for MCETS_Main.fig
%   Displays the main window for the MCETS Visualization Tool. This
%   includes the module selection launcher, sensor select options, and
%   visualization launcher for MCETS. Also creates a file 'module_list.txt'
%   in a user-specified directory (prompted at startup) to store the saved
%   module list.
%
%   USAGE:
%   MCETS_Main      Opens a new MCETS Visualization Tool window and creates
%                   'module_list.txt' in the user-specified directory
%
%   DEPENDENCIES:
%   MCETS_Main_Callback
%   Error_Message
%
%   See also MCETS_Main_Callback, Error_Message

% =========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% -------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% -------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 10/1/07, commented other possible object
%               handle assignments for future use.
% -------------------------------------------------------------------------
function varargout = MCETS_Main(varargin)

%% Load the MCETS_Main figure (MCETS_Main.fig)
fig_def = load('-mat', 'MCETS_Main.fig');
fig_def_names = fieldnames(fig_def);
fig_def = fig_def.(fig_def_names{1});
clear('fig_def_names');

%% Opens and initializes the MCETS_Main figure
fhand_0 = struct2handle(fig_def, 0);
fig_settings.Name = 'MCETS Visualization Tool';
% fig_settings.Position = [103.8 31 150 30];
fig_settings.Resize = 'off';
fig_settings.Tag = 'fig_1';
set(fhand_0, fig_settings);
clear('fig_def', 'fig_settings');
movegui(fhand_0, 'center');

%% Get handles to necessary GUI fields
% % Module listbox
% moduleListbox = findobj(fhand_0, 'Tag', 'moduleListbox');

% % Module pushbuttons
% loadPushbutton = findobj(fhand_0, 'Tag', 'loadPushbutton');
% editPushbutton = findobj(fhand_0, 'Tag', 'editPushbutton');
% removePushbutton = findobj(fhand_0, 'Tag', 'removePushbutton');

% % Sensor Selection checkboxes
% tempCheckbox = findobj(fhand_0, 'Tag', 'tempCheckbox');
% pressCheckbox = findobj(fhand_0, 'Tag', 'pressCheckbox');
% % gpsCheckbox = findobj(fhand_0, 'Tag', 'gpsCheckbox');
% accelCheckbox = findobj(fhand_0, 'Tag', 'accelCheckbox');
% gyroCheckbox = findobj(fhand_0, 'Tag', 'gyroCheckbox');
% magnCheckbox = findobj(fhand_0, 'Tag', 'magnCheckbox');
% cartposCheckbox = findobj(fhand_0, 'Tag', 'cartposCheckbox');
% geoposCheckbox = findobj(fhand_0, 'Tag', 'geoposCheckbox');
% cartvelCheckbox = findobj(fhand_0, 'Tag', 'cartvelCheckbox');
% geovelCheckbox = findobj(fhand_0, 'Tag', 'geovelCheckbox');
% timeCheckbox = findobj(fhand_0, 'Tag', 'timeCheckbox');

% % Data Acquisition sliders
% rateSlider = findobj(fhand_0, 'Tag', 'rateSlider');
% durationSlider = findobj(fhand_0, 'Tag', 'durationSlider');

% % Data Acquisition edits
% rateEdit = findobj(fhand_0, 'Tag', 'rateEdit');
% durationEdit = findobj(fhand_0, 'Tag', 'durationEdit');

% % Visualization popupmenus
% networkVisPopupmenu = findobj(fhand_0, 'Tag', 'networkVisPopupmenu');
% moduleVisPopupmenu = findobj(fhand_0, 'Tag', 'moduleVisPopupmenu');
% moduleSelPopupmenu = findobj(fhand_0, 'Tag', 'moduleSelPopupmenu');

% % Visualization Launcher pushbuttons
% networkPushbutton = findobj(fhand_0, 'Tag', 'networkPushbutton');
% modulePushbutton = findobj(fhand_0, 'Tag', 'modulePushbutton');

% Picture axes
logoAxes = findobj(fhand_0, 'Tag', 'logoAxes');
% netVisAxes = findobj(fhand_0, 'Tag', 'netVisAxes');
% moduleVisAxes = findobj(fhand_0, 'Tag', 'moduleVisAxes');

% % Status text
% recText = findobj(fhand_0, 'Tag', 'recText');
% liveText = findobj(fhand_0, 'Tag', 'liveText');

% % Record Data checkbox
% recordCheckbox = findobj(fhand_0, 'Tag', 'recordCheckbox');

% % Acquire/Playback/Stop pushbuttons
% acquirePushbutton = findobj(fhand_0, 'Tag', 'acquirePushbutton');
% playbackPushbutton = findobj(fhand_0, 'Tag', 'playbackPushbutton');
% stopPushbutton = findobj(fhand_0, 'Tag', 'stopPushbutton');

% Menu items
new = findobj(fhand_0, 'Tag', 'new');
% exit = findobj(fhand_0, 'Tag', 'exit');
% load_modules = findobj(fhand_0, 'Tag', 'load_modules');
% save_modules = findobj(fhand_0, 'Tag', 'save_modules');
% edit_modules = findobj(fhand_0, 'Tag', 'edit_modules');
% remove_selected = findobj(fhand_0, 'Tag', 'remove_selected');
% load_data = findobj(fhand_0, 'Tag', 'load_data');
% set_recording_location = findobj(fhand_0, 'Tag', 'set_recording_location');
% acquire = findobj(fhand_0, 'Tag', 'acquire');
% playback = findobj(fhand_0, 'Tag', 'playback');
% stop = findobj(fhand_0, 'Tag', 'stop');
% mcets_help = findobj(fhand_0, 'Tag', 'mcets_help');
% about_mcets = findobj(fhand_0, 'Tag', 'about_mcets');


%% Draws LL Logo
logo_image = imread('logo.bmp', 'BMP');
image(logo_image, 'Parent', logoAxes);
axis(logoAxes, 'off')
axis(logoAxes, 'image')

%% Determines working directory of the MCETS Visualization Tool
% Check if any immediate children of the current directory, or the current
% directory itself is named "MCETS"
cur_dir_name = strtok(fliplr(cd), '\');
cur_dir_name = fliplr(cur_dir_name);
cur_children = cellstr(ls(cd));
if any(strcmp('MCETS', cur_children)) % if any of the current directory's children↙
are named "MCETS"
    working_dir = [cd '\MCETS'];
elseif strcmp('MCETS', cur_dir_name) % if the current directory itself is named↙
"MCETS"
    working_dir = cd;
else
    selected_dir = uigetdir(cd, 'Choose location of MCETS working directory...'); %↙
```

```
user is prompted to select working directory.
    if selected_dir ~= 0 % if user selects a directory
        sel_dir_name = strtok(fliplr(selected_dir), '\');
        sel_dir_name = fliplr(sel_dir_name);
        sel_children = cellstr(ls(selected_dir));
        if any(strcmp('MCETS', sel_children)) % if any of the selected directory's children are named "MCETS"
            working_dir = [selected_dir '\MCETS'];
        elseif strcmp('MCETS', sel_dir_name) % if the selected directory itself is named "MCETS"
            working_dir = selected_dir;
        else % otherwise, try to create a directory named "MCETS" in the selected directory
            [success,msg,msg_ID] = mkdir(selected_dir,'MCETS');
            if success
                working_dir = [selected_dir '\MCETS'];
            else
                Error_Message(['Could not create MCETS directory. ' msg])
            end
        end
    else % if user cancels the prompt
        working_dir = '';
    end
end
clear('cur_dir_name','cur_children','selected_dir','sel_dir_name');
clear('sel_children','success','msg','msg_ID');

%% Initializes 'module_list.txt' and MCETS_Main figure data.
data.name_list = cellstr({});          % Stores a cell array of modules' names
data.IP_list = cellstr({});            % Stores a cell array of modules' IP addresses
data.rec_loc = '';                     % Stores the directory to which data will be recorded
data.connection_list = [];             % Stores an instrument object array containing all connections to modules
data.acquiring_flag = false;           % Stores a flag that is true when data is being acquired and false otherwise
data.working_dir = working_dir;        % Stores the directory that MCETS will consider its home
data.acq_duration = 0;                 % Stores the duration selected for acquisition
data.loaded_data = {};                 % Stores the full paths of all loaded data files
data.open_visuals = {};                % Stores handles and callbacks to all currently open visualization figures

if ~strcmp(data.working_dir, '') % Save location specified
    file_id = fopen([data.working_dir '\module_list.txt'], 'rt');
    if ~(file_id + 1)
        file_id = fopen([data.working_dir '\module_list.txt'], 'wt+');
    end
    module_line = fgetl(file_id);
    module_number = 1;
    while module_line ~= -1
        [IP_address, module_name] = strtok(char(module_line), sprintf('\t'));
        module_name = module_name(2:end);
        data.IP_list(module_number) = cellstr(IP_address);
        data.name_list(module_number) = cellstr(module_name);
        module_line = fgetl(file_id);
        module_number = module_number + 1;
    end
    status = fclose(file_id);
    if ~(status + 1)
        Error_Message('Could not close file: ''module_list.txt''.')
        return
    end

    set(fhand_0, 'UserData', data);
    handles.hObject = new;
    handles.fhand = fhand_0;
    MCETS_Main_Callback(handles);

else % Save location not specified
    disp('No working directory specified.')

    set(fhand_0, 'UserData', data);
    handles.hObject = new;
    handles.fhand = fhand_0;
    MCETS_Main_Callback(handles);

end
```

```
% MCETS_Main_Callback   Callback M-file for MCETS_Main.fig
%
%   USAGE:
%   MCETS_Main_Callback             Evaluates callback for the calling
%                                   object.
%
%   MCETS_Main_Callback(DATA, TAG)  Evaluates callback for object with Tag
%                                   TAG. Also passes DATA as figure's
%                                   UserData. DATA must contain a
%                                   DATA.main_hand field identifying to
%                                   which figure UserData will be saved
%                                   following 'loadPushbutton' callback.
%
%   DEPENDENCIES:
%   Edit_Modules
%   About_MCETS
%   Error_Message
%
%   See also MCETS_Main, Edit_Modules, About_MCETS, Error_Message

% =========================================================================
%         Multi-Client Embedded Telemetry System (MCETS) Project
% -------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% -------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 10/10/07, added playbackPushbutton
%               functionality.
% -------------------------------------------------------------------------
function MCETS_Main_Callback(varargin)

%% Retrieve the calling object's parent and its handle
[hObject fhand_0] = gcbo;
if (isempty(hObject) && (nargin == 1))
    hObject = varargin{1}.hObject;
end
if (isempty(fhand_0) && (nargin == 1))
    fhand_0 = varargin{1}.fhand;
end

%% Retrieve the calling object's tag and the figure's stored data
if nargin == 2
    if isstruct(varargin{1})
        data = varargin{1};
    else % this only occurs if MCETS_Main_Callback is called from tcpip_cbk
        fhand_0 = varargin{1};
        data = get(fhand_0, 'UserData');
    end
    tag = varargin{2};
else
    tag = get(hObject, 'Tag');
    data = get(fhand_0, 'UserData');
end
checkboxes = {...
    'tempCheckbox';...
    'pressCheckbox';...
    'accelCheckbox';...
    'gyroCheckbox';...
    'magnCheckbox';...
    'cartposCheckbox';...
    'geoposCheckbox';...
    'cartvelCheckbox';...
    'geovelCheckbox';...
    'timeCheckbox'};
if any(strcmp(checkboxes, tag))
```

```
    tag = 'sens_checkbox';
end
clear('checkboxes');

% disp(tag) % for debugging

%% Get handles to necessary GUI fields
% Module listbox
% moduleListbox = findobj(fhand_0, 'Tag', 'moduleListbox');
moduleListbox = findobj('Tag', 'moduleListbox');

% Module pushbuttons
loadPushbutton = findobj(fhand_0, 'Tag', 'loadPushbutton');
editPushbutton = findobj(fhand_0, 'Tag', 'editPushbutton');
removePushbutton = findobj(fhand_0, 'Tag', 'removePushbutton');

% Sensor Selection checkboxes
tempCheckbox = findobj(fhand_0, 'Tag', 'tempCheckbox');
pressCheckbox = findobj(fhand_0, 'Tag', 'pressCheckbox');
% gpsCheckbox = findobj(fhand_0, 'Tag', 'gpsCheckbox');
accelCheckbox = findobj(fhand_0, 'Tag', 'accelCheckbox');
gyroCheckbox = findobj(fhand_0, 'Tag', 'gyroCheckbox');
magnCheckbox = findobj(fhand_0, 'Tag', 'magnCheckbox');
cartposCheckbox = findobj(fhand_0, 'Tag', 'cartposCheckbox');
geoposCheckbox = findobj(fhand_0, 'Tag', 'geoposCheckbox');
cartvelCheckbox = findobj(fhand_0, 'Tag', 'cartvelCheckbox');
geovelCheckbox = findobj(fhand_0, 'Tag', 'geovelCheckbox');
timeCheckbox = findobj(fhand_0, 'Tag', 'timeCheckbox');

% Data Acquisition sliders
rateSlider = findobj(fhand_0, 'Tag', 'rateSlider');
durationSlider = findobj(fhand_0, 'Tag', 'durationSlider');

% Data Acquisition edits
rateEdit = findobj(fhand_0, 'Tag', 'rateEdit');
durationEdit = findobj(fhand_0, 'Tag', 'durationEdit');

% Visualization popupmenus
networkVisPopupmenu = findobj(fhand_0, 'Tag', 'networkVisPopupmenu');
moduleVisPopupmenu = findobj(fhand_0, 'Tag', 'moduleVisPopupmenu');
moduleSelPopupmenu = findobj(fhand_0, 'Tag', 'moduleSelPopupmenu');

% Visualization Launcher pushbuttons
networkPushbutton = findobj(fhand_0, 'Tag', 'networkPushbutton');
modulePushbutton = findobj(fhand_0, 'Tag', 'modulePushbutton');

% Picture axes
netVisAxes = findobj(fhand_0, 'Tag', 'netVisAxes');
moduleVisAxes = findobj(fhand_0, 'Tag', 'moduleVisAxes');

% Status text
recText = findobj(fhand_0, 'Tag', 'recText');
liveText = findobj(fhand_0, 'Tag', 'liveText');
playbackText = findobj(fhand_0, 'Tag', 'playbackText');
standbyText = findobj(fhand_0, 'Tag', 'standbyText');

% Record Data checkbox
recordCheckbox = findobj(fhand_0, 'Tag', 'recordCheckbox');

% Acquire/Playback/Stop pushbuttons
acquirePushbutton = findobj(fhand_0, 'Tag', 'acquirePushbutton');
playbackPushbutton = findobj(fhand_0, 'Tag', 'playbackPushbutton');
stopPushbutton = findobj(fhand_0, 'Tag', 'stopPushbutton');

% Menu items
module_menu = findobj(fhand_0, 'Tag', 'module_menu');
data_menu = findobj(fhand_0, 'Tag', 'data_menu');
new = findobj(fhand_0, 'Tag', 'new');
save_to = findobj(fhand_0, 'Tag', 'save_to');
load_modules = findobj(fhand_0, 'Tag', 'load_modules');
save_modules = findobj(fhand_0, 'Tag', 'save_modules');
```

```
edit_modules = findobj(fhand_0, 'Tag', 'edit_modules');
remove_selected = findobj(fhand_0, 'Tag', 'remove_selected');
load_data = findobj(fhand_0, 'Tag', 'load_data');
set_recording_location = findobj(fhand_0, 'Tag', 'set_recording_location');
convert_data = findobj(fhand_0, 'Tag', 'convert_data');
acquire = findobj(fhand_0, 'Tag', 'acquire');
playback = findobj(fhand_0, 'Tag', 'playback');
stop = findobj(fhand_0, 'Tag', 'stop');

%% Main callback switch
switch tag
%% File Menu Items
    case 'new'
        set(moduleListbox, 'String', []);
        data.IP_list = {};
        data.name_list = {};
        set(fhand_0, 'UserData', data);
        set(tempCheckbox, 'Value', 0);
        set(pressCheckbox, 'Value', 0);
%         set(gpsCheckbox, 'Value', 0);
        set(cartposCheckbox, 'Value', 0);
        set(geoposCheckbox, 'Value', 0);
        set(cartvelCheckbox, 'Value', 0);
        set(geovelCheckbox, 'Value', 0);
        set(timeCheckbox, 'Value', 0);
        set(accelCheckbox, 'Value', 0);
        set(gyroCheckbox, 'Value', 0);
        set(magnCheckbox, 'Value', 0);
        set(rateSlider, 'Value', 0);
        set(durationSlider, 'Value', 0);
        set(rateEdit, 'String', 1);
        set(durationEdit, 'String', 1);
        set(networkVisPopupmenu, 'Value', 1);
        set(moduleVisPopupmenu, 'Value', 1);
        set(moduleSelPopupmenu, 'Value', 1);
        set(recordCheckbox, 'Value', 0);
        set(moduleListbox, 'Value', []);
        menu_items = java_array('java.lang.String', 2);
        menu_items(1) = java.lang.String('Select Module');
        menu_items(2) = java.lang.String('---');
        menu_items_cell = cell(menu_items);
        set(moduleSelPopupmenu, 'String', menu_items_cell);
        clear('menu_items','menu_items_cell')

        % -------------------
        % --- Set Enables ---
        % -------------------

        if ~strcmp(data.working_dir, '') % Save location specified
            on_handles = [...
                ... % Modules Pane
                moduleListbox, removePushbutton, editPushbutton,...
                loadPushbutton,...
                ...
                ... % Sensors Pane
                tempCheckbox, pressCheckbox, gyroCheckbox, magnCheckbox,...
                accelCheckbox, cartposCheckbox, geoposCheckbox,...
                cartvelCheckbox, geovelCheckbox, timeCheckbox,...
                ...
                ... % Data Acquisition Pane
                rateSlider, durationSlider, rateEdit, durationEdit,...
                ...
                ... % Visualization Pane
                networkVisPopupmenu, moduleVisPopupmenu, moduleSelPopupmenu,...
                ...
                ... % Menu Items
                new, save_to, module_menu, data_menu, load_modules,...
                edit_modules, remove_selected, load_data,...
                set_recording_location, acquire,...
                ...
                ... % Data Controls
                recordCheckbox, acquirePushbutton];

            off_handles = [...
                ... % Menu Items
                playback, stop,...
                ...
                ... % Visualization Pane
                networkPushbutton, modulePushbutton,...
                ...
                ... % Data Controls
                playbackPushbutton, stopPushbutton];

            set(on_handles, 'Enable', 'on')
            set(off_handles, 'Enable', 'off')


        else % Save location not specified
            on_handles = [...
                ... % Menu Items
                new, save_to];

            off_handles = [...
                ... % Modules Pane
                moduleListbox, removePushbutton, editPushbutton,...
                loadPushbutton,...
                ...
                ... % Sensors Pane
                tempCheckbox, pressCheckbox, gyroCheckbox, magnCheckbox,...
                accelCheckbox, cartposCheckbox, geoposCheckbox,...
                cartvelCheckbox, geovelCheckbox, timeCheckbox,...
                ...
                ... % Data Acquisition Pane
                rateSlider, durationSlider, rateEdit, durationEdit,...
                ...
                ... % Visualization Pane
                networkVisPopupmenu, moduleVisPopupmenu, moduleSelPopupmenu,...
                networkPushbutton, modulePushbutton,...
                ...
                ... % Menu Items
                module_menu, data_menu,...
                ...
                ... % Data Controls
                recordCheckbox, acquirePushbutton, playbackPushbutton,...
                stopPushbutton];

            set(on_handles, 'Enable', 'on')
            set(off_handles, 'Enable', 'off')

            clear('on_handles','off_handles');
        end

    case 'save_to'

        dir_defined = ~strcmp(data.working_dir, '');

        selected_dir = uigetdir(cd, 'Choose location of MCETS working directory...'); % user is prompted to select working directory.
        if selected_dir ~= 0 % if user selects a directory
            sel_dir_name = strtok(fliplr(selected_dir), '\');
            sel_dir_name = fliplr(sel_dir_name);
            sel_children = cellstr(ls(selected_dir));
            if any(strcmp('MCETS', sel_children)) % if any of the selected directory's children are named "MCETS"
                working_dir = [selected_dir '\MCETS'];
            elseif strcmp('MCETS', sel_dir_name) % if the selected directory itself is named "MCETS"
                working_dir = selected_dir;
            else % otherwise, try to create a directory named "MCETS" in the selected directory
                [success,msg,msg_ID] = mkdir(selected_dir,'MCETS');
                if success
```

```
                    working_dir = [selected_dir '\MCETS'];
                else
                    Error_Message(['Could not create MCETS directory. ' msg])
                end
            end
        else % if user cancels the prompt
            working_dir = data.working_dir;
        end
        data.working_dir = working_dir;

        if ~strcmp(data.working_dir, '') % Save location specified
            % Create module_list.txt file
            file_id = fopen([data.working_dir '\module_list.txt'], 'rt');
            if ~(file_id + 1)
                file_id = fopen([data.working_dir '\module_list.txt'], 'wt+');
            end
            module_line = fgetl(file_id);
            module_number = 1;
            while module_line ~= -1
                [IP_address, module_name] = strtok(char(module_line), sprintf
('\t'));
                module_name = module_name(2:end);
                data.IP_list(module_number) = cellstr(IP_address);
                data.name_list(module_number) = cellstr(module_name);
                module_line = fgetl(file_id);
                module_number = module_number + 1;
            end
            status = fclose(file_id);
            if ~(status + 1)
                Error_Message('Could not close file: ''module_list.txt''.')
                return
            end

            % Re-Enable objects if previously no save location was specified
            if ~dir_defined
                on_handles = [...
                    ... % Modules Pane
                    moduleListbox, removePushbutton, editPushbutton,...
                    loadPushbutton,...
                    ...
                    ... % Sensors Pane
                    tempCheckbox, pressCheckbox, gyroCheckbox, magnCheckbox,...
                    accelCheckbox, cartposCheckbox, geoposCheckbox,...
                    cartvelCheckbox, geovelCheckbox, timeCheckbox,...
                    ...
                    ... % Data Acquisition Pane
                    rateSlider, durationSlider, rateEdit, durationEdit,...
                    ...
                    ... % Visualization Pane
                    networkVisPopupmenu, moduleVisPopupmenu, moduleSelPopupmenu,...
                    ...
                    ... % Menu Items
                    new, save_to, module_menu, data_menu, load_modules,...
                    edit_modules, remove_selected, load_data,...
                    set_recording_location, acquire,...
                    ...
                    ... % Data Controls
                    recordCheckbox, acquirePushbutton];

                off_handles = [...
                    ... % Menu Items
                    playback, stop,...
                    ...
                    ... % Visualization Pane
                    networkPushbutton, modulePushbutton,...
                    ...
                    ... % Data Controls
                    playbackPushbutton, stopPushbutton];

                set(on_handles, 'Enable', 'on')
                set(off_handles, 'Enable', 'off')
            end

        else % Save location not specified
            disp('No working directory specified.') % <-- might want to eventually remove this and figure out something else to do.
        end

        set(fhand_0, 'UserData', data)

    case 'exit'
        close;

%% Module Menu Items
    case 'load_modules'
        % ************ READ FROM .txt FILE ***********
        file_id = fopen([data.working_dir '\module_list.txt'], 'rt');
        if ~(file_id + 1)
            file_id = fopen([data.working_dir '\module_list.txt'], 'wt+');
        end
        module_line = fgetl(file_id);
        module_number = 1;
        while module_line ~= -1
            [IP_address, module_name] = strtok(char(module_line), sprintf('\t'));
            module_name = module_name(2:end);
            data.IP_list(module_number) = cellstr(IP_address);
            data.name_list(module_number) = cellstr(module_name);
            module_line = fgetl(file_id);
            module_number = module_number + 1;
        end
        status = fclose(file_id);
        if ~(status + 1)
            Error_Message('Could not close file: ''module_list.txt''.')
            return
        end

        module_list_cell = cell(length(data.IP_list), 1);
        for idx = 1:length(data.IP_list)
            module_list_cell(idx) = cellstr([char(data.name_list(idx)) ' (' char(data.IP_list(idx)) ')']);
        end

        set(fhand_0, 'UserData', data);
        set(moduleListbox, 'String', module_list_cell);

    case 'save_modules'
        % save module list to .txt
        file_id = fopen([data.working_dir '\module_list.txt'], 'wt');
        for module_number = 1:length(data.IP_list)
            temp_string = sprintf([char(data.IP_list(module_number))...
                '\t' char(data.name_list(module_number)) '\n']);
            num_written = fwrite(file_id, temp_string, 'char');
            if (num_written ~= numel(temp_string))
                Error_Message('Write attempt to module_list.txt failed.')
                return
            end
        end
        status = fclose(file_id);
        if ~(status + 1)
            Error_Message('Could not close file: ''module_list.txt''.')
            return
        end
        clear('file_id','temp_string','num_written');

    case 'edit_modules'
        % save module list to .txt
        file_id = fopen([data.working_dir '\module_list.txt'], 'wt');
        for module_number = 1:length(data.IP_list)
            temp_string = sprintf([char(data.IP_list(module_number))...
                '\t' char(data.name_list(module_number)) '\n']);
            num_written = fwrite(file_id, temp_string, 'char');
            if (num_written ~= numel(temp_string))
```

```
                Error_Message('Write attempt to module_list.txt failed.')
                return
            end
        end
        status = fclose(file_id);
        if ~(status + 1)
            Error_Message('Could not close file: ''module_list.txt''.')
            return
        end
        clear('file_id','temp_string','num_written');
        Edit_Modules(fhand_0)

    case 'remove_selected'
        % removes selected items from figure's UserData
        listbox_string = get(moduleListbox, 'String');
        to_remove = zeros(1, length(listbox_string));
        values = get(moduleListbox, 'Value');
        to_remove(values) = 1;
        indices = 1:length(listbox_string);
        indices = find(indices .* ~to_remove);
        name_list = data.name_list;
        IP_list = data.IP_list;
        names_to_keep = name_list(indices);
        IPs_to_keep = IP_list(indices);
        data.name_list = names_to_keep;
        data.IP_list = IPs_to_keep;

        % creates module list from figure's UserData
        module_list = cell(length(data.IP_list), 1);
        for idx = 1:length(data.IP_list)
            module_list(idx) = cellstr([char(data.name_list(idx)) ' (' char(data.IP_list(idx)) ')']);
        end
        set(moduleListbox, 'Value', []);
        set(moduleListbox, 'String', module_list);
        set(fhand_0, 'UserData', data);


%% Data Menu Items
    case 'load_data'
        sp = [data.working_dir '\*.mcets'];
        [fn, pn] = uigetfile({'*.mcets', 'MCETS Telemetry Files (*.mcets)';...
            '*.*', 'All Files'}, 'Load Telemetry Data:', sp);
        if (all(pn ~= 0) && all(fn ~= 0))
            if ~any(strcmp(strcat(pn, fn), data.loaded_data)) % ignores user selection if file has already been loaded.
                data.loaded_data = [data.loaded_data; strcat(pn, fn)];
                set(fhand_0, 'UserData', data);
            end
            clear('sp','fn','pn');

            % Change enables/disables
            off_handles = [acquire; acquirePushbutton;...
                loadPushbutton; load_modules; editPushbutton;...
                remove_selected; recordCheckbox;...
                set_recording_location; removePushbutton;...
                save_to; edit_modules; tempCheckbox; pressCheckbox;...
                accelCheckbox; gyroCheckbox; magnCheckbox; cartposCheckbox;...
                geoposCheckbox; cartvelCheckbox; geovelCheckbox; timeCheckbox;...
                rateSlider; rateEdit; durationSlider; durationEdit];
            set(off_handles, 'Enable', 'off')
            on_handles = [stop; stopPushbutton; playbackPushbutton; playback];
            set(on_handles, 'Enable', 'on')
            clear('off_handles','on_handles');

            % Load module name/IP, checkboxes, rate, duration.
            loaded_data = data.loaded_data;
            module_names = {};
            module_IPs = {};
            rate_setting = 0;
            duration_setting = 0;
            possible_checks = {...
                'set(tempCheckbox, ''Value'', 1); ',...
                'set(pressCheckbox, ''Value'', 1); ',...
                'set(accelCheckbox, ''Value'', 1); ',...
                'set(gyroCheckbox, ''Value'', 1); ',...
                'set(magnCheckbox, ''Value'', 1); ',...
                'set(cartposCheckbox, ''Value'', 1); ',...
                'set(geoposCheckbox, ''Value'', 1); ',...
                'set(cartvelCheckbox, ''Value'', 1); ',...
                'set(geovelCheckbox, ''Value'', 1); ',...
                'set(timeCheckbox, ''Value'', 1); '};
            set(tempCheckbox, 'Value', 0);  % resets checkboxes
            set(pressCheckbox, 'Value', 0);
            set(accelCheckbox, 'Value', 0);
            set(gyroCheckbox, 'Value', 0);
            set(magnCheckbox, 'Value', 0);
            set(cartposCheckbox, 'Value', 0);
            set(geoposCheckbox, 'Value', 0);
            set(cartvelCheckbox, 'Value', 0);
            set(geovelCheckbox, 'Value', 0);
            set(timeCheckbox, 'Value', 0);

            %               HEADER FORMAT
            %  _______________________________________________________
            % | 1 | 2 | 3 | 4 | 5 | 6 - 29 |30 |31 |32 |33 |    % <-- bytes
            % |  CHK  | R |  DUR  |  NAME  |IP1|IP2|IP3|IP4|    % IP1 refers to↙
most significant octet
            for file_num = 1:length(loaded_data)
                fid = fopen(loaded_data{file_num}, 'rt');
                header = fread(fid, 33, 'uint8');

                % Sets sensor checkboxes
                sensor_bin = header(1:2);
                sensor_bin = uint8(dec2bin(sensor_bin, 8))-48;
                checkbox_bool = boolean([sensor_bin(1, 1:5) sensor_bin(2, 1:5)]);
                checks = possible_checks(checkbox_bool);
                eval(strcat(checks{:}));

                % Sets data rate edit box
                rate_head = uint8(header(3));
                if rate_setting == 0
                    set(rateEdit, 'String', num2str(rate_head));
                elseif rate_head ~= rate_setting
                    set(rateEdit, 'String', 'VAR');
                end

                % Sets duration edit box
                duration_head = uint16(header(4:5));
                duration_head = bitshift(duration_head(1), 8) + duration_head(2);
                if duration_setting == 0
                    set(durationEdit, 'String', num2str(duration_head));
                elseif duration_head ~= duration_setting
                    set(durationEdit, 'String', 'VAR');
                end

                % Makes module listbox cell arrays
                module_names = [module_names; deblank(char(header(6:29)'))];
                IP = [' (' num2str(header(30)) '.' num2str(header(31)) '.' num2str↙
(header(32)) '.' num2str(header(33)) ')'];
                module_IPs = [module_IPs; IP];
            end
            % Sets module listbox
            set(moduleListbox, 'String', strcat(module_names, module_IPs));

        end

    case 'set_recording_location'
        sp = [data.working_dir '\*.mcets'];
        [fn, pn] = uiputfile({'*.mcets', 'MCETS Telemetry Files (*.mcets)';...
            '*.*', 'All Files'}, 'Save Telemetry Data As:', sp);
        if (all(pn ~= 0) && all(fn ~= 0))
```

```
            data.rec_loc = strcat(pn, fn);
            set(fhand_0, 'UserData', data);
            file_id = fopen(data.rec_loc, 'wt');
            fclose(file_id);
            clear('sp','fn','pn');
        end

    case 'convert_data'
        loaded_data = data.loaded_data;
        for file_num = 1:length(loaded_data)
            mcets_file = loaded_data{file_num};
            mcets_info = dir(mcets_file);
            disp(['Converting ' mcets_info.name ' (' num2str(mcets_info.bytes) ' bytes)...'])
            txt_file = mcets2txt(mcets_file);
            disp('Done!')
            winopen(txt_file);
        end

    case 'acquire'
        % Change object enables
        off_handles = [acquire; acquirePushbutton; moduleListbox;...
            loadPushbutton; load_modules; editPushbutton;...
            remove_selected; recordCheckbox; playbackPushbutton; playback;...
            new; load_data; set_recording_location; removePushbutton;...
            save_to; edit_modules; tempCheckbox; pressCheckbox;...
            accelCheckbox; gyroCheckbox; magnCheckbox; cartposCheckbox;...
            geoposCheckbox; cartvelCheckbox; geovelCheckbox; timeCheckbox;...
            rateSlider; rateEdit; durationSlider; durationEdit];
        set(off_handles, 'Enable', 'off')
        on_handles = [stop; stopPushbutton];
        set(on_handles, 'Enable', 'on')
        clear('off_handles','on_handles');

        % Change status panel
        set(standbyText, 'Visible', 'off')
        set(liveText, 'Visible', 'on')
        if get(recordCheckbox, 'Value')
            set(recText, 'Visible', 'on')
        end

        % Create and open TCPIP objects
        t_data.main_hand = fhand_0;
        IP_list = data.IP_list;
        name_list = data.name_list;
        start_port = 50; % remote port for all modules.
        failed = boolean(zeros(1, length(IP_list))); % stores which connections fail, if any.

        % *.mcets file has a header containing 2 bytes storing which checkboxes were selected.
        checkboxes_index = boolean([...
            get(tempCheckbox, 'Value'),...
            get(pressCheckbox, 'Value'),...
            get(accelCheckbox, 'Value'),...
            get(gyroCheckbox, 'Value'),...
            get(magnCheckbox, 'Value'),0,0,0,...
            get(cartposCheckbox, 'Value'),...
            get(geoposCheckbox, 'Value'),...
            get(cartvelCheckbox, 'Value'),...
            get(geovelCheckbox, 'Value'),...
            get(timeCheckbox, 'Value'),0,0,0]);
        checkboxes_header = '0000000000000000';
        checkboxes_header(checkboxes_index) = '1';
        checkboxes_header_MSB = checkboxes_header(1:8);
        checkboxes_header_MSB = bin2dec(checkboxes_header_MSB);
        checkboxes_header_LSB = checkboxes_header(9:16);
        checkboxes_header_LSB = bin2dec(checkboxes_header_LSB);
        checkboxes_header = [char(checkboxes_header_MSB) char(checkboxes_header_LSB)];

        % determines data rate and duration currently selected
        rate = get(rateEdit, 'String');
        rate = str2double(rate);
        rate_header = char(rate);
        duration = get(durationEdit, 'String');
        duration = str2double(duration);
        data.acq_duration = duration;
        set(fhand_0, 'UserData', data);
        if isinf(duration)
            duration = 0;
        end
        duration_header = dec2bin(duration, 16);
        duration_MSB = bin2dec(duration_header(1:8));
        duration_LSB = bin2dec(duration_header(9:16));
        duration_header = [char(duration_MSB) char(duration_LSB)];

        connection_list = tcpip('dummy'); % unfortunately, this seems to be the only way to initialize an instrument array
        delete(connection_list);
        for module_num = 1:length(IP_list)
            spaces = '                        '; % white space to pad 24 bytes for name
            name = name_list{module_num};
            name = [name spaces(1:end-length(name))];
            IP = IP_list{module_num};
            [IP1,remain] = strtok(IP, '.');
            [IP2,remain] = strtok(remain(2:end), '.');
            [IP3,remain] = strtok(remain(2:end), '.');
            IP4 = remain(2:end);
            IP1 = char(str2double(IP1));
            IP2 = char(str2double(IP2));
            IP3 = char(str2double(IP3));
            IP4 = char(str2double(IP4));
            IP = [IP1 IP2 IP3 IP4];

            fid = fopen([data.working_dir '\' name_list{module_num} '.mcets'], 'wt'); % overwrites file if necessary
            fclose(fid);
            fid = fopen([data.working_dir '\' name_list{module_num} '.mcets'], 'at'); % permission set to append

            %                HEADER FORMAT
            %  ______________________________________________________
            %  | 1 | 2 | 3 | 4 | 5 | 6 - 29 |30 |31 |32 |33 |     % <-- bytes
            %  |  CHK  | R |  DUR  |  NAME  |IP1|IP2|IP3|IP4|     % IP1 refers to most significant octet

            fwrite(fid, [checkboxes_header rate_header duration_header name IP], 'uint8'); % write 33 byte header
            t_data.fid = fid;
            t_data.full_path = [data.working_dir '\' name_list{module_num} '.mcets'];
            % expected bytes per packet = bytes from checked items +   inTemp if MAG3 is used    + DOP and sat stats if GPS is used + header/footer/batt voltage is always sent
            total_bytes = sum(checkboxes_index .* [2 2 6 6 6 0 0 0 24 24 12 12 9 0 0 0]) + any(checkboxes_index(3:5))*6 + any(checkboxes_index(9:13))*15 + 8;
            t_data.total_bytes = double(rate) * double(duration) * double(total_bytes);
            t_data.duration = double(duration);
            t_data.start_time = clock; % stores the time that connection started.

            t = tcpip(IP_list{module_num}, start_port);
            t.BytesAvailableFcn = @tcpip_cbk;
            t.BytesAvailableFcnMode = 'byte';
            t.BytesAvailableFcnCount = 10;
            t.Timeout = .01;
            t.Terminator = '';
            t.InputBufferSize = 65536;
            t.UserData = t_data;
            connection_list(module_num) = t;
            disp(['Connecting to ' IP_list{module_num} ' ...'])
            try
                fopen(connection_list(module_num));
            catch
                disp('Failed!')
                failed(module_num) = true;
            end
            if ~failed(module_num)
                disp('Connected!')
            end
        end
        pause(1); % allows module to establish connection
        connection_list = connection_list(~failed); % saves only those connections that do not fail

        % Sets connection list and acquiring flag to figure's UserData
        if ~isempty(connection_list)
            data.acquiring_flag = true;
        end
        data.connection_list = connection_list;
        set(fhand_0, 'UserData', data);

        % Sends Request Packets
        connection_list = data.connection_list;
        for module_num = 1:length(connection_list)
            % If RemoteHost of the TCPIP object might be a name rather than
            % an IPv4 address, use the code below.
            %   address = java.net.InetAddress.getByName(get(connection_list(module_num), 'RemoteHost'));
            %   IP_address = char(address.getHostAddress);

            % If RemoteHost of the TCPIP object will always be an IPv4
            % address, use the code below.
            IP_address = get(connection_list(module_num), 'RemoteHost');

            % determines the module ID for the request packet
            module_id = strtok(fliplr(IP_address), '.');
            module_id = fliplr(module_id);
            module_id = str2double(module_id);

            % creates and sends request packet
            req_packet = make_request_packet(module_id, rate, duration, checkboxes_index);
            fwrite(connection_list(module_num), req_packet);
        end

    case 'playback'
        % Change object enables
        off_handles = [acquire; acquirePushbutton; moduleListbox;...
            loadPushbutton; load_modules; editPushbutton;...
            remove_selected; rateSlider; rateEdit; durationSlider;...
            durationEdit; recordCheckbox; playbackPushbutton; playback;...
            new; load_data; set_recording_location; tempCheckbox;...
            pressCheckbox; cartposCheckbox; geoposCheckbox;...
            cartvelCheckbox; geovelCheckbox; timeCheckbox; accelCheckbox;...
            removePushbutton; gyroCheckbox; magnCheckbox;...
            save_to; edit_modules];
        set(off_handles, 'Enable', 'off')
        on_handles = [stop; stopPushbutton];
        set(on_handles, 'Enable', 'on')
        clear('off_handles','on_handles');

        % Change status panel
        set(standbyText, 'Visible', 'off')
        set(playbackText, 'Visible', 'on')

        % Start Visualizations
        open_visuals = data.open_visuals;
        vis_handles = open_visuals(1,:);
        vis_callbacks = open_visuals(2,:);
        % pass in the UserData and the Tag to each visualization
        for index = 1:length(vis_handles)
            user_data = get(vis_handles{index}, 'UserData');
            vis_hand = vis_handles{index};

            % All visualization callbacks will have the option to pass in
            % UserData, a tag called 'beginPlayback' that starts the
            % visualization, and a handle to itself.
            function_call = [vis_callbacks{index} '(user_data, ''beginPlayback'', vis_hand)'];
            eval(function_call);
        end

    case 'stop'
        if data.acquiring_flag
            % Stops Transmission
            connection_list = data.connection_list;
            for module_num = 1:length(connection_list)
                tcpip_cbk(connection_list(module_num), [], NaN)
                fclose(connection_list(module_num));
                t_data = get(connection_list(module_num), 'UserData');

                % data.loaded_data = [data.loaded_data; t_data.full_path]; % add data file to loaded_data field in figure's UserData
                fclose(t_data.fid);
                delete(connection_list(module_num));
            end
            data.connection_list = [];
            data.acq_duration = 0;
            set(fhand_0, 'UserData', data);
        else
            % Stops Playback
            % ***************************************************************

        end

        % Change object enables
        on_handles = [acquire; acquirePushbutton; moduleListbox;...
            loadPushbutton; load_modules; editPushbutton;...
            remove_selected; rateSlider; rateEdit; durationSlider;...
            durationEdit; recordCheckbox; save_to; edit_modules;...
            new; load_data; set_recording_location; tempCheckbox;...
            pressCheckbox; cartposCheckbox; geoposCheckbox;...
            cartvelCheckbox; geovelCheckbox; timeCheckbox; accelCheckbox;...
            gyroCheckbox; magnCheckbox; removePushbutton];

        off_handles = [stop; stopPushbutton];

        if ~isempty(data.loaded_data) % if *.mcets file is loaded for playback
            on_handles = [on_handles; playback; playbackPushbutton];
            off_handles = [off_handles; acquire; acquirePushbutton;...
                moduleListbox; loadPushbutton; load_modules;...
                editPushbutton; remove_selected; rateSlider; rateEdit;...
                durationSlider; durationEdit; recordCheckbox; edit_modules;...
                tempCheckbox; pressCheckbox; cartposCheckbox;...
                geoposCheckbox; cartvelCheckbox; geovelCheckbox;...
                timeCheckbox; accelCheckbox; gyroCheckbox; magnCheckbox;...
                removePushbutton];
        end

        set(on_handles, 'Enable', 'on')
        set(off_handles, 'Enable', 'off')
        clear('off_handles','on_handles');

        % Change status panel
        set(standbyText, 'Visible', 'on')
        set(playbackText, 'Visible', 'off')
        set(liveText, 'Visible', 'off')
        set(recText, 'Visible', 'off')

%% Help Menu Items
    case 'mcets_help'
```

```
        disp('Help is not yet available for MCETS Visualization Tool.')
    case 'about_mcets'
        About_MCETS;

%% Module selection
    case 'moduleListbox'
        possible_modules = get(moduleListbox, 'String');
        selected_modules = possible_modules(get(moduleListbox, 'Value'));
        menu_items = java_array('java.lang.String', 2);
        menu_items(1) = java.lang.String('Select Module');
        menu_items(2) = java.lang.String('---');
        menu_items_cell = cell(menu_items);
        menu_items_cell = [menu_items_cell; selected_modules];
        set(moduleSelPopupmenu, 'String', menu_items_cell);
        clear↙
('menu_items','menu_items_cell','possible_modules','selected_modules');
    case 'loadPushbutton'
        % Read from .txt file
        file_id = fopen([data.working_dir '\module_list.txt'], 'rt');
        if ~(file_id + 1)
            file_id = fopen([data.working_dir '\module_list.txt'], 'wt+');
        end
        module_line = fgetl(file_id);
        module_number = 0;
        while module_line ~= -1
            module_number = module_number + 1;
            [IP_address, module_name] = strtok(char(module_line), sprintf('\t'));
            module_name = module_name(2:end);
            data.IP_list(module_number) = cellstr(IP_address);
            data.name_list(module_number) = cellstr(module_name);
            module_line = fgetl(file_id);
        end
        status = fclose(file_id);
        if ~(status + 1)
            Error_Message('Could not close file: ''module_list.txt''.')
            return
        end

        module_list_cell = cell(length(data.IP_list), 1);
        for idx = 1:length(data.IP_list)
            module_list_cell(idx) = cellstr([char(data.name_list(idx)) ' (' char↙
(data.IP_list(idx)) ')']);
        end
        if nargin == 2
            main_hand = data.main_hand;
            data_temp = get(main_hand, 'UserData');
            data_temp.IP_list = data.IP_list;
            data_temp.name_list = data.name_list;
            set(main_hand, 'UserData', data_temp);
            clear('data_temp')
        else
            set(fhand_0, 'UserData', data);
        end
        set(moduleListbox, 'String', module_list_cell);

    case 'editPushbutton'
        % save module list to .txt
        file_id = fopen([data.working_dir '\module_list.txt'], 'wt');
        for module_number = 1:length(data.IP_list)
            temp_string = sprintf([char(data.IP_list(module_number))...
                '\t' char(data.name_list(module_number)) '\n']);
            num_written = fwrite(file_id, temp_string, 'char');
            if (num_written ~= numel(temp_string))
                Error_Message('Write attempt to module_list.txt failed.')
                return
            end
        end
        status = fclose(file_id);
        if ~(status + 1)
            Error_Message('Could not close file: ''module_list.txt''.')
            return
        end
        clear('file_id','temp_string','num_written');

        Edit_Modules(fhand_0)

    case 'removePushbutton'
        % removes selected items from figure's UserData
        listbox_string = get(moduleListbox, 'String');
        to_remove = zeros(1, length(listbox_string));
        values = get(moduleListbox, 'Value');
        to_remove(values) = 1;
        indices = 1:length(listbox_string);
        indices = find(indices .* ~to_remove);
        name_list = data.name_list;
        IP_list = data.IP_list;
        names_to_keep = name_list(indices);
        IPs_to_keep = IP_list(indices);
        data.name_list = names_to_keep;
        data.IP_list = IPs_to_keep;

        % creates module list from figure's UserData
        module_list = cell(length(data.IP_list), 1);
        for idx = 1:length(data.IP_list)
            module_list(idx) = cellstr([char(data.name_list(idx)) ' (' char(data.IP_list(idx)) ')']);
        end
        set(moduleListbox, 'Value', []);
        set(moduleListbox, 'String', module_list);
        set(fhand_0, 'UserData', data);

%% Sensor Checkboxes
    case 'sens_checkbox'

%% Data Acquisition Settings
    case 'rateSlider'
        rate = get(rateSlider, 'Value');
        rate = floor(99*rate)+1;
        set(rateEdit, 'String', num2str(rate));
        clear('rate');
    case 'durationSlider'
        duration = get(durationSlider, 'Value');
        duration = floor(65534*(duration.^3))+1;
        set(durationEdit, 'String', num2str(duration));
        clear('duration');
    case 'rateEdit'
        rate = str2double(get(rateEdit, 'String'));
        if rate > 100
            rate = 100;
        elseif (rate < 1) || isnan(rate)
            rate = 1;
        else
            rate = round(rate);
        end
        set(rateEdit, 'String', num2str(rate));
        rate = (rate-1)/99;
        set(rateSlider, 'Value', rate);
        clear('rate');
    case 'durationEdit'
        duration = str2double(get(durationEdit, 'String'));
        if isinf(duration) || duration == 0
            duration = Inf;
        elseif duration > 65535
            duration = 65535; % set to max
        elseif (duration < 1) || isnan(duration)
            duration = 1; % set to min
        else
            duration = round(duration);
        end
        set(durationEdit, 'String', num2str(duration));
        if isinf(duration)
            duration = 1; % set slider all the way to the right
        else
            duration = ((duration-1)/65534)^(1/3);
        end
        set(durationSlider, 'Value', duration);
        clear('duration');

%% Visualization Launcher Pane
    case 'moduleSelPopupmenu'
        if get(moduleSelPopupmenu, 'Value') <= 2
            set(moduleListbox, 'Enable', 'on');
            if isempty(data.loaded_data) % dont turn on these buttons if data is↙
loaded
                set(removePushbutton, 'Enable', 'on');
                set(editPushbutton, 'Enable', 'on');
                set(edit_modules, 'Enable', 'on');
                set(remove_selected, 'Enable', 'on');
            end
            set(modulePushbutton, 'Enable', 'off');
            if get(moduleSelPopupmenu, 'Value') == 2
                set(moduleSelPopupmenu, 'Value', 1); % ensures that '---' is not↙
selected.
            end
        else
            % modules may not be removed or edited when a module is selected in↙
popup menu
            set(moduleListbox, 'Enable', 'off');
            set(removePushbutton, 'Enable', 'off');
            set(editPushbutton, 'Enable', 'off');
            set(edit_modules, 'Enable', 'off');
            set(remove_selected, 'Enable', 'off');
            if get(moduleVisPopupmenu, 'Value') > 2
                set(modulePushbutton, 'Enable', 'on');
            end
        end

    case 'moduleVisPopupmenu'
        if get(moduleVisPopupmenu, 'Value') <= 2
            set(modulePushbutton, 'Enable', 'off');
            if get(moduleVisPopupmenu, 'Value') == 2
                set(moduleVisPopupmenu, 'Value', 1); % ensures that '---' is not↙
selected.
            end
            preview_image = zeros(150,250,3);
            image(preview_image, 'Parent', moduleVisAxes);
            axis(moduleVisAxes, 'off')
            axis(moduleVisAxes, 'image')
            set(moduleVisAxes, 'Tag', 'moduleVisAxes')
        else
            if get(moduleSelPopupmenu, 'Value') > 2
                set(modulePushbutton, 'Enable', 'on');
            end
            visualization = get(moduleVisPopupmenu, 'String');
            visualization = char(visualization(get(moduleVisPopupmenu, 'Value')));
            switch visualization
                case 'XYZ Multiplot'
                    % display preview image
                    preview_image = imread('XYZ_Multiplot_pic.bmp', 'BMP');
                    image(preview_image, 'Parent', moduleVisAxes);
                    axis(moduleVisAxes, 'off')
                    axis(moduleVisAxes, 'image')
                    set(moduleVisAxes, 'Tag', 'moduleVisAxes')
            end
        end

    case 'networkVisPopupmenu'
        if get(networkVisPopupmenu, 'Value') <= 2
            set(networkPushbutton, 'Enable', 'off');
            if get(networkVisPopupmenu, 'Value') == 2
                set(networkVisPopupmenu, 'Value', 1); % ensures that '---' is not↙
selected.
            end
        else
            set(networkPushbutton, 'Enable', 'on');
        end

    case 'modulePushbutton'
        visualization = get(moduleVisPopupmenu, 'String');
        visualization = char(visualization(get(moduleVisPopupmenu, 'Value')));
        module_name = get(moduleSelPopupmenu, 'String');
        module_name = module_name{get(moduleSelPopupmenu, 'Value')};
        [tok, remain] = strtok(fliplr(module_name));
        module_name = deblank(fliplr(remain));
        clear('tok','remain');
        loaded_modules = data.loaded_data;
        for index = 1:length(loaded_modules) % flips all loaded paths
            loaded_modules{index} = fliplr(loaded_modules{index});
        end
        loaded_modules = strtok(loaded_modules, '\');
        for index = 1:length(loaded_modules) % flips all loaded paths and drops '.↙
mcets'
            temp_module = loaded_modules{index};
            temp_module = temp_module(7:end);
            temp_module = fliplr(temp_module);
            loaded_modules{index} = temp_module;
        end
        checkboxes_sel = boolean([...
            get(tempCheckbox, 'Value'),...
            get(pressCheckbox, 'Value'),...
            get(accelCheckbox, 'Value'),...
            get(gyroCheckbox, 'Value'),...
            get(magnCheckbox, 'Value'),...
            get(cartposCheckbox, 'Value'),...
            get(geoposCheckbox, 'Value'),...
            get(cartvelCheckbox, 'Value'),...
            get(geovelCheckbox, 'Value'),...
            get(timeCheckbox, 'Value')]);

        switch visualization
            case 'XYZ Multiplot'
                if isempty(data.loaded_data)
                    % Live Data (not yet implemented)
                    XYZ_Multiplot([], 'live')
                else
                    % Recorded Data
                    file_path = strcmp(loaded_modules, module_name);
                    file_path = data.loaded_data(file_path);
                    fig_hand = XYZ_Multiplot(file_path{1}, 'recorded',↙
checkboxes_sel);
                    data.open_visuals{1, end+1} = fig_hand;
                    data.open_visuals{2, end} = 'XYZ_Multiplot_Callback';
                    set(fhand_0, 'UserData', data);
                end
        end

    case 'networkPushbutton'


%% Data pushbuttons
case 'acquirePushbutton'
        % Change object enables
        off_handles = [acquire; acquirePushbutton; moduleListbox;...
            loadPushbutton; load_modules; editPushbutton;...
            remove_selected; recordCheckbox; playbackPushbutton; playback;...
            new; load_data; set_recording_location; removePushbutton;...
            save_to; edit_modules; tempCheckbox; pressCheckbox;...
            accelCheckbox; gyroCheckbox; magnCheckbox; cartposCheckbox;...
            geoposCheckbox; cartvelCheckbox; geovelCheckbox; timeCheckbox;...
            rateSlider; rateEdit; durationSlider; durationEdit];
        set(off_handles, 'Enable', 'off')
        on_handles = [stop; stopPushbutton];
        set(on_handles, 'Enable', 'on')
        clear('off_handles','on_handles');

        % Change status panel
        set(standbyText, 'Visible', 'off')
        set(liveText, 'Visible', 'on')
        if get(recordCheckbox, 'Value')
            set(recText, 'Visible', 'on')
        end

        % Create and open TCPIP objects
        t_data.main_hand = fhand_0;
        IP_list = data.IP_list;
        name_list = data.name_list;
        start_port = 50; % remote port for all modules.
        failed = boolean(zeros(1, length(IP_list))); % stores which connections fail, if any.

        % *.mcets file has a header containing 2 bytes storing which checkboxes were selected.
        checkboxes_index = boolean([...
            get(tempCheckbox, 'Value'),...
            get(pressCheckbox, 'Value'),...
            get(accelCheckbox, 'Value'),...
            get(gyroCheckbox, 'Value'),...
            get(magnCheckbox, 'Value'),0,0,0,...
            get(cartposCheckbox, 'Value'),...
            get(geoposCheckbox, 'Value'),...
            get(cartvelCheckbox, 'Value'),...
            get(geovelCheckbox, 'Value'),...
            get(timeCheckbox, 'Value'),0,0,0]);
        checkboxes_header = '0000000000000000';
        checkboxes_header(checkboxes_index) = '1';
        checkboxes_header_MSB = checkboxes_header(1:8);
        checkboxes_header_MSB = bin2dec(checkboxes_header_MSB);
        checkboxes_header_LSB = checkboxes_header(9:16);
        checkboxes_header_LSB = bin2dec(checkboxes_header_LSB);
        checkboxes_header = [char(checkboxes_header_MSB) char(checkboxes_header_LSB)];

        % determines data rate and duration currently selected
        rate = get(rateEdit, 'String');
        rate = str2double(rate);
        rate_header = char(rate);
        duration = get(durationEdit, 'String');
        duration = str2double(duration);
        data.acq_duration = duration;
        set(fhand_0, 'UserData', data);
        if isinf(duration)
            duration = 0;
        end
        duration_header = dec2bin(duration, 16);
        duration_MSB = bin2dec(duration_header(1:8));
        duration_LSB = bin2dec(duration_header(9:16));
        duration_header = [char(duration_MSB) char(duration_LSB)];

        connection_list = tcpip('dummy'); % unfortunately, this seems to be the only way to initialize an instrument array
        delete(connection_list);
        for module_num = 1:length(IP_list)
            spaces = '                        '; % white space to pad 24 bytes for name
            name = name_list{module_num};
            name = [name spaces(1:end-length(name))];
            IP = IP_list{module_num};
            [IP1,remain] = strtok(IP, '.');
            [IP2,remain] = strtok(remain(2:end), '.');
            [IP3,remain] = strtok(remain(2:end), '.');
            IP4 = remain(2:end);
            IP1 = char(str2double(IP1));
            IP2 = char(str2double(IP2));
            IP3 = char(str2double(IP3));
            IP4 = char(str2double(IP4));
            IP = [IP1 IP2 IP3 IP4];

            fid = fopen([data.working_dir '\' name_list{module_num} '.mcets'],'wt'); % overwrites file if necessary
            fclose(fid);
            fid = fopen([data.working_dir '\' name_list{module_num} '.mcets'],'at'); % permission set to append

            %               HEADER FORMAT
            % ______________________________________________
            % | 1 | 2 | 3 | 4 | 5 | 6 - 29 |30 |31 |32 |33 |    % <-- bytes
            % |  CHK  | R |  DUR  |  NAME  |IP1|IP2|IP3|IP4|    % IP1 refers to most significant octet

            fwrite(fid, [checkboxes_header rate_header duration_header name IP],'uint8'); % write 33 byte header
            t_data.fid = fid;
            t_data.full_path = [data.working_dir '\' name_list{module_num} '.mcets'];
            % expected bytes per packet = bytes from checked items + inTemp if MAG3 is used   + DOP and sat stats if GPS is used + header/footer/batt voltage is always sent
            total_bytes = sum(checkboxes_index .* [2 2 6 6 6 0 0 0 24 24 12 12 9 0 0 0]) + any(checkboxes_index(3:5))*6 + any(checkboxes_index(9:13))*15 + 8;
            t_data.total_bytes = double(rate) * double(duration) * double(total_bytes);
            t_data.duration = double(duration);
            t_data.start_time = clock; % stores the time that connection started.

            t = tcpip(IP_list{module_num}, start_port);
            t.BytesAvailableFcn = @tcpip_cbk;
            t.BytesAvailableFcnMode = 'byte';
            t.BytesAvailableFcnCount = 10;
            t.Timeout = .01;
            t.Terminator = '';
            t.InputBufferSize = 65536;
            t.UserData = t_data;
            connection_list(module_num) = t;
            disp(['Connecting to ' IP_list{module_num} ' ...'])
            try
                fopen(connection_list(module_num));
            catch
                disp('Failed!')
                failed(module_num) = true;
            end
            if ~failed(module_num)
                disp('Connected!')
            end
        end
        pause(1); % allows module to establish connection
        connection_list = connection_list(~failed); % saves only those connections that do not fail

        % Sets connection list and acquiring flag to figure's UserData
        if ~isempty(connection_list)
            data.acquiring_flag = true;
        end
        data.connection_list = connection_list;
        set(fhand_0, 'UserData', data);

        % Sends Request Packets
        connection_list = data.connection_list;
        for module_num = 1:length(connection_list)
            % If RemoteHost of the TCPIP object might be a name rather than
            % an IPv4 address, use the code below.
            %   address = java.net.InetAddress.getByName(get(connection_list(module_num), 'RemoteHost'));
            %   IP_address = char(address.getHostAddress);

            % If RemoteHost of the TCPIP object will always be an IPv4
            % address, use the code below.
            IP_address = get(connection_list(module_num), 'RemoteHost');

            % determines the module ID for the request packet
            module_id = strtok(fliplr(IP_address), '.');
            module_id = fliplr(module_id);
            module_id = str2double(module_id);

            % creates and sends request packet
            req_packet = make_request_packet(module_id, rate, duration,↙
checkboxes_index);
            fwrite(connection_list(module_num), req_packet);
        end

    case 'playbackPushbutton'
        % Change object enables
        off_handles = [acquire; acquirePushbutton; moduleListbox;...
            loadPushbutton; load_modules; editPushbutton;...
            remove_selected; rateSlider; rateEdit; durationSlider;...
            durationEdit; recordCheckbox; playbackPushbutton; playback;...
            new; load_data; set_recording_location; tempCheckbox;...
            pressCheckbox; cartposCheckbox; geoposCheckbox;...
            cartvelCheckbox; geovelCheckbox; timeCheckbox; accelCheckbox;...
            removePushbutton; gyroCheckbox; magnCheckbox;...
            save_to; edit_modules];
        set(off_handles, 'Enable', 'off')
        on_handles = [stop; stopPushbutton];
        set(on_handles, 'Enable', 'on')
        clear('off_handles','on_handles');

        % Change status panel
        set(standbyText, 'Visible', 'off')
        set(playbackText, 'Visible', 'on')

        % Start Visualizations
        open_visuals = data.open_visuals;
        vis_handles = open_visuals(1,:);
        vis_callbacks = open_visuals(2,:);
        % pass in the UserData and the Tag to each visualization
        for index = 1:length(vis_handles)
            user_data = get(vis_handles{index}, 'UserData');
            vis_hand = vis_handles{index};

            % All visualization callbacks will have the option to pass in
            % UserData, a tag called 'beginPlayback' that starts the
            % visualization, and a handle to itself.
            function_call = [vis_callbacks{index} '(user_data, ''beginPlayback'',↙
vis_hand)'];
            eval(function_call);
        end

    case 'stopPushbutton'
        if data.acquiring_flag
            % Stops Transmission
            connection_list = data.connection_list;
            for module_num = 1:length(connection_list)
                tcpip_cbk(connection_list(module_num), [], NaN)
                fclose(connection_list(module_num));
                t_data = get(connection_list(module_num), 'UserData');

                % data.loaded_data = [data.loaded_data; t_data.full_path]; % add↙
data file to loaded_data field in figure's UserData
                fclose(t_data.fid);
                delete(connection_list(module_num));
            end
            data.connection_list = [];
            data.acq_duration = 0;
            set(fhand_0, 'UserData', data);
        else
            % Stops Playback
            % ************************************************************

        end

        % Change object enables
        on_handles = [acquire; acquirePushbutton; moduleListbox;...
            loadPushbutton; load_modules; editPushbutton;...
            remove_selected; rateSlider; rateEdit; durationSlider;...
            durationEdit; recordCheckbox; save_to; edit_modules;...
            new; load_data; set_recording_location; tempCheckbox;...
            pressCheckbox; cartposCheckbox; geoposCheckbox;...
            cartvelCheckbox; geovelCheckbox; timeCheckbox; accelCheckbox;...
            gyroCheckbox; magnCheckbox; removePushbutton];

        off_handles = [stop; stopPushbutton];

        if ~isempty(data.loaded_data) % if *.mcets file is loaded for playback
            on_handles = [on_handles; playback; playbackPushbutton];
            off_handles = [off_handles; acquire; acquirePushbutton;...
                moduleListbox; loadPushbutton; load_modules;...
                editPushbutton; remove_selected; rateSlider; rateEdit;...
                durationSlider; durationEdit; recordCheckbox; edit_modules;...
                tempCheckbox; pressCheckbox; cartposCheckbox;...
                geoposCheckbox; cartvelCheckbox; geovelCheckbox;...
                timeCheckbox; accelCheckbox; gyroCheckbox; magnCheckbox;...
                removePushbutton];
        end

        set(on_handles, 'Enable', 'on')
        set(off_handles, 'Enable', 'off')
        clear('off_handles','on_handles');

        % Change status panel
        set(standbyText, 'Visible', 'on')
        set(playbackText, 'Visible', 'off')
        set(liveText, 'Visible', 'off')
        set(recText, 'Visible', 'off')

end

%% Updates request to modules if necessary
% if any(strcmp(tag,{'sens_checkbox','durationSlider','rateSlider','durationEdit','rateEdit','acquirePushbutton'}))
%     % If MCETS is acquiring data, this block of code will update each
%     % module with a new request.
%     if data.acquiring_flag
%         % determines which checkboxes are currently checked
%         data_options = {...
%             'temperature';...
%             'pressure';...
%             'acceleration';...
%             'angular rate';...
%             'magnetic field';...
%             'cartesian position';...
%             'geodetic position';...
%             'cartesian velocity';...
%             'geodetic velocity';...
%             'receiver time'};
%         data_options = data_options(boolean([...
%             get(tempCheckbox, 'Value'),...
%             get(pressCheckbox, 'Value'),...
%             get(accelCheckbox, 'Value'),...
%             get(gyroCheckbox, 'Value'),...
%             get(magnCheckbox, 'Value'),...
%             get(cartposCheckbox, 'Value'),...
%             get(geoposCheckbox, 'Value'),...
%             get(cartvelCheckbox, 'Value'),...
%             get(geovelCheckbox, 'Value'),...
%             get(timeCheckbox, 'Value')]));
%         checkboxes_sel = make_checkbox_bool(data_options{:});
%
%         % determines data rate and duration currently selected
%         rate = get(rateEdit, 'String');
%         rate = str2double(rate);
%         duration = get(durationEdit, 'String');
%         duration = str2double(duration);
%         duration = uint32(duration - data.acq_duration); % determines duration to add to current
%         data.acq_duration = duration;
%         set(fhand_0, 'UserData', data);
%         if duration == uint32(inf)
%             duration = 0;
%         end
%
%         connection_list = data.connection_list;
%         for module_num = 1:length(connection_list)
%             % If RemoteHost of the TCPIP object might be a name rather than
%             % an IPv4 address, use the code below.
%             %             address = java.net.InetAddress.getByName(get(connection_list(module_num), 'RemoteHost'));
%             %             IP_address = char(address.getHostAddress);
%
%             % If RemoteHost of the TCPIP object will always be an IPv4
%             % address, use the code below.
%             IP_address = get(connection_list(module_num), 'RemoteHost');
%
%             % determines the module ID for the request packet
%             module_id = strtok(fliplr(IP_address), '.');
%             module_id = fliplr(module_id);
%             module_id = str2double(module_id);
%
%             % creates and sends request packet
%             req_packet = make_request_packet(module_id, rate, duration, checkboxes_sel);
%             fwrite(connection_list(module_num), req_packet);
%         end
%     end
% end
% tcpip_cbk     Callback M-file for BytesAvailableFcn
%
%   DEPENDENCIES:
%   MCETS_Main_Callback
%
%   See also MCETS_Main, MCETS_Main_Callback, fwrite

% ==========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% --------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% --------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         October 1, 2007
% LAST EDIT:    Matt Babina, 10/9/07, commented out any superfluous,
%               incomplete, or bugged code segments.
% --------------------------------------------------------------------------
function tcpip_cbk(t, eventdata, varargin)

bytes_recd = get(t, 'ValuesReceived');
t_data = get(t, 'UserData');
fid = t_data.fid;
% duration = t_data.duration;
% total_bytes = t_data.total_bytes;
% bytes_recd = varargin{1};
if t.BytesAvailable>0
    bytes_to_read = t.BytesAvailable;
    data = fread(t, bytes_to_read, 'uint8');

    % Swaps bytes in data.
    dat_length = length(data);
    swapped = zeros(2,dat_length/2);
    swapped(1:dat_length) = data;
    swapped = flipud(swapped);
    data(1:dat_length) = swapped;
    data = uint8(data);

    fwrite(fid, data, 'uint8');
%     if floor(bytes_recd/132) == bytes_recd/132
%         disp([num2str(bytes_recd+bytes_to_read) ' bytes received'])
%     end
%     if etime(clock, t_data.start_time) >= duration
%         % Trigger stop callback
%         MCETS_Main_Callback(t_data.main_hand, 'stopPushbutton');
%     end
end
```

```
% Edit_Modules      M-file for Edit_Modules.fig
%   Displays the Edit Modules window for the MCETS visualization tool.
%
%   USAGE:
%   Edit_Modules                Opens a new Edit Modules window.
%
%   Edit_Modules(FIGHANDLE)     Opens a new Edit Modules window and stores
%                               FIGHANDLE in *.main_hand of the struct in
%                               UserData field of Edit_Modules.fig
%
%   DEPENDENCIES:
%   Error_Message
%
%   See also Edit_Modules_Callback, Error_Message

% ========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% ------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% ------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 17, 2007
% LAST EDIT:    Matt Babina, 10/1/07, changed any reference of 'client' to
%               'module' instead.
% ------------------------------------------------------------------------
function varargout = Edit_Modules(varargin)

%% Load the Edit_Modules figure (Edit_Modules.fig)
fig_def = load('-mat', 'Edit_Modules.fig');
fig_def_names = fieldnames(fig_def);
fig_def = fig_def.(fig_def_names{1});
clear('fig_def_names');

%% Opens and initializes the Edit_Modules figure
fhand_0 = struct2handle(fig_def, 0);
fig_settings.Name = 'Edit Modules';
fig_settings.Resize = 'off';
fig_settings.Tag = 'fig_2';
set(fhand_0, fig_settings);
clear('fig_def', 'fig_settings');
movegui(fhand_0, 'center');

%% Populate Module listbox and initialize Edit_Modules figure data.
modulesListbox = findobj(fhand_0, 'Tag', 'modulesListbox');

data.name_list = cellstr({}); data.IP_list = cellstr({});
if nargin == 1
    data.main_hand = varargin{1};
end

temp_data = get(data.main_hand, 'UserData');
data.working_dir = temp_data.working_dir;
clear('temp_data');
file_id = fopen([data.working_dir '\module_list.txt'], 'rt');
if ~(file_id + 1)
    file_id = fopen([data.working_dir '\module_list.txt'], 'wt+');
end
module_line = fgetl(file_id);
module_number = 1;
while module_line ~= -1
    [IP_address, module_name] = strtok(char(module_line), sprintf('\t'));
    module_name = module_name(2:end);
    data.IP_list(module_number) = cellstr(IP_address);
    data.name_list(module_number) = cellstr(module_name);
    module_line = fgetl(file_id);
    module_number = module_number + 1;
```

```
end
status = fclose(file_id);
if ~(status + 1)
    Error_Message('Could not close file: ''module_list.txt''.')
    return
end

module_list_cell = cell(length(data.IP_list), 1);
for idx = 1:length(data.IP_list)
    module_list_cell(idx) = cellstr([char(data.name_list(idx)) ' (' char(data.IP_list↙
(idx)) ')']);
end


set(modulesListbox, 'String', module_list_cell);
set(fhand_0, 'UserData', data);
set(modulesListbox, 'UserData', data);
set(fhand_0,'WindowStyle','modal'); % makes the window modal
```

```
% Edit_Modules_Callback     Callback M-file for Edit_Modules.fig
%
%   USAGE:
%   Edit_Modules_Callback       Evaluates callback for the calling object.
%
%   DEPENDENCIES:
%   MCETS_Main_Callback
%   Error_Message
%
%   See also Edit_Modules, MCETS_Main_Callback, Error_Message

% ========================================================================
%            Multi-Client Embedded Telemetry System (MCETS) Project
% ------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% ------------------------------------------------------------------------
% PROGRAMMER:    Matt Babina
% DATE:          September 17, 2007
% LAST EDIT:     Matt Babina, 10/10/07, commented section of code to add
%                TCP/IP connection checking, if desired.
% ------------------------------------------------------------------------
function Edit_Modules_Callback(varargin)

%% Retrieve the calling object's parent and its handle
[hObject fhand_0] = gcbo;
if (isempty(hObject) && (nargin == 1))
    hObject = varargin{1}.hObject;
end
if (isempty(fhand_0) && (nargin == 1))
    fhand_0 = varargin{1}.fhand;
end

%% Retrieve the calling object's tag and the figure's stored data
tag = get(hObject, 'Tag');
fig_data = get(fhand_0, 'UserData');

disp(tag) % for debugging

%% Get handles to necessary GUI fields
% IP/Name edits
ipEdit = findobj(fhand_0, 'Tag', 'ipEdit');
nameEdit = findobj(fhand_0, 'Tag', 'nameEdit');

% Connect/Remove pushbuttons
connectPushbutton = findobj(fhand_0, 'Tag', 'connectPushbutton');
removePushbutton = findobj(fhand_0, 'Tag', 'removePushbutton');

% Module listbox
modulesListbox = findobj(fhand_0, 'Tag', 'modulesListbox');
listbox_data = get(modulesListbox, 'UserData'); % retrieve listbox data

% Connection Notification text
connectText = findobj(fhand_0, 'Tag', 'connectText');

% Clear/Cancel/OK pushbuttons
clearPushbutton = findobj(fhand_0, 'Tag', 'clearPushbutton');
cancelPushbutton = findobj(fhand_0, 'Tag', 'cancelPushbutton');
okPushbutton = findobj(fhand_0, 'Tag', 'okPushbutton');

%% Check for "enter" KeyPress condition
if isequal(get(fhand_0,'CurrentKey'),'return') && any(strcmp(tag,↙
{'ipEdit','nameEdit'}))
    tag = 'connectPushbutton';
end

%% Main callback switch
```

```
switch tag
%% Clear/Cancel/OK pushbuttons
    case 'clearPushbutton'
        set(ipEdit, 'String', '0.0.0.0');
        set(nameEdit, 'String', 'Module_0');
        set(modulesListbox, 'Value', 0);
        set(modulesListbox, 'String', []);
        fig_data.IP_list = {};
        fig_data.name_list = {};
        set(fhand_0, 'UserData', fig_data);
    case 'cancelPushbutton'
        close;
    case 'okPushbutton'
        % save module list to .txt
        file_id = fopen([fig_data.working_dir '\module_list.txt'], 'wt');
        for module_number = 1:length(fig_data.IP_list)
            temp_string = sprintf([char(fig_data.IP_list(module_number))...
                '\t' char(fig_data.name_list(module_number)) '\n']);
            num_written = fwrite(file_id, temp_string, 'char');
            if (num_written ~= numel(temp_string))
                Error_Message('Write attempt to module_list.txt failed.')
                return
            end
        end
        status = fclose(file_id);
        if ~(status + 1)
            Error_Message('Could not close file: ''module_list.txt''.')
            return
        end
        clear('file_id','temp_string','num_written');
        MCETS_Main_Callback(fig_data, 'loadPushbutton')
        close;

%% Connect pushbutton
    case 'connectPushbutton'
        % error checking
        IP_address = get(ipEdit, 'String');
        duplicate_flag = 0;
        invalidIP_flag = 0;

        % check for valid IP
        if sum(IP_address == '.') == 3
            [token, remain] = strtok(IP_address, '.');
            IP_strings{1} = token;
            IP_bytes(1) = str2double(token);
            remain = remain(2:end);
            [token, remain] = strtok(remain, '.');
            IP_strings{2} = token;
            IP_bytes(2) = str2double(token);
            remain = remain(2:end);
            [token, remain] = strtok(remain, '.');
            IP_strings{3} = token;
            IP_bytes(3) = str2double(token);
            IP_strings{4} = remain(2:end);
            IP_bytes(4) = str2double(remain(2:end));
            clear('remain','token');
            are_bytes = (IP_bytes <= 255) .* (IP_bytes >= 0) .* (floor(IP_bytes) == IP_bytes);
            if any(~are_bytes)
                invalidIP_flag = 1;
                not_bytes = IP_strings(find(~are_bytes));
                error_string = '';
                for bad_byte = 1:length(not_bytes)
                    error_string = [error_string num2str(char(not_bytes(bad_byte))) ', '];
                end
                error_string = error_string(1:end-2);
                if length(not_bytes) > 1
                    error_string2 = ' are not valid IP octets.';
                else
                    error_string2 = ' is not a valid IP octet.';
                end
                error_string = [error_string error_string2];
                clear('error_string2','bad_byte','not_bytes','are_bytes');
            else
                IP_address = [num2str(IP_bytes(1)) '.' num2str(IP_bytes(2)) '.'...
                    num2str(IP_bytes(3)) '.' num2str(IP_bytes(4))];
            end
        else
            invalidIP_flag = 1;
            error_string = 'IP address must be in dot-decimal notation, containing 4↙
valid octets (0-255) separated by ''.''';
        end
        if invalidIP_flag
            Error_Message(error_string)
            return
        end

        if any(strcmp(IP_address, fig_data.IP_list))
            duplicate_flag = 1; % sets flag if a duplicate IP is entered.
        end
        module_name = get(nameEdit, 'String');
        if strcmp(deblank(module_name), '') || (length(module_name) > 24)
            module_name = ['IP_' IP_address];  % sets a default name based on IP if↙
no valid name is entered.
        end
        while any(strcmp(module_name, fig_data.name_list)) && ~duplicate_flag
            check = 57;
            count = 0;
            while (check <= 57) && (check >= 48)
                check = module_name(end - count);
                count = count+1;
            end
            count = count - 1;
            if (check == '_') && (module_name(end) <= 57 && module_name(end) >= 48)
                number = module_name((end-count+1):end);
                number = str2double(number);
                number = number + 1;
                module_name = [module_name(1:(end-count)) num2str(number)];
            end
            clear('check','count','number');
        end

        % Sets module list to the figure's UserData.
        if ~(any(strcmp(module_name, fig_data.name_list)) && duplicate_flag)
            if duplicate_flag % renames IP
                temp = (1:length(fig_data.IP_list))';
                index = find(temp.*strcmp(IP_address, fig_data.IP_list));
                name_list = cellstr(fig_data.name_list);
                name_list(index) = cellstr(module_name);
                fig_data.name_list = name_list;
                clear('temp','index');
            else % adds new element to list
                name_list = cellstr(fig_data.name_list);
                IP_list = cellstr(fig_data.IP_list);
                name_list = [name_list module_name];
                IP_list = [IP_list IP_address];
                fig_data.name_list = name_list;
                fig_data.IP_list = IP_list;
            end
            % Open TCP/IP objects here with IP_address
            % If desired, TCP/IP objects may be opened then closed to check
            % that the module is ready to receive a request.  If the module
            % cannot be contacted, the connection failed text can be made
            % visible.
        end
        set(fhand_0, 'UserData', fig_data)

        % creates module list from fig_data
        module_list = cell(length(fig_data.IP_list), 1);
        for idx = 1:length(fig_data.IP_list)
            module_list(idx) = cellstr([char(fig_data.name_list(idx)) ' (' char↙
(fig_data.IP_list(idx)) ')']);
        end
        set(modulesListbox, 'String', module_list);
        clear('module_list','name_list','IP_list','duplicate_flag');

%% Remove pushbutton
    case 'removePushbutton'
        % removes selected items from figure's UserData
        listbox_string = get(modulesListbox, 'String');
        to_remove = zeros(1, length(listbox_string));
        values = get(modulesListbox, 'Value');
        to_remove(values) = 1;
        indices = 1:length(listbox_string);
        indices = find(indices .* ~to_remove);
        name_list = fig_data.name_list;
        IP_list = fig_data.IP_list;
        names_to_keep = name_list(indices);
        IPs_to_keep = IP_list(indices);
        fig_data.name_list = names_to_keep;
        fig_data.IP_list = IPs_to_keep;

        % creates module list from figure's UserData
        module_list = cell(length(fig_data.IP_list), 1);
        for idx = 1:length(fig_data.IP_list)
            module_list(idx) = cellstr([char(fig_data.name_list(idx)) ' (' char(fig_data.IP_list(idx)) ')']);
        end
        set(modulesListbox, 'Value', []);
        set(modulesListbox, 'String', module_list);
        set(fhand_0, 'UserData', fig_data);

end



```

F.2. MCETS Server-to-Client Packet Generator (MATLAB Language)

```matlab
% make_request_packet: Make MCETS data request packet.
%   PKT = make_request_packet(MODULE, RATE, T, CHECKBOXES)
%       returns an ASCII string PKT given CLIENT, RATE  T, and CHECKBOXES.
%       MODULE is the last octet of the module's IP address. RATE is the
%       desired rate (in Hz) of data acquisition. T is the desired duration
%       (in seconds) of data acquisition. CHECKBOXES must be a 10-element
%       boolean array representing which checkboxes the user selected for
%       data acquisition.
%
%   See also make_checkbox_bool

% ========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% ------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% ------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 10/9/07, swapped MSB and LSB for duration and
%               whatdata in order to be properly read by the module.
% ------------------------------------------------------------------------
function req_packet = make_request_packet(client_number, rate, duration,↙
checkboxes_sel)

client_number = uint8(client_number);
client_number = char(client_number); % casts client_number as a char

rate = uint8(rate);
rate = char(rate); % casts rate as a char

duration = uint16(duration);
duration_bits = dec2bin(duration, 16); % converts duration to bits
duration_MSB = duration_bits(1:8); % most significant bits of duration
duration_MSB = bin2dec(duration_MSB); % converts back to decimal
duration_MSB = char(duration_MSB); % casts duration_MSB as a char
duration_LSB = duration_bits(9:16); % least significant bits of duration
duration_LSB = bin2dec(duration_LSB); % converts back to decimal
duration_LSB = char(duration_LSB); % casts duration_LSB as a char

if numel(checkboxes_sel) == 10
    whatdata_MSB = [checkboxes_sel(1:5) 0 0 0]; % appends reserved bits
    whatdata_MSB = num2str(whatdata_MSB);
    whatdata_MSB = bin2dec(whatdata_MSB); % converts to decimal
    whatdata_MSB = char(whatdata_MSB); % casts whatdata_MSB as a char
    whatdata_LSB = [checkboxes_sel(6:10) 0 0 0]; % appends reserved bits
    whatdata_LSB = num2str(whatdata_LSB);
    whatdata_LSB = bin2dec(whatdata_LSB); % converts to decimal
    whatdata_LSB = char(whatdata_LSB); % casts whatdata_LSB as a char
else
    whatdata_MSB = checkboxes_sel(1:8);
    whatdata_MSB = num2str(whatdata_MSB);
    whatdata_MSB = bin2dec(whatdata_MSB); % converts to decimal
    whatdata_MSB = char(whatdata_MSB); % casts whatdata_MSB as a char
    whatdata_LSB = checkboxes_sel(9:16);
    whatdata_LSB = num2str(whatdata_LSB);
    whatdata_LSB = bin2dec(whatdata_LSB); % converts to decimal
    whatdata_LSB = char(whatdata_LSB); % casts whatdata_LSB as a char
end

% concatenates the various chars that make up the request packet
% ** module expects LSB first for duration and whatdata
req_packet = [client_number, rate, duration_LSB, duration_MSB, whatdata_LSB,↙
whatdata_MSB];


```

```
% make_checkbox_bool    Make MCETS checkbox select boolean.
%   CHECKBOXES = make_checkbox_bool(varargin)
%       returns a 10-element boolean array, CHECKBOXES, representing which
%       checkboxes the user selected for data acquisition.
%       make_checkbox_bool accepts up to ten strings as inputs,
%       representing the names of checkboxes the user has selected.
%
%   Selection strings are as follows:
%   'temperature'          - MAXIM DS600 Temperature Sensor
%   'pressure'             - Motorola MPX4250A Pressure Sensor
%   'acceleration'         - MEMSense MAG3 (MAG10-1200S050) Accelerometer
%   'angular rate'         - MEMSense MAG3 (MAG10-1200S050) Gyroscope
%   'magnetic field'       - MEMSense MAG3 (MAG10-1200S050) Magnetometer
%   'cartesian position'   - JAVAD GPS (JNS100) Cartesian Position
%   'geodetic position'    - JAVAD GPS (JNS100) Geodetic Position
%   'cartesian velocity'   - JAVAD GPS (JNS100) Cartesian Velocity
%   'geodetic velocity'    - JAVAD GPS (JNS100) Geodetic Velocity
%   'receiver time'        - JAVAD GPS (JNS100) Receiver Time
%
%   See also make_request_packet, parse_data

% ========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% ------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% ------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 10/1/07, fixed bug with unrecognized input
%               strings.
% ------------------------------------------------------------------------
function checkboxes_sel = make_checkbox_bool(varargin)

if nargin <= 10
    checkboxes_sel = zeros(1, 10);
    checkboxes_sel(1) = any(strcmpi(varargin(1:nargin), 'temperature'));
    checkboxes_sel(2) = any(strcmpi(varargin(1:nargin), 'pressure'));
    checkboxes_sel(3) = any(strcmpi(varargin(1:nargin), 'acceleration'));
    checkboxes_sel(4) = any(strcmpi(varargin(1:nargin), 'angular rate'));
    checkboxes_sel(5) = any(strcmpi(varargin(1:nargin), 'magnetic field'));
    checkboxes_sel(6) = any(strcmpi(varargin(1:nargin), 'cartesian position'));
    checkboxes_sel(7) = any(strcmpi(varargin(1:nargin), 'geodetic position'));
    checkboxes_sel(8) = any(strcmpi(varargin(1:nargin), 'cartesian velocity'));
    checkboxes_sel(9) = any(strcmpi(varargin(1:nargin), 'geodetic velocity'));
    checkboxes_sel(10) = any(strcmpi(varargin(1:nargin), 'receiver time'));
    checkboxes_sel = boolean(checkboxes_sel);
else
    error('make_checkbox_bool not defined for more than 10 input arguments.');
end



```

F.3. MCETS Data Packet Parsing Functions (MATLAB Language)

```matlab
% mcets2txt   Converts a *.mcets file to human-readable form
%   TXT = mcets2txt(MCETS)        Parses binary data in a *.mcets file that
%                                 exists with the full path MCETS and writes
%                                 the data in a human-readable form. The
%                                 human-readable file is a *.txt of the same
%                                 name, whose full path is returned, TXT.
%
%   DEPENDENCIES:
%   parse_data
%
%   See also parse_data, MCETS_Main, MCETS_Main_Callback

% =========================================================================
%             Multi-Client Embedded Telemetry System (MCETS) Project
% -------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% -------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 10/10/07, added help.
% -------------------------------------------------------------------------
function txt_file = mcets2txt(mcets_file)

mcets_fid = fopen(mcets_file, 'rt');
mcets_info = dir(mcets_file);
txt_file = [mcets_info.name(1:end-6) '.txt']; % replaces .mcets with .txt for new file name
k = strfind(mcets_file, mcets_info.name);
if k ~= 1
    txt_file = [mcets_file(1:k-1) txt_file];
end
txt_fid = fopen(txt_file, 'wt'); % creates file, overwriting if necessary
fclose(txt_fid);
txt_fid = fopen(txt_file, 'at'); % opens file for writing (appending)

% *.mcets file has a header containing 2 bytes storing which checkboxes were selected.
whatdata_MSB = fread(mcets_fid, 1, 'uint8');
whatdata_MSB = dec2bin(whatdata_MSB, 8);
whatdata_MSB = uint8(whatdata_MSB) - 48;
whatdata_MSB = whatdata_MSB(1:5);
whatdata_LSB = fread(mcets_fid, 1, 'uint8');
whatdata_LSB = dec2bin(whatdata_LSB, 8);
whatdata_LSB = uint8(whatdata_LSB) - 48;
whatdata_LSB = whatdata_LSB(1:5);
checkboxes_sel = boolean([whatdata_MSB whatdata_LSB]); % Sets checkboxes_sel boolean array.

% *.mcets file has a header containing 24 bytes storing the name of the module
fseek(mcets_fid, 6, 'bof');
module_name = fread(mcets_fid, 24, 'uint8');
module_name = char(module_name');
module_name = deblank(module_name);

% *.mcets file has a header containing 4 bytes storing the IPv4 address of the module
module_IP = fread(mcets_fid, 4, 'uint8');
module_IP = [num2str(module_IP(1)) '.' num2str(module_IP(2)) '.' num2str(module_IP(3)) '.' num2str(module_IP(4))];

%% Determines file headings based on checkboxes
heading_headings = {...
    '                       |';...
    '                       |';...
    '                         Acceleration                  |';...
    '                         Angular Rate                  |';...
    '                 Magnetic Field                |';...
    '               Cartesian Position              |';...
    '               Geodetic Position               |';...
    '               Cartesian Velocity              |';...
    '               Geodetic Velocity               |';...
    '         Receiver Time         |'};
column_headings = {...
    'Temperature (C)|';...
    'Pressure (psi) |';...
    'X (m/s/s)      |Y (m/s/s)      |Z (m/s/s)      |';...
    'X (Deg/s)      |Y (Deg/s)      |Z (Deg/s)      |';...
    'X (mGauss)     |Y (mGauss)     |Z (mGauss)     |';...
    'X (m)          |Y (m)          |Z (m)          |';...
    'Latitude (Deg) |Longitude (Deg)|Altitude (m)   |';...
    'X (m/s)        |Y (m/s)        |Z (m/s)        |';...
    'Northing (m/s) |Easting (m/s)  |Altitude (m/s) |';...
    '(MM/DD/YYYY   HH:MM:SS.mmm)    |'};
separator_headings = {...
    '---------------|';...
    '---------------|';...
    '---------------|---------------|---------------|';...
    '---------------|---------------|---------------|';...
    '---------------|---------------|---------------|';...
    '---------------|---------------|---------------|';...
    '---------------|---------------|---------------|';...
    '---------------|---------------|---------------|';...
    '---------------|---------------|---------------|';...
    '-------------------------------|'};
heading_headings = heading_headings(checkboxes_sel);
column_headings = column_headings(checkboxes_sel);
separator_headings = separator_headings(checkboxes_sel);
MAG_used = any(checkboxes_sel(3:5));
GPS_used = any(checkboxes_sel(6:10));
if MAG_used
    heading_headings{end+1} = '           MAG3 Internal  Temperature          |';
    column_headings{end+1}  = 'X (C)          |Y (C)          |Z (C)          |';
    separator_headings{end+1}='---------------|---------------|---------------|';
end
if GPS_used
    heading_headings{end+1} = '                              Dilution of Position                             |';
    column_headings{end+1}  = 'HDOP           |VDOP           |TDOP           |PDOP           |GDOP           |';
    separator_headings{end+1}='---------------|---------------|---------------|---------------|---------------|';
    heading_headings{end+1} = '               |               |               |';
    column_headings{end+1}  = 'Sats Locked    |Sats Available |Sats Used      |';
    separator_headings{end+1}='---------------|---------------|---------------|';
end
heading_headings{end+1} = '               |               |';
column_headings{end+1}  = 'Battery (Volts)|Data Rate (Hz) |';
separator_headings{end+1}='---------------|---------------|';

checkboxes_str = {...
    'MAXIM DS600 Temperature Sensor';...
    'Motorola MPX4250A Pressure Sensor';...
    'MEMSense MAG3 (MAG10-1200S050) Accelerometer';...
    'MEMSense MAG3 (MAG10-1200S050) Gyroscope';...
    'MEMSense MAG3 (MAG10-1200S050) Magnetometer';...
    'JAVAD GPS (JNS100) Cartesian Position';...
    'JAVAD GPS (JNS100) Geodetic Position';...
    'JAVAD GPS (JNS100) Cartesian Velocity';...
    'JAVAD GPS (JNS100) Geodetic Velocity';...
    'JAVAD GPS (JNS100) Receiver Time'};
checkboxes_str = checkboxes_str(checkboxes_sel);

%% Writes heading to .txt file
fwrite(txt_fid, sprintf(['Module:     ' module_name '\n'...
    'IP address: ' module_IP '\n\n']), 'char');


```

```
fwrite(txt_fid, sprintf('Data included in transmission:\n'), 'char');
for index = 1:length(checkboxes_str)
    fwrite(txt_fid, sprintf(['\t' checkboxes_str{index} '\n']), 'char');
end
fwrite(txt_fid, [sprintf('\nLast modification to ') mcets_info.name ': '...
    mcets_info.date sprintf(' (System Time)\n\n\n\n')], 'char');

for index = 1:length(heading_headings)
    fwrite(txt_fid, heading_headings{index}, 'char');
end
fwrite(txt_fid, sprintf('\n'), 'char');
for index = 1:length(column_headings)
    fwrite(txt_fid, column_headings{index}, 'char');
end
fwrite(txt_fid, sprintf('\n'), 'char');
for index = 1:length(separator_headings)
    fwrite(txt_fid, separator_headings{index}, 'char');
end
fwrite(txt_fid, sprintf('\n'), 'char');

%% Writes a line to file for every packet
spaces = '               |';
tic;
packet_length = 1;
while packet_length ~= 0
    header = fread(mcets_fid, 2, 'uint8');
    if length(header) < 2, break; end % exits while loop if there is no next packet↙
to read
    packet_length = uint8(header(2));
    data_string = fread(mcets_fid, double(packet_length), 'uint8');
    data_string = [header; data_string];
    data_packet = parse_data(data_string, checkboxes_sel);

    % If temperature data is part of the data packet
    if isfield(data_packet, 'temperature')
        temperature = data_packet.temperature;

        % calculate temperature (C) based on data
        temperature = double(temperature);
        v_out = (temperature*3.3)/4095;
        temperature = (v_out - 0.509)/0.00645;

        % writes result to file
        temperature = num2str(temperature);
        fwrite(txt_fid, [temperature spaces(end-(15-length(temperature)):end)],↙
'char');
    end

    % If pressure data is part of the data packet
    if isfield(data_packet, 'pressure')
        pressure = data_packet.pressure;

        % calculate pressure (psi) based on data
        pressure = double(pressure);
        v_out = (pressure*5.5)/4095;
        pressure = 0.145038 * (50 * v_out + 10);

        % writes result to file
        pressure = num2str(pressure);
        fwrite(txt_fid, [pressure spaces(end-(15-length(pressure)):end)], 'char');
    end

    % If acceleration data is part of the data packet
    if isfield(data_packet, 'x_accel') % y_accel, z_accel also exist.
        xyz_accel = [data_packet.x_accel; data_packet.y_accel; data_packet.z_accel];

        % calculate acceleration (m/s/s) based on data
        xyz_accel = double(xyz_accel);
        v_out = (xyz_accel.*5.5)./4095;
        xyz_accel = 5 .* v_out - 12.5;
        xyz_accel = 9.80665 .* xyz_accel; % converts from g to m/s/s

        % writes result to file
        xyz_accel = num2str(xyz_accel);
        xyz_accel = strtrim(cellstr(xyz_accel));
        fwrite(txt_fid, [xyz_accel{1} spaces(end-(15-length(xyz_accel{1})):end)...
            xyz_accel{2} spaces(end-(15-length(xyz_accel{2})):end)...
            xyz_accel{3} spaces(end-(15-length(xyz_accel{3})):end)], 'char');
    end

    % If angular rate data is part of the data packet
    if isfield(data_packet, 'x_gyro') % y_gyro , z_gyro also exist.
        xyz_gyro = [data_packet.x_gyro; data_packet.y_gyro; data_packet.z_gyro];

        % calculate angular rate (Deg/s) based on data
        xyz_gyro = double(xyz_gyro);
        v_out = (xyz_gyro.*5.5)./4095;
        xyz_gyro = 800 .* v_out - 2000;

        % writes result to file
        xyz_gyro = num2str(xyz_gyro);
        xyz_gyro = strtrim(cellstr(xyz_gyro));
        fwrite(txt_fid, [xyz_gyro{1} spaces(end-(15-length(xyz_gyro{1})):end)...
            xyz_gyro{2} spaces(end-(15-length(xyz_gyro{2})):end)...
            xyz_gyro{3} spaces(end-(15-length(xyz_gyro{3})):end)], 'char');
    end

    % If magnetic field data is part of the data packet
    if isfield(data_packet, 'x_magnet') % y_magnet , z_magnet also exist.
        xyz_magnet = [data_packet.x_magnet; data_packet.y_magnet; data_packet.z_magnet];

        % calculate magnetic field (mGauss) based on data
        xyz_magnet = double(xyz_magnet);
        v_out = (xyz_magnet.*5.5)./4095;
        xyz_magnet = v_out - 2.5;
        xyz_magnet = xyz_magnet .* 1000; % convert from Gauss to mGauss

        % writes result to file
        xyz_magnet = num2str(xyz_magnet);
        xyz_magnet = strtrim(cellstr(xyz_magnet));
        fwrite(txt_fid, [xyz_magnet{1} spaces(end-(15-length(xyz_magnet{1})):end)...
            xyz_magnet{2} spaces(end-(15-length(xyz_magnet{2})):end)...
            xyz_magnet{3} spaces(end-(15-length(xyz_magnet{3})):end)], 'char');
    end

    % If cartesian position is part of the data packet
    if isfield(data_packet, 'x_pos') % y_pos , z_pos also exist.
        xyz_pos = [data_packet.x_pos; data_packet.y_pos; data_packet.z_pos];

        % writes result to file
        xyz_pos = num2str(xyz_pos);
        xyz_pos = strtrim(cellstr(xyz_pos));
        fwrite(txt_fid, [xyz_pos{1} spaces(end-(15-length(xyz_pos{1})):end)...
            xyz_pos{2} spaces(end-(15-length(xyz_pos{2})):end)...
            xyz_pos{3} spaces(end-(15-length(xyz_pos{3})):end)], 'char');
    end

    % If geodetic position is part of the data packet
    if isfield(data_packet, 'latitude') % longitude , altitude also exist.
        lla_pos = [data_packet.latitude; data_packet.longitude; data_packet.altitude];

        % writes result to file
        lla_pos = num2str(lla_pos);
        lla_pos = strtrim(cellstr(lla_pos));
        fwrite(txt_fid, [lla_pos{1} spaces(end-(15-length(lla_pos{1})):end)...
            lla_pos{2} spaces(end-(15-length(lla_pos{2})):end)...
            lla_pos{3} spaces(end-(15-length(lla_pos{3})):end)], 'char');
    end

    % If cartesian velocity is part of the data packet
    if isfield(data_packet, 'x_vel') % y_vel , z_vel also exist.
        xyz_vel = [data_packet.x_vel; data_packet.y_vel; data_packet.z_vel];

        % writes result to file
        xyz_vel = num2str(xyz_vel);
        xyz_vel = strtrim(cellstr(xyz_vel));
        fwrite(txt_fid, [xyz_vel{1} spaces(end-(15-length(xyz_vel{1})):end)...
            xyz_vel{2} spaces(end-(15-length(xyz_vel{2})):end)...
            xyz_vel{3} spaces(end-(15-length(xyz_vel{3})):end)], 'char');
    end

    % If geodetic velocity is part of the data packet
    if isfield(data_packet, 'north_vel') % east_vel, alt_vel also exist.
        lla_vel = [data_packet.north_vel; data_packet.east_vel; data_packet.alt_vel];

        % writes result to file
        lla_vel = num2str(lla_vel);
        lla_vel = strtrim(cellstr(lla_vel));
        fwrite(txt_fid, [lla_vel{1} spaces(end-(15-length(lla_vel{1})):end)...
            lla_vel{2} spaces(end-(15-length(lla_vel{2})):end)...
            lla_vel{3} spaces(end-(15-length(lla_vel{3})):end)], 'char');
    end

    % If receiver time is part of the data packet
    if isfield(data_packet, 'time_ms') % year, month, day, time_ref also exist.
        date_mdy = [uint16(data_packet.month); uint16(data_packet.day); uint16(data_packet.year)];
        %     time_ref = data_packet.time_ref; % time_ref may be used in future versions.
        time_ms = data_packet.time_ms;

        str0 = '0'; % zero
        % calculate date (MM/DD/YYYY) based on data
        date = [str0(date_mdy(1)<10) num2str(date_mdy(1), '%2.0f') '/' str0(date_mdy(2)<10) num2str(date_mdy(2), '%2.0f') '/' num2str(date_mdy(3), '%4.0f')];

        % calculate time (HH:MM:SS.mmm) based on data
        time_h = floor(time_ms/3600000); % 3600000 ms per hour
        time_ms = rem(time_ms, 3600000);
        time_m = floor(time_ms/60000);   % 60000 ms per minute
        time_ms = rem(time_ms, 60000);
        time_s = floor(time_ms/1000);    % 1000 ms per second
        time_ms = rem(time_ms, 1000);
        time = [str0(time_h<10) num2str(time_h, '%2.0f') ':' str0(time_m<10) num2str(time_m, '%2.0f') ':' str0(time_s<10) num2str(time_s, '%2.0f') '.' str0(time_ms<100) str0(time_ms<10) num2str(time_ms, '%3.0f')];

        date_and_time = [date '   ' time]; % 25 characters long

        fwrite(txt_fid, [date_and_time spaces(10:end)], 'char');
    end

    % If any MAG3 data is part of the data packet
    if isfield(data_packet, 'x_intemp') % y_intemp, z_intemp also exist.
        xyz_intemp = [data_packet.x_intemp; data_packet.y_intemp; data_packet.z_intemp];

        % calculate MAG3 internal temperature (C) based on data
        xyz_intemp = double(xyz_intemp);
        v_out = (xyz_intemp.*5.5)./4095;
        xyz_intemp = (v_out - 2.29) ./ 0.0084;

        % writes result to file
        xyz_intemp = num2str(xyz_intemp);

        xyz_intemp = strtrim(cellstr(xyz_intemp));
        fwrite(txt_fid, [xyz_intemp{1} spaces(end-(15-length(xyz_intemp{1})):end)...
            xyz_intemp{2} spaces(end-(15-length(xyz_intemp{2})):end)...
            xyz_intemp{3} spaces(end-(15-length(xyz_intemp{3})):end)], 'char');
    end

    % If any GPS data is part of the data packet
    if isfield(data_packet, 'hdop') % vdop, tdop, pdop, gdop, sats_locked, sats_avail, sats_used also exist.
        hdop = data_packet.hdop;
        vdop = data_packet.vdop;
        tdop = data_packet.tdop;
        pdop = data_packet.pdop;
        gdop = data_packet.gdop;
        sats_locked = data_packet.sats_locked;
        sats_avail = data_packet.sats_avail;
        sats_used = data_packet.sats_used;

        % writes result to file
        hdop = num2str(hdop);
        vdop = num2str(vdop);
        tdop = num2str(tdop);
        pdop = num2str(pdop);
        gdop = num2str(gdop);
        sats_locked = num2str(sats_locked);
        sats_avail = num2str(sats_avail);
        sats_used = num2str(sats_used);
        fwrite(txt_fid, [hdop spaces(end-(15-length(hdop)):end)...
            vdop spaces(end-(15-length(vdop)):end)...
            tdop spaces(end-(15-length(tdop)):end)...
            pdop spaces(end-(15-length(pdop)):end)...
            gdop spaces(end-(15-length(gdop)):end)...
            sats_locked spaces(end-(15-length(sats_locked)):end)...
            sats_avail spaces(end-(15-length(sats_avail)):end)...
            sats_used spaces(end-(15-length(sats_used)):end)], 'char');
    end

    % batt_voltage is always a field of the data packet.
    batt_voltage = data_packet.batt_voltage';
    fwrite(txt_fid, [batt_voltage spaces(end-(15-length(batt_voltage)):end)], 'char');

    % data_rate is always a field of the data packet.
    data_rate = data_packet.data_rate;
    data_rate = num2str(data_rate);
    fwrite(txt_fid, [data_rate spaces(end-(15-length(data_rate)):end)], 'char');
    fwrite(txt_fid, sprintf('\n'), 'char');
end
toc

%% Closes file
fclose(txt_fid);
```

```
% parse_data    Parse an MCETS data packet.
%   DATA_PACKET = parse_data(DATA_STRING, CHECKBOXES)
%       parses DATA_STRING, an ASCII string, into a struct based on
%       CHECKBOXES.  CHECKBOXES must be a 10-element boolean array
%       representing which checkboxes the user selected for data
%       acquisition. Returns a struct with a field for each selected
%       variable.
%
%   DEPENDENCIES:
%   ASCIIbits2uint8
%   ASCIIbits2uint16
%   ASCIIbits2uint32
%   ASCIIbits2single
%   ASCIIbits2double
%
%   See also ASCIIbits2uint8, ASCIIbits2uint16, ASCIIbits2uint32,
%   ASCIIbits2single, ASCIIbits2double, make_checkbox_bool

% =========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% -------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% -------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 9/25/07, added support for variable packet
%               lengths.
% -------------------------------------------------------------------------
function data_packet = parse_data(data_string, checkboxes_sel)

data_string = char(data_string); % input #1
checkboxes_sel = boolean(checkboxes_sel); % input #2
checkboxes_str =↙
{'temp';'press';'accel';'gyro';'magn';'cartpos';'geopos';'cartvel';'geovel';'time'};
checkboxes = checkboxes_str(checkboxes_sel);
MAG_used = any(strcmp('accel', checkboxes)) ||...
    any(strcmp('gyro', checkboxes)) ||...
    any(strcmp('magn', checkboxes));
GPS_used = any(strcmp('cartpos', checkboxes)) ||...
    any(strcmp('geopos', checkboxes)) ||...
    any(strcmp('cartvel', checkboxes)) ||...
    any(strcmp('geovel', checkboxes));
bytes_sum = 2;
corrupted_str = '';

% -header-
module_ID = uint8(data_string(1));
bytes_to_transmit = uint8(data_string(2)); % does not include the 2-byte header
if bytes_to_transmit == 0
    data_packet.eof = 'END OF FILE'; % might be needed for future applications of↙
parse_data
    return
end

% Temperature
if any(strcmp('temp', checkboxes))

    % Read bytes to struct fields
    data_packet.temperature = data_string(bytes_sum+1:bytes_sum+2);
    bytes_sum = bytes_sum + length(data_packet.temperature);

    % Convert to struct fields to proper type
    data_packet.temperature = ASCIIbits2uint16(data_packet.temperature);
end

% Pressure
```

```
if any(strcmp('press', checkboxes))

    % Read bytes to struct fields
    data_packet.pressure = data_string(bytes_sum+1:bytes_sum+2);
    bytes_sum = bytes_sum + length(data_packet.pressure);

    % Convert to struct fields to proper type
    data_packet.pressure = ASCIIbits2uint16(data_packet.pressure);
end

% Tri-Axial Acceleration
if any(strcmp('accel', checkboxes))

    % Read bytes to struct fields
    data_packet.x_accel = data_string(bytes_sum+1:bytes_sum+2);
    data_packet.y_accel = data_string(bytes_sum+3:bytes_sum+4);
    data_packet.z_accel = data_string(bytes_sum+5:bytes_sum+6);
    bytes_sum = bytes_sum + length(data_packet.x_accel) +...
        length(data_packet.y_accel) +...
        length(data_packet.z_accel);

    % Convert to struct fields to proper type
    data_packet.x_accel = ASCIIbits2uint16(data_packet.x_accel);
    data_packet.y_accel = ASCIIbits2uint16(data_packet.y_accel);
    data_packet.z_accel = ASCIIbits2uint16(data_packet.z_accel);
end

% Tri-Axial Angular Rate
if any(strcmp('gyro', checkboxes))

    % Read bytes to struct fields
    data_packet.x_gyro = data_string(bytes_sum+1:bytes_sum+2);
    data_packet.y_gyro = data_string(bytes_sum+3:bytes_sum+4);
    data_packet.z_gyro = data_string(bytes_sum+5:bytes_sum+6);
    bytes_sum = bytes_sum + length(data_packet.x_gyro) +...
        length(data_packet.y_gyro) +...
        length(data_packet.z_gyro);

    % Convert to struct fields to proper type
    data_packet.x_gyro = ASCIIbits2uint16(data_packet.x_gyro);
    data_packet.y_gyro = ASCIIbits2uint16(data_packet.y_gyro);
    data_packet.z_gyro = ASCIIbits2uint16(data_packet.z_gyro);
end

% Tri-Axial Magnetic Field
if any(strcmp('magn', checkboxes))

    % Read bytes to struct fields
    data_packet.x_magnet = data_string(bytes_sum+1:bytes_sum+2);
    data_packet.y_magnet = data_string(bytes_sum+3:bytes_sum+4);
    data_packet.z_magnet = data_string(bytes_sum+5:bytes_sum+6);
    bytes_sum = bytes_sum + length(data_packet.x_magnet) +...
        length(data_packet.y_magnet) +...
        length(data_packet.z_magnet);

    % Convert to struct fields to proper type
    data_packet.x_magnet = ASCIIbits2uint16(data_packet.x_magnet);
    data_packet.y_magnet = ASCIIbits2uint16(data_packet.y_magnet);
    data_packet.z_magnet = ASCIIbits2uint16(data_packet.z_magnet);
end

% Tri-Axial Internal Temperature (MAG3)
if MAG_used

    % Read bytes to struct fields
    data_packet.x_intemp = data_string(bytes_sum+1:bytes_sum+2);
    data_packet.y_intemp = data_string(bytes_sum+3:bytes_sum+4);
    data_packet.z_intemp = data_string(bytes_sum+5:bytes_sum+6);
    bytes_sum = bytes_sum + length(data_packet.x_intemp) +...
        length(data_packet.y_intemp) +...
        length(data_packet.z_intemp);

    % Convert to struct fields to proper type
    data_packet.x_intemp = ASCIIbits2uint16(data_packet.x_intemp);
    data_packet.y_intemp = ASCIIbits2uint16(data_packet.y_intemp);
    data_packet.z_intemp = ASCIIbits2uint16(data_packet.z_intemp);
end

% Cartesian Position (ECR)
if any(strcmp('cartpos', checkboxes))

    % Read bytes to struct fields
    data_packet.x_pos = data_string(bytes_sum+1:bytes_sum+8);
    data_packet.y_pos = data_string(bytes_sum+9:bytes_sum+16);
    data_packet.z_pos = data_string(bytes_sum+17:bytes_sum+24);
    bytes_sum = bytes_sum + length(data_packet.x_pos) +...
        length(data_packet.y_pos) +...
        length(data_packet.z_pos);

    % Convert to struct fields to proper type
    data_packet.x_pos = ASCIIbits2double(data_packet.x_pos);
    data_packet.y_pos = ASCIIbits2double(data_packet.y_pos);
    data_packet.z_pos = ASCIIbits2double(data_packet.z_pos);
end

% Geodetic Position
if any(strcmp('geopos', checkboxes))

    % Read bytes to struct fields
    data_packet.latitude = data_string(bytes_sum+1:bytes_sum+8);
    data_packet.longitude = data_string(bytes_sum+9:bytes_sum+16);
    data_packet.altitude = data_string(bytes_sum+17:bytes_sum+24);
    bytes_sum = bytes_sum + length(data_packet.latitude) +...
        length(data_packet.longitude) +...
        length(data_packet.altitude);

    % Convert to struct fields to proper type
    data_packet.latitude = ASCIIbits2double(data_packet.latitude);
    data_packet.longitude = ASCIIbits2double(data_packet.longitude);
    data_packet.altitude = ASCIIbits2double(data_packet.altitude);
end

% Cartesian Velocity (ECR)
if any(strcmp('cartvel', checkboxes))

    % Read bytes to struct fields
    data_packet.x_vel = data_string(bytes_sum+1:bytes_sum+4);
    data_packet.y_vel = data_string(bytes_sum+5:bytes_sum+8);
    data_packet.z_vel = data_string(bytes_sum+9:bytes_sum+12);
    bytes_sum = bytes_sum + length(data_packet.x_vel) +...
        length(data_packet.y_vel) +...
        length(data_packet.z_vel);

    % Convert to struct fields to proper type
    data_packet.x_vel = ASCIIbits2single(data_packet.x_vel);
    data_packet.y_vel = ASCIIbits2single(data_packet.y_vel);
    data_packet.z_vel = ASCIIbits2single(data_packet.z_vel);
end

% Geodetic Velocity
if any(strcmp('geovel', checkboxes))

    % Read bytes to struct fields
    data_packet.north_vel = data_string(bytes_sum+1:bytes_sum+4);
    data_packet.east_vel = data_string(bytes_sum+5:bytes_sum+8);
    data_packet.alt_vel = data_string(bytes_sum+9:bytes_sum+12);
    bytes_sum = bytes_sum + length(data_packet.north_vel) +...
        length(data_packet.east_vel) +...
        length(data_packet.alt_vel);

    % Convert to struct fields to proper type
    data_packet.north_vel = ASCIIbits2single(data_packet.north_vel);
    data_packet.east_vel = ASCIIbits2single(data_packet.east_vel);
    data_packet.alt_vel = ASCIIbits2single(data_packet.alt_vel);
end

% Dilution of Position
if GPS_used

    % Read bytes to struct fields
    data_packet.hdop = data_string(bytes_sum+1:bytes_sum+4);
    data_packet.vdop = data_string(bytes_sum+5:bytes_sum+8);
    data_packet.tdop = data_string(bytes_sum+9:bytes_sum+12);
    bytes_sum = bytes_sum + length(data_packet.hdop) +...
        length(data_packet.vdop) +...
        length(data_packet.tdop);

    % Convert to struct fields to proper type
    data_packet.hdop = ASCIIbits2single(data_packet.hdop);
    data_packet.vdop = ASCIIbits2single(data_packet.vdop);
    data_packet.tdop = ASCIIbits2single(data_packet.tdop);

    % Calculate GDOP and PDOP
    data_packet.pdop = sqrt(data_packet.hdop^2 + data_packet.vdop^2);
    data_packet.gdop = sqrt(data_packet.hdop^2 + data_packet.vdop^2 + data_packet.↙
tdop^2);
end

% Satellite Statistics
if GPS_used

    % Read bytes to struct fields
    data_packet.sats_locked = data_string(bytes_sum+1);
    data_packet.sats_avail = data_string(bytes_sum+2);
    data_packet.sats_used = data_string(bytes_sum+3);
    bytes_sum = bytes_sum + length(data_packet.sats_locked) +...
        length(data_packet.sats_avail) +...
        length(data_packet.sats_used);

    % Convert to struct fields to proper type
    data_packet.sats_locked = ASCIIbits2uint8(data_packet.sats_locked);
    data_packet.sats_avail = ASCIIbits2uint8(data_packet.sats_avail);
    data_packet.sats_used = ASCIIbits2uint8(data_packet.sats_used);
end

% Receiver Time
if any(strcmp('time', checkboxes))

    % Read bytes to struct fields
    data_packet.year = data_string(bytes_sum+1:bytes_sum+2);
    data_packet.month = data_string(bytes_sum+3);
    data_packet.day = data_string(bytes_sum+4);
    data_packet.time_ref = data_string(bytes_sum+5);
    data_packet.time_ms = data_string(bytes_sum+6:bytes_sum+9);
    bytes_sum = bytes_sum + length(data_packet.year) +...
        length(data_packet.month) +...
        length(data_packet.day) +...
        length(data_packet.time_ref) +...
        length(data_packet.time_ms);

    % Convert to struct fields to proper type
    data_packet.year = ASCIIbits2uint16(data_packet.year);
    data_packet.month = ASCIIbits2uint8(data_packet.month);
    data_packet.day = ASCIIbits2uint8(data_packet.day);
    data_packet.time_ref = ASCIIbits2uint8(data_packet.time_ref);
    data_packet.time_ms = ASCIIbits2uint32(data_packet.time_ms);
end

% Battery Voltage
data_packet.batt_voltage = data_string(bytes_sum+1:bytes_sum+4);
bytes_sum = bytes_sum + length(data_packet.batt_voltage);

% Data Status
```

```
data_packet.data_rate = uint8(data_string(bytes_sum+1));
data_status = data_string(bytes_sum+2);
bytes_sum = bytes_sum + length(data_packet.data_rate) +...
    length(data_status);
data_status = dec2bin(uint8(data_status), 8);
acq_complete = str2double(data_status(2));
tx_complete = str2double(data_status(4));
osc_fault = str2double(data_status(6));
clear('data_status');
if acq_complete
    corrupted_str = [corrupted_str 'Acquisition Incomplete. '];
end
if tx_complete
    corrupted_str = [corrupted_str 'Transmission Incomplete. '];
end
if bytes_to_transmit ~= bytes_sum-2
    corrupted_str = [corrupted_str 'Bytes Missing. '];
end
if osc_fault
    error('OSCILLATOR FAULT!');
end
data_packet.corrupted_str = corrupted_str;
```

F.4. Miscellaneous MCETS Server Functions (MATLAB Language)

```matlab
% About_MCETS - M-file for About_MCETS.fig
%   Displays information about the MCETS Visualization Tool
%
%   USAGE:
%   About_MCETS                 Displays information about MCETS and labels
%                               version text as 'Version 0.1b'
%
%   About_MCETS('VERSION')      Displays information about MCETS and sets
%                               version text to VERSION

% ========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% ------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% ------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 12, 2007
% LAST EDIT:    Matt Babina, 9/12/07, added help.
% ------------------------------------------------------------------------
function varargout = About_MCETS(varargin)

%% Load the About_MCETS figure (About_MCETS.fig)
fig_def = load('-mat', 'About_MCETS.fig');
fig_def_names = fieldnames(fig_def);
fig_def = fig_def.(fig_def_names{1});
clear('fig_def_names');

%% Opens and initializes the About_MCETS figure
fhand_0 = struct2handle(fig_def, 0);
fig_settings.Name = 'About MCETS Tool';
fig_settings.Resize = 'off';
fig_settings.Tag = 'fig_1';
set(fhand_0, fig_settings);
clear('fig_def', 'fig_settings');
movegui(fhand_0, 'center');

%% Get handles to necessary GUI fields
versionText = findobj(fhand_0, 'Tag', 'versionText');

%% Displays Version
if nargin >= 1
    set(versionText, 'String', varargin{1});
else
    set(versionText, 'String', 'Version 0.1b');
end
```

```
% Error_Message     M-file for Error_Message.fig
%   Displays an error message in a modal window that waits for user
%   response.
%
%   USAGE:
%   Error_Message               passes the string 'ERROR' into the text
%                               field of Error_Message.fig
%
%   Error_Message('MESSAGE')    passes the string MESSAGE into the text
%                               field of Error_Message.fig
%
%   DEPENDENCIES:
%   dialogicons.mat
%
%   See also Error_Message_Callback

% ========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% ------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% ------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 9/25/07, centered text above 'OK' button.
% ------------------------------------------------------------------------
function varargout = Error_Message(varargin)

%% Load the Error_Message figure (Error_Message.fig)
fig_def = load('-mat', 'Error_Message.fig');
fig_def_names = fieldnames(fig_def);
fig_def = fig_def.(fig_def_names{1});
clear('fig_def_names');

%% Opens and initializes the Error_Message figure
fhand_0 = struct2handle(fig_def, 0);
fig_settings.Name = 'Error!';
% fig_settings.Position = [103.8 31 150 30];
fig_settings.Resize = 'off';
fig_settings.Tag = 'fig_0';
set(fhand_0, fig_settings);
clear('fig_def', 'fig_settings');
movegui(fhand_0, 'center');

%% Get handles to necessary GUI fields
axes1 = findobj(fhand_0, 'Tag', 'axes1');
messageText = findobj(fhand_0, 'Tag', 'messageText');

%% Display error message/icon
% Set messageText to display the error message.
if nargin == 1
    set(messageText, 'String', varargin{1});
else
    set(messageText, 'String', 'ERROR');
end

% Show a question icon from dialogicons.mat - variables warnIconData
% and warnIconMap
load dialogicons.mat

IconData=warnIconData;
warnIconMap(256,:) = get(fhand_0, 'Color');
IconCMap=warnIconMap;

Img=image(IconData, 'Parent', axes1);
set(fhand_0, 'Colormap', IconCMap);

set(axes1, ...
    'Visible', 'off', ...
    'YDir'    , 'reverse'         , ...
    'XLim'    , get(Img,'XData'), ...
    'YLim'    , get(Img,'YData')  ...
    );

%% Make the figure modal and makes it wait for user response
set(fhand_0,'WindowStyle','modal')

uiwait(fhand_0); % (see UIRESUME)


```

```
% Error_Message_Callback    Callback M-file for Error_Message.fig
%
%   USAGE:
%   Error_Message_Callback               Evaluates callback for the calling
%                                        object.
%
%   Error_Message_Callback(DATA, TAG)    Evaluates callback for object with
%                                        Tag TAG. Also passes DATA as
%                                        figure's UserData.

% ==========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% --------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% --------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 9/25/07, centered text above 'OK' button.
% --------------------------------------------------------------------------
function Error_Message_Callback(varargin)

%% Retrieve the calling object's parent and its handle
[hObject fhand_0] = gcbo;
if (isempty(hObject) && (nargin == 1))
    hObject = varargin{1}.hObject;
end
if (isempty(fhand_0) && (nargin == 1))
    fhand_0 = varargin{1}.fhand;
end

%% Retrieve the calling object's tag and the figure's stored data
tag = get(hObject, 'Tag');
data = get(fhand_0, 'UserData');

if nargin == 2
    data = varargin{1};
    tag = varargin{2};
end

% disp(tag) % for debugging

%% Main callback switch
%  figure closes on press of "enter", "escape", or the "OK" pushbutton
switch tag
    case 'errorokPushbutton'
        close;
    case 'KeyPress'
        % Check for "enter" or "escape"
        if isequal(get(fhand_0,'CurrentKey'),'escape') || isequal(get↙
(fhand_0,'CurrentKey'),'return')
            close;
        end
end
```

```
% ASCIIbits2double  Convert ASCII to a double-precision number.
%   N = ASCIIbits2double(STR)
%       converts the bits used to encode the ASCII string STR to a double
%       precision floating point number. STR must be a one-dimensional
%       array of chars with length 8. N is of type double.
%
%   See also ASCIIbits2uint8, ASCIIbits2uint16, ASCIIbits2uint32,
%   ASCIIbits2single

% ========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% ------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% ------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 9/25/07, added help.
% ------------------------------------------------------------------------
function number = ASCIIbits2double(ASCII_array)

%% Array size checking
% (Must be a one-dimenional array with a length of 8)
[M,N] = size(ASCII_array);
if M~=1
    if N~=1
        error('Argument must be a one-dimensional array')
    else
        ASCII_array = ASCII_array';
        [M,N] = size(ASCII_array);
    end
end
if N ~= 8
    error(sprintf('Argument must contain 8 ASCII characters'))
end

%% Create an array of bits from ASCII characters
% LSB is placed in bits_array(1) and MSB is placed in bits_array(64)
bits = uint32(ASCII_array);
bits = reshape(bits, 8, N/8);
bits = dec2bin(bits, 8);
bits = bits';
bits = bits(:)';
bits_array = fliplr(uint16(bits) - 48);

%% Translates bits based on IEEE 754 standard
% sign is the MSB (0 == positive, 1 == negative)
sign = double(bits_array(64));

% the integer bit is implicitly 1 unless the value of the double is zero.
integer = 1;

% bits 52 through 62 (bits_array(53:63)) denote the biased exponent.
exponent_powers = uint16(2.^(0:10));
exponent = sum(bits_array(53:63).*exponent_powers);

if exponent == 0 % value of the double is zero.
    exponent = 1023;
    integer = 0;
elseif exponent == 2047 % special cases (reserved exponent)
    if sum(bits_array(1:52))==0 % if the mantissa is all zeros, the double is Inf or↙
-Inf
        number = ((-1)^sign) * inf;
    else % otherwise the double is NaN
        number = NaN;
    end
    return
```

```
end
signif = [boolean(bits_array(1:52)) integer]; % significant bits (integer is implied)
powers = -52:0;
powers = powers + (exponent-1023); % addition of powers (exponent is biased by 1023)
powers = powers(boolean(signif));
number = (-1)^sign * sum(2.^powers);
```

```
% ASCIIbits2single  Convert ASCII to a single-precision number.
%   N = ASCIIbits2single(STR)
%       converts the bits used to encode the ASCII string STR to a single
%       precision floating point number. STR must be a one-dimensional
%       array of chars with length 4. N is of type single.
%
%   See also ASCIIbits2uint8, ASCIIbits2uint16, ASCIIbits2uint32,
%   ASCIIbits2double

% ==========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% --------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% --------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 9/25/07, added help.
% --------------------------------------------------------------------------
function number = ASCIIbits2single(ASCII_array)

%% Array size checking
% (Must be a one-dimenional array with a length of 4)
[M,N] = size(ASCII_array);
if M~=1
    if N~=1
        error('Argument must be a one-dimensional array')
    else
        ASCII_array = ASCII_array';
        [M,N] = size(ASCII_array);
    end
end
if N ~= 4
    error(sprintf('Argument must contain 4 ASCII characters'))
end

%% Create an array of bits from ASCII characters
% LSB is placed in bits_array(1) and MSB is placed in bits_array(32)
bits = uint32(ASCII_array);
bits = reshape(bits, 4, N/4);
bits = dec2bin(bits, 8);
bits = bits';
bits = bits(:)';
bits_array = fliplr(uint16(bits) - 48);

%% Translates bits based on IEEE 754 standard
% sign is the MSB (0 == positive, 1 == negative)
sign = double(bits_array(32));

% the integer bit is implicitly 1 unless the value of the single is zero.
integer = 1;

% bits 23 through 30 (bits_array(24:31)) denote the biased exponent.
exponent_powers = uint16(2.^(0:7));
exponent = sum(bits_array(24:31).*exponent_powers);

if exponent == 0 % value of the double is zero.
    exponent = 127;
    integer = 0;
elseif exponent == 255 % special cases (reserved exponent)
    if sum(bits_array(1:23))==0 % if the mantissa is all zeros, the single is Inf or↙
-Inf
        number = ((-1)^sign) * inf;
    else % otherwise the double is NaN
        number = NaN;
    end
    return
```

```
end
signif = [boolean(bits_array(1:23)) integer]; % significant bits (integer is implied)
powers = -23:0;
powers = powers + (exponent-127); % addition of powers (exponent is biased by 127)
powers = powers(boolean(signif));
number = (-1)^sign * sum(2.^powers);
number = single(number);
```

```
% ASCIIbits2uint8    Convert ASCII to an unsigned 8-bit integer.
%   N = ASCIIbits2uint8(STR)
%       converts the bits used to encode the ASCII string STR to an array
%       of unsigned 8-bit integers. STR must be a one-dimensional array of
%       chars or a single char. N is of type uint8.
%
%   See also ASCIIbits2uint16, ASCIIbits2uint32, ASCIIbits2single,
%   ASCIIbits2double

% ========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% ------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% ------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 9/25/07, added help.
% ------------------------------------------------------------------------
function number = ASCIIbits2uint8(ASCII_array)

[M,N] = size(ASCII_array);
if M~=1
    if N~=1
        error('Argument must be a one-dimensional array')
    else
        ASCII_array = ASCII_array';
    end
end

number = uint8(ASCII_array);
```

```
% ASCIIbits2uint16  Convert ASCII to an unsigned 16-bit integer.
%   N = ASCIIbits2uint16(STR)
%       converts the bits used to encode the ASCII string STR to an array
%       of unsigned 16-bit integers. STR must be a one-dimensional array of
%       chars with length 2*length(N). N is of type uint16.
%
%   See also ASCIIbits2uint8, ASCIIbits2uint32, ASCIIbits2single,
%   ASCIIbits2double

% =========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% -------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% -------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 9/25/07, added help.
% -------------------------------------------------------------------------
function number = ASCIIbits2uint16(ASCII_array)

[M,N] = size(ASCII_array);
if M~=1
    if N~=1
        error('Argument must be a one-dimensional array')
    else
        ASCII_array = ASCII_array';
        [M,N] = size(ASCII_array);
    end
end
if N/2 ~= floor(N/2)
    error('Argument must have an even number of ASCII characters')
end

number = uint16(ASCII_array);
number = reshape(number, 2, N/2);
number(1,:) = bitshift(number(1,:), 8);
number = sum(number);
number = uint16(number);
```

```
% ASCIIbits2uint32  Convert ASCII to an unsigned 32-bit integer.
%   N = ASCIIbits2uint32(STR)
%       converts the bits used to encode the ASCII string STR to an array
%       of unsigned 32-bit integers. STR must be a one-dimensional array of
%       chars with length 4*length(N). N is of type uint32.
%
%   See also ASCIIbits2uint8, ASCIIbits2uint16, ASCIIbits2single,
%   ASCIIbits2double

% ========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% ------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% ------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         September 25, 2007
% LAST EDIT:    Matt Babina, 9/25/07, added help.
% ------------------------------------------------------------------------
function number = ASCIIbits2uint32(ASCII_array)

[M,N] = size(ASCII_array);
if M~=1
    if N~=1
        error('Argument must be a one-dimensional array')
    else
        ASCII_array = ASCII_array';
        [M,N] = size(ASCII_array);
    end
end
if N/4 ~= floor(N/4)
    error('Argument must have a multiple of four ASCII characters')
end

number = uint32(ASCII_array);
number = reshape(number, 4, N/4);
number(3,:) = bitshift(number(3,:), 8);
number(2,:) = bitshift(number(2,:), 16);
number(1,:) = bitshift(number(1,:), 24);
number = sum(number);
number = uint32(number);
```

```
% update_timer_cbk      Callback M-file for TimerFcn
%
%   DEPENDENCIES:
%   XYZ_Multiplot_Callback  (or whatever function call is passed in to
%   varargin{2})
%
%   See also XYZ_Multiplot, XYZ_Multiplot_Callback, timer

% =========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% -------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% -------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         October 1, 2007
% LAST EDIT:    Matt Babina, 10/10/07, added help.
% -------------------------------------------------------------------------
function update_timer_cbk(timer_obj, eventdata, varargin)

fhand = varargin{1};
update_call = varargin{2};
user_data = get(fhand, 'UserData');

function_call = [update_call '(user_data, ''update'', fhand);'];
eval(function_call);
```

```
% XYZ_Multiplot     M-file for XYZ_Multiplot.fig
%   Opens a new XYZ Multiplot visualization window and properly initializes
%   the figure's UserData.  Also properly loads the icons for play/pause
%   controls.
%
%   USAGE:
%   XYZ_Multiplot                   Opens a new XYZ Multiplot Visualization
%                                   with proper initialization, but no file
%                                   path yet specified.
%
%   XYZ_Multiplot(PATH, STR, CHK)   Opens a new XYZ Multiplot Visualization
%                                   with proper initialization and file
%                                   path specified with the string PATH.
%                                   STR may be either 'recorded' or 'live'.
%                                   CHK is a 10 element boolean array
%                                   storing the sensor checkboxes checked.
%
%   DEPENDENCIES:
%   MCETS_icons.mat
%   update_timer_cbk
%
%   See also MCETS_Main_Callback, XYZ_Multiplot_Callback, update_timer_cbk

% =========================================================================
%          Multi-Client Embedded Telemetry System (MCETS) Project
% -------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% -------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         October 5, 2007
% LAST EDIT:    Matt Babina, 10/5/07, added help.
% -------------------------------------------------------------------------
function varargout = XYZ_Multiplot(varargin)

%% Load the XYZ_Multiplot figure (XYZ_Multiplot.fig)
fig_def = load('-mat', 'XYZ_Multiplot.fig');
fig_def_names = fieldnames(fig_def);
fig_def = fig_def.(fig_def_names{1});
clear('fig_def_names');

%% Opens and initializes the XYZ_Multiplot figure
fhand_0 = struct2handle(fig_def, 0);
fig_settings.Name = 'XYZ Multiplot';
% fig_settings.Position = [103.8 31 150 30];
fig_settings.Resize = 'off';
fig_settings.Tag = 'fig_1';
set(fhand_0, fig_settings);
clear('fig_def', 'fig_settings');

%% Get handles to necessary GUI fields
xAxes = findobj(fhand_0, 'Tag', 'xAxes');
yAxes = findobj(fhand_0, 'Tag', 'yAxes');
zAxes = findobj(fhand_0, 'Tag', 'zAxes');
playpausePushbutton = findobj(fhand_0, 'Tag', 'playpausePushbutton');
bofPushbutton = findobj(fhand_0, 'Tag', 'bofPushbutton');
eofPushbutton = findobj(fhand_0, 'Tag', 'eofPushbutton');
datasetPopupmenu = findobj(fhand_0, 'Tag', 'datasetPopupmenu');

%% Label x, y, and z, axes.
ylabel(xAxes, 'x (units)', 'Tag', 'xLabel');
ylabel(yAxes, 'y (units)', 'Tag', 'yLabel');
ylabel(zAxes, 'z (units)', 'Tag', 'zLabel');

%% Create line objects on parent axes
line('XData', 0, 'YData', 0, 'Parent', xAxes, 'Tag', 'xLine', 'Color', 'red');
line('XData', 0, 'YData', 0, 'Parent', yAxes, 'Tag', 'yLine', 'Color', 'red');
line('XData', 0, 'YData', 0, 'Parent', zAxes, 'Tag', 'zLine', 'Color', 'red');

%% Load and display icons
load('MCETS_icons.mat');
set(playpausePushbutton, 'CData', play_icon);
set(bofPushbutton, 'CData', beginning_icon);
set(eofPushbutton, 'CData', end_icon);

%% Populate 'Select Data Set' Popupmenu
checkboxes_sel = varargin{3};
which_sets = checkboxes_sel(3:6);
data_sets_cell = {'Acceleration'; 'Angular Rate'; 'Magnetic Field'; 'Cartesian Position'};
data_sets_cell = data_sets_cell(which_sets);
data_sets_cell = [{'Select Data Set';'---'}; data_sets_cell];
set(datasetPopupmenu, 'String', data_sets_cell);

%% Parse recorded data and save to *.mat file
if ~strcmp(varargin{2}, 'live')
    mcets_fid = fopen(varargin{1}, 'rt');

    % pre-allocates arrays for all possible variables
    fseek(mcets_fid, 2, 'bof');
    rate = fread(mcets_fid, 1, 'uint8');
    duration = fread(mcets_fid, 2, 'uint8');
    duration = uint16(duration);
    duration = bitshift(duration(1), 8) + duration(2);
    max_packets = double(rate) * double(duration);
    try
        acceleration = NaN(max_packets, 3);
        angular_rate = NaN(max_packets, 3);
        magnetic_field = NaN(max_packets, 3);
        cart_position = NaN(max_packets, 3);
        time = repmat({'HH:MM:SS.mmm'}, max_packets, 1);
    catch
        Error_Message('Out of memory. This version of XYZ_Multiplot loads all data before displaying it.');
        return
    end
    fseek(mcets_fid, 34, 'bof'); % seeks past 33 byte header
    packet_length = -1;
    index = 1;
    while packet_length ~= 0
        header = fread(mcets_fid, 2, 'uint8');
        if length(header) < 2, break; end % exits while loop if there is no next packet to read
        packet_length = uint8(header(2));
        data_string = fread(mcets_fid, double(packet_length), 'uint8');
        data_string = [header; data_string];
        data_packet = parse_data(data_string, checkboxes_sel);
        if isfield(data_packet, 'x_accel') % y_accel, z_accel also exist.
            acceleration(index, 1) = data_packet.x_accel;
            acceleration(index, 2) = data_packet.y_accel;
            acceleration(index, 3) = data_packet.z_accel;
        end
        if isfield(data_packet, 'x_gyro') % y_gyro , z_gyro also exist.
            angular_rate(index, 1) = data_packet.x_gyro;
            angular_rate(index, 2) = data_packet.y_gyro;
            angular_rate(index, 3) = data_packet.z_gyro;
        end
        if isfield(data_packet, 'x_magnet') % y_magnet , z_magnet also exist.
            magnetic_field(index, 1) = data_packet.x_magnet;
            magnetic_field(index, 2) = data_packet.y_magnet;
            magnetic_field(index, 3) = data_packet.z_magnet;
        end
        if isfield(data_packet, 'x_pos') % y_pos , z_pos also exist.
            cart_position(index, 1) = data_packet.x_pos;
            cart_position(index, 2) = data_packet.y_pos;
            cart_position(index, 3) = data_packet.z_pos;
        end
        if isfield(data_packet, 'time_ms')
```

```
            str0 = '0'; % zero
            time_ms = data_packet.time_ms;
            % calculate time (HH:MM:SS.mmm) based on data
            time_h = floor(time_ms/3600000); % 3600000 ms per hour
            time_ms = rem(time_ms, 3600000);
            time_m = floor(time_ms/60000);   % 60000 ms per minute
            time_ms = rem(time_ms, 60000);
            time_s = floor(time_ms/1000);    % 1000 ms per second
            time_ms = rem(time_ms, 1000);
            time{index, 1} = [str0(time_h<10) num2str(time_h, '%2.0f') ':' str0↙
(time_m<10) num2str(time_m, '%2.0f') ':' str0(time_s<10) num2str(time_s, '%2.0f') '.'↙
str0(time_ms<100) str0(time_ms<10) num2str(time_ms, '%3.0f')];
        end
        index = index+1;
    end
    acceleration = acceleration(~isnan(acceleration));
    acceleration = reshape(acceleration, numel(acceleration)/3, 3);
    angular_rate = angular_rate(~isnan(angular_rate));
    angular_rate = reshape(angular_rate, numel(angular_rate)/3, 3);
    magnetic_field = magnetic_field(~isnan(magnetic_field));
    magnetic_field = reshape(magnetic_field, numel(magnetic_field)/3, 3);
    cart_position = cart_position(~isnan(cart_position));
    cart_position = reshape(cart_position, numel(cart_position)/3, 3);
    time = time(~strcmp(time, 'HH:MM:SS.mmm'));
    [token, remain] = strtok(fliplr(varargin{1}), '\');
    remain = remain(2:end);
    save_dir = fliplr(remain);
    clear('token','remain');

    % Writes the data arrays to hidden files. XYZ_Multiplot_Callback
    % expects these files to exist.
    warning('off')
    delete([save_dir '\Acceleration.mat']);
    delete([save_dir '\AngularRate.mat']);
    delete([save_dir '\MagneticField.mat']);
    delete([save_dir '\CartPosition.mat']);
    delete([save_dir '\Time.mat']);
    warning('on')
    save([save_dir '\Acceleration'], 'acceleration');
    fileattrib([save_dir '\Acceleration.mat'], '+h');
    save([save_dir '\AngularRate'], 'angular_rate');
    fileattrib([save_dir '\AngularRate.mat'], '+h');
    save([save_dir '\MagneticField'], 'magnetic_field');
    fileattrib([save_dir '\MagneticField.mat'], '+h');
    save([save_dir '\CartPosition'], 'cart_position');
    fileattrib([save_dir '\CartPosition.mat'], '+h');
    save([save_dir '\Time'], 'time');
    fileattrib([save_dir '\Time.mat'], '+h');

end

%% Create update timer
update_timer = timer('Period', 0.1, 'ExecutionMode', 'fixedRate');
set(update_timer, 'TimerFcn', {@update_timer_cbk, fhand_0,↙
'XYZ_Multiplot_Callback'})

%% Initialize the figure's UserData
data.update_timer = update_timer;                       % Stores the timer object↙
for updating visualization
data.status = 'paused';                                 % Stores the play/pause↙
status of the figure
data.time_position = 0;                                 % Stores time position (in↙
seconds) of the visualization
data.data_set = '';                                     % Stores which data set is↙
selected for view
data.rate = rate;                                       % Stores the rate of data to↙
be visualized
if nargin == 3
    data.islive = strcmp(varargin{2}, 'live');          % Stores whether the data is↙
live (streaming)
    data.file_path = varargin{1};                       % Stores the full file path↙
```

```
to the *.mcets data
    if ~data.islive
        data.file_dir = save_dir;                          % Stores the directory of
recorded *.mat data
        data.duration = max([length(acceleration),...   % Stores the duration of
data to be visualized
            length(angular_rate),...
            length(magnetic_field),...
            length(cart_position),...
            length(time)]);
    end
else
    data.islive = '';
    data.file_path = '';
end
set(fhand_0, 'UserData', data);
varargout{1} = fhand_0;
```

```
% XYZ_Multiplot_Callback    Callback M-file for XYZ_Multiplot_Callback.fig
%
%   USAGE:
%   XYZ_Multiplot_Callback          Evaluates callback for the calling
%                                   object.
%
%   XYZ_Multiplot(DATA, TAG, H)     Evaluates callback for object with Tag
%                                   TAG. Also passes DATA as figure's
%                                   UserData and a handle to the figure, H.
%
%   DEPENDENCIES:
%   MCETS_icons.mat
%   Acceleration.mat      _|
%   AngularRate.mat        |
%   MagneticField.mat      |- set in XYZ_Multiplot
%   CartPosition.mat       |
%   Time.mat              _|
%   update_timer_cbk
%
%   See also MCETS_Main_Callback, XYZ_Multiplot, update_timer_cbk

% =========================================================================
%           Multi-Client Embedded Telemetry System (MCETS) Project
% -------------------------------------------------------------------------
% Created by:
%       Matthew Babina
%       Ryan Moniz
%       Michael Sangillo
% for MIT Lincoln Laboratory and Worcester Polytechnic Institute towards
% fulfillment of the Major Qualifying Project.
% -------------------------------------------------------------------------
% PROGRAMMER:   Matt Babina
% DATE:         October 8, 2007
% LAST EDIT:    Matt Babina, 10/11/07, commented out most scrub slider
%               functionality. This should be an easy fix.
% -------------------------------------------------------------------------
function varargout = XYZ_Multiplot_Callback(varargin)


%% Retrieve the calling object's parent and its handle
[hObject fhand_0] = gcbo;
if (isempty(hObject) && (nargin == 1))
    hObject = varargin{1}.hObject;
end
if (isempty(fhand_0) && (nargin == 1))
    fhand_0 = varargin{1}.fhand;
end

%% Retrieve the calling object's tag and the figure's stored data
if nargin == 3
    data = varargin{1};
    tag = varargin{2};
    fhand_0 = varargin{3};
else
    tag = get(hObject, 'Tag');
    data = get(fhand_0, 'UserData');
end

% disp(tag) % for debugging

%% Get handles to necessary GUI fields
datasetPopupmenu = findobj(fhand_0, 'Tag', 'datasetPopupmenu');
xAxes = findobj(fhand_0, 'Tag', 'xAxes');
yAxes = findobj(fhand_0, 'Tag', 'yAxes');
zAxes = findobj(fhand_0, 'Tag', 'zAxes');
xLine = findobj(xAxes, 'Tag', 'xLine');
yLine = findobj(yAxes, 'Tag', 'yLine');
zLine = findobj(zAxes, 'Tag', 'zLine');
x_Label = get(xAxes, 'YLabel');
y_Label = get(yAxes, 'YLabel');
z_Label = get(zAxes, 'YLabel');
```

```
playpausePushbutton = findobj(fhand_0, 'Tag', 'playpausePushbutton');
bofPushbutton = findobj(fhand_0, 'Tag', 'bofPushbutton');
eofPushbutton = findobj(fhand_0, 'Tag', 'eofPushbutton');
scrubSlider = findobj(fhand_0, 'Tag', 'scrubSlider');
timeText = findobj(fhand_0, 'Tag', 'timeText'); % NOTE: need to properly tag time text in *.fig

%% Sets number of seconds on either side of current time on axes.
window_size = 5;

%% Main callback switch
switch tag
    case 'beginPlayback'
        set(bofPushbutton, 'Enable', 'on');
        set(eofPushbutton, 'Enable', 'on');
        set(playpausePushbutton, 'Enable', 'on');
        set(scrubSlider, 'Enable', 'inactive');
        set(xAxes, 'XLim', [-window_size window_size]);
        set(yAxes, 'XLim', [-window_size window_size]);
        set(zAxes, 'XLim', [-window_size window_size]);
        XYZ_Multiplot_Callback(data, 'playpausePushbutton', fhand_0);
%          start(data.update_timer)

    case 'datasetPopupmenu'
        if get(datasetPopupmenu, 'Value') <= 2
            if get(datasetPopupmenu, 'Value') == 2
                set(datasetPopupmenu, 'Value', 1); % ensures that '---' is not selected.
            end
            data.data_set = '';
            set(x_Label, 'String', 'x (units)');
            set(y_Label, 'String', 'y (units)');
            set(z_Label, 'String', 'z (units)');
            set(xAxes, 'YLim', [-1 1]);
            set(yAxes, 'YLim', [-1 1]);
            set(zAxes, 'YLim', [-1 1]);
            set(xAxes, 'YTickLabel', [-1;0;1]);
            set(yAxes, 'YTickLabel', [-1;0;1]);
            set(zAxes, 'YTickLabel', [-1;0;1]);
            set(xAxes, 'YTick', [-1;0;1]);
            set(yAxes, 'YTick', [-1;0;1]);
            set(zAxes, 'YTick', [-1;0;1]);
        else
            data_sets_cell = get(datasetPopupmenu, 'String');
            data_set = char(data_sets_cell(get(datasetPopupmenu, 'Value')));
            switch data_set
                case 'Acceleration'
                    data.data_set = 'Acceleration';
                    set(x_Label, 'String', 'x (m/s≤)');
                    set(y_Label, 'String', 'y (m/s≤)');
                    set(z_Label, 'String', 'z (m/s≤)');
                    set(xAxes, 'YLim', [0 3380]);
                    set(yAxes, 'YLim', [0 3380]);
                    set(zAxes, 'YLim', [0 3380]);
                    set(xAxes, 'YTickLabel', [-20;0;20]);
                    set(yAxes, 'YTickLabel', [-20;0;20]);
                    set(zAxes, 'YTickLabel', [-20;0;20]);
                    set(xAxes, 'YTick', [0;1861;3380]); % a reading of 0 off the 12-bit ADC corresponds to -100 m/s^2
                    set(yAxes, 'YTick', [0;1861;3380]); % a reading of 1861 off the 12-bit ADC corresponds to 0 m/s^2
                    set(zAxes, 'YTick', [0;1861;3380]); % a reading of 3380 off the 12-bit ADC corresponds to 100 m/s^2

                case 'Angular Rate'
                    data.data_set = 'Angular Rate';
                    set(x_Label, 'String', 'x (Deg/s)');
                    set(y_Label, 'String', 'y (Deg/s)');
                    set(z_Label, 'String', 'z (Deg/s)');
                    set(xAxes, 'YLim', [745 2978]);
                    set(yAxes, 'YLim', [745 2978]);
                    set(zAxes, 'YLim', [745 2978]);
                    set(xAxes, 'YTickLabel', [-1200;0;1200]);
                    set(yAxes, 'YTickLabel', [-1200;0;1200]);
                    set(zAxes, 'YTickLabel', [-1200;0;1200]);
                    set(xAxes, 'YTick', [745;1861;2978]); % a reading of 745 off the 12-bit ADC corresponds to -1200 Deg/s
                    set(yAxes, 'YTick', [745;1861;2978]); % a reading of 1861 off the 12-bit ADC corresponds to 0 Deg/s
                    set(zAxes, 'YTick', [745;1861;2978]); % a reading of 2978 off the 12-bit ADC corresponds to 1200 Deg/s

                case 'Magnetic Field'
                    data.data_set = 'Magnetic Field';
                    set(x_Label, 'String', 'x (mGauss)');
                    set(y_Label, 'String', 'y (mGauss)');
                    set(z_Label, 'String', 'z (mGauss)');
                    set(xAxes, 'YLim', [447 3276]);
                    set(yAxes, 'YLim', [447 3276]);
                    set(zAxes, 'YLim', [447 3276]);
                    set(xAxes, 'YTickLabel', [-1900;0;1900]);
                    set(yAxes, 'YTickLabel', [-1900;0;1900]);
                    set(zAxes, 'YTickLabel', [-1900;0;1900]);
                    set(xAxes, 'YTick', [447;1861;3276]); % a reading of 447 off the 12-bit ADC corresponds to -1900 mGauss
                    set(yAxes, 'YTick', [447;1861;3276]); % a reading of 1861 off the 12-bit ADC corresponds to 0 mGauss
                    set(zAxes, 'YTick', [447;1861;3276]); % a reading of 3276 off the 12-bit ADC corresponds to 1900 mGauss

                case 'Cartesian Position'
                    data.data_set = 'Cartesian Position';
                    set(x_Label, 'String', 'x (m)');
                    set(y_Label, 'String', 'y (m)');
                    set(z_Label, 'String', 'z (m)');
                    set(xAxes, 'YLim', [-300 300]);
                    set(yAxes, 'YLim', [-300 300]);
                    set(zAxes, 'YLim', [-300 300]);
                    set(xAxes, 'YTickLabel', [-300;0;300]);
                    set(yAxes, 'YTickLabel', [-300;0;300]);
                    set(zAxes, 'YTickLabel', [-300;0;300]);
                    set(xAxes, 'YTick', [-300;0;300]);
                    set(yAxes, 'YTick', [-300;0;300]);
                    set(zAxes, 'YTick', [-300;0;300]);
            end
        end
        set(fhand_0, 'UserData', data);

    case 'playpausePushbutton'
        if strcmp(data.status, 'playing') % if 'pause' is pressed
            stop(data.update_timer)

            % change play/pause icon
            data.status = 'paused';
            load('MCETS_icons.mat');
            set(playpausePushbutton, 'CData', play_icon);
            set(fhand_0, 'UserData', data);
        elseif strcmp(data.status, 'paused') % if 'play' is pressed
            start(data.update_timer)

            % change play/pause icon
            data.status = 'playing';
            load('MCETS_icons.mat');
            set(playpausePushbutton, 'CData', pause_icon);
            set(fhand_0, 'UserData', data);
        end

    case 'bofPushbutton'
        set(scrubSlider, 'Value', 0);
        data.time_position = 0;
        set(fhand_0, 'UserData', data);

    case 'eofPushbutton'
        set(scrubSlider, 'Value', 1);
        data.time_position = (data.duration/data.rate) - get(data.update_timer,↙
'Period');  % Runs one more cycle of update at the end of file
        set(fhand_0, 'UserData', data);

    case 'scrubSlider'
%         scrub_pos = get(scrubSlider, 'Value');
%         data.time_position = scrub_pos * (data.duration- get(data.update_timer,↙
'Period'));
%
%         set(fhand_0, 'UserData', data);

    case 'update'
        % Loads proper data array
        data_set = data.data_set;
        switch data_set
            case 'Acceleration'
                load([data.file_dir '\Acceleration.mat'])
                YData_array = acceleration;
                clear('acceleration');
            case 'Angular Rate'
                load([data.file_dir '\AngularRate.mat'])
                YData_array = angular_rate;
                clear('angular_rate');
            case 'Magnetic Field'
                load([data.file_dir '\MagneticField.mat'])
                YData_array = magnetic_field;
                clear('magnetic_field');
            case 'Cartesian Position'
                load([data.file_dir '\CartPosition.mat'])
                YData_array = cart_position;
                clear('cart_position');
            case ''
                return
        end
        load([data.file_dir '\Time.mat'])
        update_period = get(data.update_timer, 'Period');
        time_position = data.time_position;
        time_position = round(data.rate*time_position)/data.rate;
        if time_position >= data.duration/data.rate; % length(YData_array)/data.rate
            XYZ_Multiplot_Callback(data, 'playpausePushbutton', fhand_0);
            return
        end

        % Update axes properties.
        % X-axis
        set(xAxes, 'XLim', [time_position-window_size time_position+window_size]);

        % Y-axis
        set(yAxes, 'XLim', [time_position-window_size time_position+window_size]);

        % Z-axis
        set(zAxes, 'XLim', [time_position-window_size time_position+window_size]);


        % Update line properties.
        x_data = time_position-window_size:1/data.rate:time_position; %-↙
update_period;
        % X-line
        y_data = YData_array(:,1);
        y_data = [zeros(window_size*data.rate,1); y_data]; % pads data prior to time↙
zero with zeros
        y_data = y_data(round(time_position*data.rate+1):round((time_position+5)↙
*data.rate+1));
        set(xLine, 'XData', x_data, 'YData', y_data)
        clear('y_data');

        % Y-line
        y_data = YData_array(:,2);
        y_data = [zeros(window_size*data.rate,1); y_data]; % pads data prior to time↙
```

```
zero with zeros
        y_data = y_data(round(time_position*data.rate+1):round((time_position+5)*data.rate+1));
        set(yLine, 'XData', x_data, 'YData', y_data)
        clear('y_data');

        % Z-line
        y_data = YData_array(:,3);
        y_data = [zeros(window_size*data.rate,1); y_data]; % pads data prior to time zero with zeros
        y_data = y_data(round(time_position*data.rate+1):round((time_position+5)*data.rate+1));
        set(zLine, 'XData', x_data, 'YData', y_data)
        clear('y_data');

        % Update time (UTC)
        if numel(time) > 0
            current_time = time{round((time_position)*data.rate+1)};
            set(timeText, 'String', [current_time ' UTC']);
        end

        data.time_position = time_position + update_period;
        scrub_pos = data.time_position / (data.duration/data.rate) - 0.01; % (length(YData_array)/data.rate);
        warning('off');
        set(scrubSlider, 'Value', scrub_pos);
        warning('on');
        set(fhand_0, 'UserData', data);
end
```

Acronym Glossary

- **µA:** Microamperes
- **µW:** Microwatts

- **AC:** Alternating Current
- **ACLK:** Auxiliary Clock
- **ADC:** Analog-to-Digital Converter
- **ADC12:** 12-Bit Analog-to-Digital Converter
- **AGND:** Analog Ground
- **ASCII:** American Standard Code for Information Exchange

- **BMDS:** Ballistic Missile Defense System

- **cm:** Centimeter
- **CMOS:** Complimentary Metal-Oxide Semiconductor
- **CONN:** Connection Status
- **CPHA:** Clock Phase
- **CPOL:** Clock Polarity
- **CPU:** Central Processing Unit
- **CTS:** Clear-to-Send

- **DAC:** Digital-to-Analog Converter
- **dB:** Decibels
- **dBi:** Isotropic Decibels
- **DC:** Direct Current
- **DCO:** Digitally-Controlled Oscillator
- **DCE:** Data Circuit-Terminating Equipment
- **DCCP:** Datagram Congestion Control Protocol

- **DCD:** Data Carrier Detect
- **DDP:** Datagram Delivery Protocol
- **DGND:** Digital Ground
- **DoD:** The United States' Department of Defense
- **DOP:** Dilution of Precision
- **DSR:** Data Set Ready
- **DTE:** Data Terminal Equipment
- **DTR:** Data Terminal Ready

- **EEPROM:** Electrically Erasable Programmable Read-Only Memory
- **EIA:** Electronic Industries Alliance

- **FFRD:** Federally Funded Research and Development Center
- **FTP:** File Transfer Protocol

- **G:** Signal Ground
- **GB:** Gigabyte
- **GDOP:** Geometric Dilution of Precision
- **GLONASS:** Global Navigation Satellite System
- **GNSS:** Global Navigation Satellite System
- **GPS:** Global Positioning System
- **GRIL:** GPS Receiver Interface Language
- **GUI:** Graphical User Interface

- **HC:** High-Current
- **HTTP:** HyperText Transfer Protocol

- **I/O:** Input/Output
- **IEEE:** Institute of Electrical and Electronics Engineers
- **IC:** Integrated Circuit

- **IMAP:**	Internet Message Access Protocol
- **IP:**	Internet Protocol
- **IPX:**	Internetwork Packet Exchange
- **kΩ:**	Kiloohm
- **KB:**	Kilobytes
- **kbps:**	Kilobits per Second
- **HDOP:**	Horizontal Dilution of Precision
- **KHz:**	Kilohertz
- **ksps:**	Kilosamples per Second
- **LSB:**	Least Significant Bit
- **LC:**	Low Current
- **LCD:**	Liquid Crystal Display
- **LLC:**	Logical Link Control
- **MAC:**	Media Access Control
- **MATLAB:**	Matrix Laboratory
- **MB:**	Megabytes
- **Mbps:**	Megabits per Second
- **MCETS:**	Multi-Client Embedded Telemetry System
- **MCLK:**	Main Clock
- **MHz:**	Megahertz
- **MIPS:**	Million Instructions per Second
- **MISO:**	Master Input, Slave Output
- **MIT:**	Massachusetts Institute of Technology
- **mm:**	Millimeters
- **MOSFET:**	Metal-Oxide-Semiconductor Field-Effect Transistor
- **MOSI:**	Master Output, Slave Input
- **MQP:**	Major Qualifying Project

- **MSB:** Most Significant Bit
- **MUX:** Analog Multiplexer
- **mV:** Millivolts
- **mW:** Milliwatts
- **ns:** Nanoseconds

- **OEM:** Original Equipment Manufacture
- **Op-Amp:** Operational Amplifier

- **PCB:** Printed Circuit Board
- **PDOP:** Position Dilution of Precision
- **PGND:** Power Ground
- **POP:** Post Office Protocol
- **POST:** Power On Self Test
- **PPP:** Point-to-Point Protocol
- **PR:** Pseudo Range
- **PSI:** Pounds per Square Inch

- **RAM:** Random Access Memory
- **RF:** Radio Frequency
- **RI:** Ring Indicator
- **RS-232:** Recommend Standard 232
- **RTS:** Request-to-Send
- **RxD:** Received Data

- **SAR:** Successive-Approximation-Register
- **SCLK:** Serial Clock
- **SCTP:** Stream Control Transmission Protocol
- **SDI:** Serial Data In

- **SDO:** Serial Data Out
- **SIMO:** Slave Input, Master Output
- **SMCLK:** Submain Clock
- **SMTP:** Simple Mail Transfer Protocol
- **SOAP:** Simple Object Access Protocol
- **SOMI:** Slave Output, Master Input
- **SPI:** Serial Peripheral Interface
- **SSID:** Service Select Identifier
- **SSH:** Secure Shell

- **TCP:** Transmission Control Protocol
- **TCP/IP:** Transmission Control Protocol/Internet Protocol
- **TDOP:** Time Dilution of Precision
- **TTL:** Transistor-Transistor Logic
- **TxD:** Transmitted Data

- **UART:** Universal Asynchronous Receiver-Transmitter
- **UDP:** User Datagram Protocol
- **USART:** Universal Synchronous/Asynchronous Receiver-Transmitter
- **USB:** Universal Serial Bus
- **UTC:** Coordinated Time Universal

- **VDOP:** Vertical Dilution of Precision

- **WITS:** Wireless Instrumentation and Telemetry System
- **WPI:** Worcester Polytechnic Institute