DATA ACQUISITION UNIT FOR A CVT DYNAMOMETER

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Michael J. Kilzer

MEng Field Advisor: Bruce R. Land

Degree Date: May, 2013

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Data Acquisition Unit for a CVT Dynamometer

Author: Michael J. Kilzer

Abstract:

This report discusses the design, implementation, and testing of a custom-built data acquisition board for the Baja SAE team at Cornell University. Specifically, the data acquisition unit is to be used on the team's continuously variable transmission dynamometer. The CVT is responsible for transmitting the power from the motor to the gearbox which drives the wheels of the Baja vehicle. Therefore, it is important to correctly tune the CVT for maximum power throughput using the dynamometer. This project aids in that endeavor by providing the team with the means to collect data from 16 analog channels, each with 12-bit accuracy. Hardware has also been implemented to amplify and filter the analog signals coming from various sensors the team may use. Furthermore, the project provides the team with the capability to integrate digital components and motor controllers for additional extensibility in its dynamometer setup. The data gathered by the data acquisition unit is sampled at a user-specified rate and sent to a computer terminal for further processing. The data acquisition unit is user-configurable via a command string sent from the computer to the unit via USB. Testing of the external interrupts, ADCs, and filter frequency response are performed to show that the project meets its specifications.

Contents

List of Figures

1 Executive Summary

The purpose of this project is to develop, build, and test improved electronics for the Baja SAE team's CVT dynamometer. A continuously variable transmission works through a system of clutches, flyweights, springs, and a belt. Clutches are attached to the motor and the drivetrain of the Baja vehicle. In each clutch is a spring and flyweight system. These springs and flyweights are tuned such that they exert forces on the sheaves of the clutches to push a belt along the radius of the clutch as they spin. Through changing the radii, the belt traverses a continuum of gear ratios between the engine and drivetrain after the motor reaches a certain RPM. The data acquisition system for the CVT dynamometer now only takes 3 inputs, limiting the amount of data the Baja team can measure in any particular run. The amplifiers and filters also saturate, which can cause problems when attempting to record accurate data. The current DAQ only samples at 10 samples/second. While this is adequate, it is a bit slow.

This project is a large improvement over the old electronics in the CVT dynamometer. The data acquisition board can sample 16 analog channels concurrently with 12-bit accuracy. Furthermore, the board can communicate over 16 general purpose input/output pins. The data acquisition board also counts the number of interrupts that occur on each external interrupt, which is useful for gathering RPM data. There were some minor issues in achieving communication between the board and computer, but these problems were easily solved after examining the schematic closely.

The amplifier board is responsible for both the amplification of small analog signals and filtering out high frequency noise. The amplification is accomplished through the use of an instrumentation amplifier, and the low-pass filtering is accomplished though the use of a Sallen-Key low-pass filter. The output of the amplifier is passed into the LPF and another output on the board so the user can choose between unfiltered and filtered outputs. There were a couple problems encountered when debugging this board. However, these problems were also solved through a careful examination of the schematics and some probing work with an oscilloscope.

In all, the project met and in some cases surpassed its requirements. The integration of this data acquisition hardware into the new dynamometer will help the team realize a better-tuned CVT and in turn perform even better in dynamic events at competition.

2 Design Problem

2.1 Initial Thoughts

As a member of the Baja SAE team during my undergraduate time at Cornell, it was only natural to pursue a Master of Engineering project that would allow me to contribute to the success of the team. Several ideas were put forward including an electronic continuously variable transmission. After playing with the ECVT idea for a while, I found the idea to be rather impractical for the team's needs and decided to pursue a different idea. Some team members suggested creating improved and configurable hardware for the team's CVT dynamometer. The idea intrigued me and I decided to pursue this idea as my project.

The CVT dynamometer had been in a state of constant development since I joined the team in January 2009. A few different designs had been tried both mechanically and electronically. However, none of the designs were ever very reliable or easy to use. Mechanically, the stand holding the CVT and motor in place did not dampen the vibrations well, leading to the stand shaking rather violently and with an appreciable amount of noise. The electronics were often unreliable as well, either saturating or simply not working.

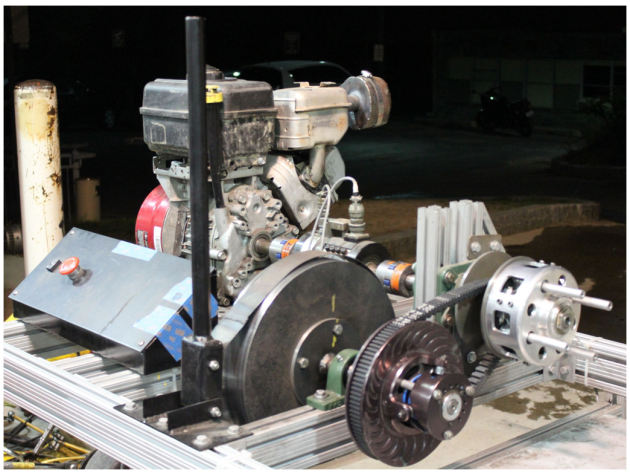

Figure 1: New Dynamometer

During the course of this school year, Brian Curless, a member of the Baja electrical subteam designed, built, and tested a new and improved CVT dynamometer testing station, shown in figure 1. The station was much more useable and reliable than previous iterations. Brian worked to develop the mechanical and initial electrical components used for the dynamometer. The dynamometer worked very well and the testing performed with it helped the team to do well in dynamic events at the 2013 Tennessee Baja SAE competition. The electronics consisted of hall effect sensors that measured engine and load flywheel RPM and another sensor that measured the output of a torque sensor. This data was processed and sent to a

computer terminal.

While this is adequate for basic testing, the team desired the ability to read data from more sensors and configure the data acquisition unit accordingly. Taking what I had learned from SYSEN 5100, I started the process of gathering information to help me better define my project.

2.2 Context Diagram

In order to have a better understanding of the system that I was to build, I needed to see how the other parts of the CVT dynamometer were to interact with my system. Knowing this information is very helpful since systems often fail at their interfaces. This helps to prevent misunderstandings and problems further down the road when it comes time to integrate the hardware into the dynamometer itself.

The context diagram allows the designer to determine how the system will interact with its outside environment. This tool is to be used once some preliminary designs have been suggested for the system. The context diagram consists of a box representing the system that is surrounded by boxes describing the various outside systems with which the system can interact. Rectilinear lines are drawn and annotated to illustrate how the systems are related to one another. In using the context diagram, I was better able to characterize how the data acquisition hardware would interact with systems both internal and external to the dynamometer unit. The context diagram I used for planning this project is shown in figure 2.

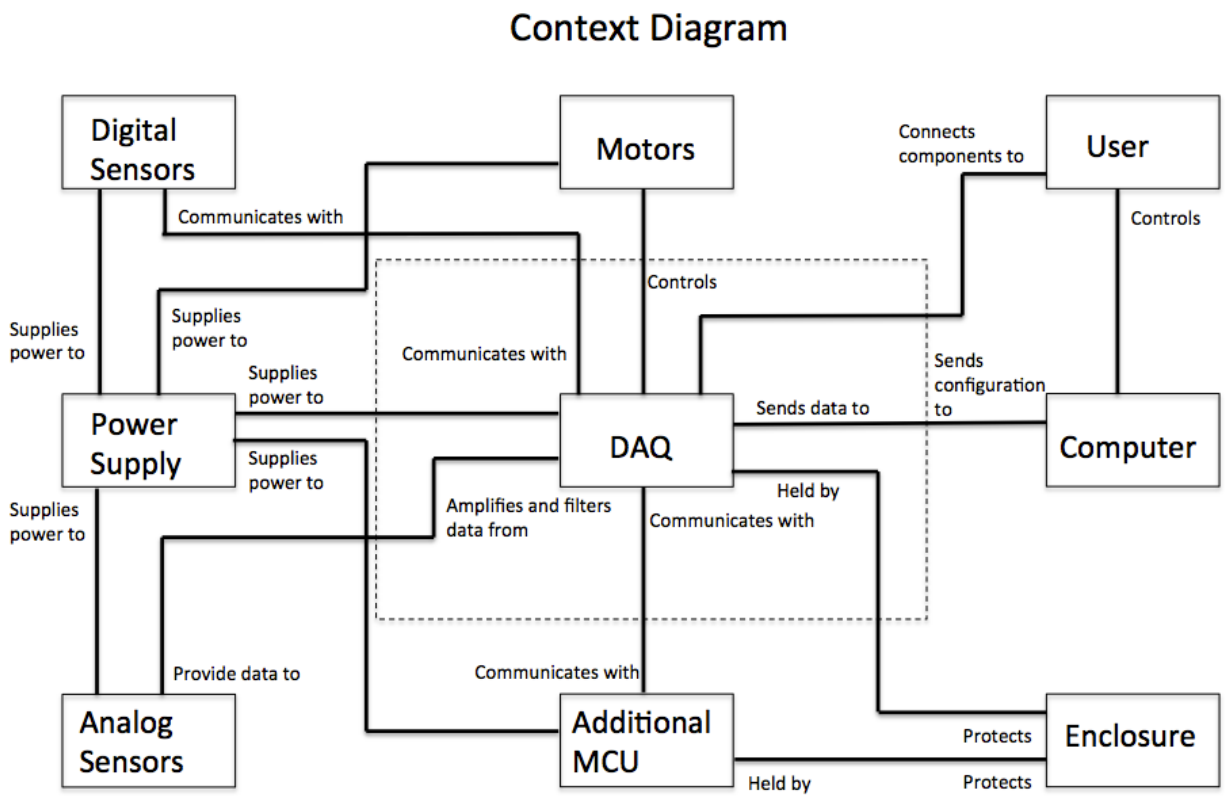

Figure 2: Context Diagram for CVT Dynamometer DAQ

In looking at the context diagram, one can see that the data acquisition unit needs to interact with the user, computer, its enclosure, analog and digital sensors, power supply, and even servo motors or another microcontroller. The context diagram is very general. For example, analog sensors are simply documented as "providing data to" the DAQ unit. This generality is intentional so as to not limit oneself to a particular design solution.

Overall, I needed to ensure that the unit is easily manipulable and configurable by the user. Furthermore, the unit needs to take in data from both analog and digital sensors, communicating with them if needed. The data acquisition unit needed to be able to control an electric motor if necessary. Additionally, the system also needed to take in power from an external power supply. Finally, the system needed to communicate with a computer and possibly even an additional microcontroller if a situation warrants it.

2.3 Customer Affinity Process and Originating Requirements

With the context diagram in place, I had a better idea of what the system should look like. However, the specific requirements of the system still needed to be determined. Through

using an informal customer affinity process and creating a formal list of originating requirements, I had a list of requirements to benchmark by project against. In comparing my project with these requirements, I could determine whether my project was "successful" or not.

Typically in completing a customer affinity process, many comments (often thousands) are collected and organized into a hierarchy 3-5 layers deep. Gathering this information and organizing it in this way allows the customer's functional needs for the system to be more easily determined. Distilling these overarching functional needs allows the designers to bridge the divide between human language and quantitative elements that are more easily used to measure success in technical environments. Furthermore, determining these functional needs highlights areas were more detail from the customer is needed so that the system can meet their needs.

In the case of this project, a more informal structure was used. A couple meetings were held with myself, the drivetrain subteam, and the electrical subteam in order to determine what each group wanted out of the new data acquisition hardware. Using these comments from the customer affinity process, the following needs were determined:

- Configurable
- Adaptable
- Robust

With these needs determined I could go about creating a list of originating requirements. Originating requirement lists are numbered lists that document what the system will do using "shall" statements. Furthermore, any requirement must be unambiguous and verifiable via measurement. An abstract function name is also listed so that in further systems engineering tools used the entire requirement does not need to be written out. Originating requirements are those originally determined when designing the system. In using these tables, I was able to learn about the various conditions and requirements I needed to test and keep in mind when designing my system.

Resulting from discussions with the interested parties, many requirements for the system were considered. The group specified that the system communicate with a computer using serial to USB technology. This is because implementing native USB on microcontrollers can be difficult. Therefore, using serial communications was considered the best route to take. The group also specified that the board will have 16 ADC channels and 16 general purpose digital input/output pins. The number 16 was chosen as the team could not think of a situation where there would be more than 16 channels that needed to be measured or used.

The system also needed to measure the RPMs of both the engine and the flywheel used in tuning the CVT. The system was to provide 3.3V, 5V, and 12V sources of power that could be used to power sensors, motors, and the like. The system was to have a blinking LED on it so that the operator could tell if there was a problem with the board. A blinking LED would indicate that the system's code is running normally. If the board were resetting, the LED would not blink, indicating a problem. The group also specified that the system measure the analog channels with a 16 bit resolution.

Physically, the system was to have separate grounding planes for analog and digital components. Furthermore, it was to be mounted in the dynamometer enclosure using DIN rails. A very important physical characteristic that was specified for the project was the use of terminal blocks instead of proprietary connectors. The use of these blocks allows the user to easily change and reconfigure the sensors. Using terminal blocks also allows for different gauges and types of wire to be used, further decreasing the complexity of attaching a new sensor or part to the data acquisition unit.

There were also software specifications that were defined at the beginning of the project. For one, the system was to intake the number of trials to be run and the duration of each trial. Furthermore, the group specified that the system output data that was already averaged. Finally, the group also wanted the system to manage warm-up and cool-down times for the dynamometer, taking these parameters as an input.

During the course of the project, some of these specifications were reduced or removed altogether due to time, complexity, or overlap with someone else's work. A good example of this is the specification that the ADCs measure data with a 16-bit resolution. After discussing the specification with Bruce Land, it was decided that this resolution was "overkill" for this project. More specifically, the noise measured by 16-bit ADCs would be greater than the differences in measurements from the actual signal. Considering that the system is operating in a high-noise environment (thanks to the spark plug in the engine), this was definitely applicable to the dynamometer.

Furthermore, the signal processing requirements were removed as this was considered to overlap with Andy Michaels' data processing code on the receiving computer. The portions of the project to automate the testing using automatic runs were curtailed due to time and complexity constraints. The original plan was to include servo motors that would actuate the throttle and the the brake on the dynamometer automatically. Implementing and adding this automatic throttle and brake engagement would have prevented the team from using the dynamometer while the system was built and tested. Therefore, it was decided to forgo the automatic control of the dynamometer this time around. In fact, manual control has the added benefit of letting the operator vary the throttle and brake easily and in a variable way. Implementing quick throttle variability can actually be a complex problem to solve on

its own.

While some requirements were removed, other requirements for filters, checksums, and output strings were added. After further discussion and thought, the following list of originating requirements was used to benchmark the project:

Requirement #	Requirement	Alias
OR.1	The system shall communicate with a computer via serial to USB using an FTDI chip.	FTDI
OR.2	The system shall have 16 general purpose I/O and 16 analog inputs.	I/O
OR.3	The system shall measure the rotation rate of the engine in teeth/sample.	Engine Rotation
OR.4	The system shall measure the rotation rate of the flywheel in teeth/sample.	Flywheel Rotation
OR.5	The system shall pin-out all four external interrupts.	External Interrupts
OR.6	The system shall provide 3.3V and 5V power for digital and analog components.	Power
OR.7	The system shall be mounted to DIN rails.	Mounting
OR.8	The system shall include a blinking LED.	Status LED
OR.9	The system shall have separate ground planes for analog and digital components.	Ground Planes
OR.10	The system shall be mounted on a PCB.	PCB
OR.11	The system shall use terminal blocks to connect with other components.	Terminal Blocks
OR.12	The system shall measure analog signals with 12-bit resolution.	ADC
OR.13	The system shall have the ability to amplify analog signals from 5 to 1000 gain.	Amplification
OR.14	The system shall be able to low-pass filter analog signals with a -40dB/decade attenuation rate.	LPF
OR.15	The system shall include a checksum in its output string to the computer.	Checksum
OR.16	The system shall have a variable sample rate.	Sample Rate
OR.17	The system shall have the ability to turn data outputs from external interrupts and the ADCs on/off.	Configuration
OR.18	The system's configuration string shall be of the format "$100,0xFF,0xFFFF"	Config String
OR.19	The system shall use space characters to delimit the data sent to the computer.	Delimiters
OR.20	The system shall use a micro USB port to connect to the computer.	USB
OR.21	The system shall use a 12V power supply.	Power Supply
OR.22	The system shall sample and output data at least 20 times/second.	Min Sample Rate

Figure 3: List of Final Originating Requirements

3 Range of Solutions

3.1 Use of a Commercial Data Acquisition Solution

In designing the system, there were a few opportunities where design choices needed to be made. In implementing this project, one simply could have used a commercial data acquisition solution such as one from National Instruments. While these are very nice, they are also very expensive. Furthermore, the Baja team wanted direct control over the hardware and software used to implement the data acquisition unit. This would not be possible using a commercial solution as the hardware and software are proprietary. Therefore, for cost and adaptability reasons, the team and myself decided to go the route of an in-house data acquisition solution rather than buying one on the market.

3.2 Adjustable Resistances and Capacitances

One major design decision was determining how to implement the selectable gain and filters on the amplification board. The gain for the instrumentation amplifier is set by a single resistor. For an easily user-selectable gain, this problem begs to use a potentiometer. While this will work, one must keep in mind where the amplification board will be. The board will be subjected to vibrations from the engine which could cause the screw on the potentiometer to move. Furthermore, potentiometers do not have a very high accuracy. This means that every time the potentiometer is adjusted, it will need to be checked with an ohmmeter before proceeding. Similarly to the instrumentation amplifier, the low-pass filter discrete components need to be adjustable. Another solution to address this problem would be to use header pins and "adapters" containing single resistors that could be plugged into these pins. This is a novel idea (thanks to Bruce Land) that makes switching out resistors and capacitors very easy. On the other hand, one must solder components to these header adapters. On examining the headers and through-hole components, I felt that it would be tricky to solder these well. Furthermore, one must keep a bank of resistors and capacitors on hand to change these out.

After discussing these solutions with my teammates, I decided to leave simple solder pads on the amplifier board for the resistors and capacitors needed to adjust the gain and filters. There are a couple reasons for this decision. One, we decided that the sensors attached to these boards would not change that often, making adjustable resistors and capacitors more of a luxury than anything. If new sensors are added to the system, a new channel on an amplifier board will be populated for it. Secondly, the use of discrete resistors and capacitors lets the team use higher tolerance parts than they would have been able to otherwise. For these reasons, I felt that this was the best decision.

4 Design & Implementation

4.1 Overview

The design of the dynamometer data acquisition unit consists of two systems that work in concert to measure, process, and send data. The hardware, consisting of the signal amplifiers and data acquisition electronics, are essential for gathering the data that the team needs. By the same token, the software is just as important for this system. The embedded software in the DAQ sends commands to the ADCs to read values, parses the data, and transmits it via UART to a workstation for further analysis. In the following sections, the design and implementation of both the hardware and the software will be discussed in detail.

4.2 Hardware

The hardware that makes up the DAQ is made up of several subsystems:

1. Power Subsystem
2. Processing Subsystem
3. ADC Subsystem
4. Connector Subsystem
5. Communication Subsystem
6. Amplification Subsystem
7. Sallen-Key Low-pass Filtering Subsystem

Annotated pictures of the PCBs are included in the appendix of this report as figures 12 and 13. Both the amplifier and data acquisition PCBs were created using Altium Designer. The low-pass filtering and amplification subsystems are present only on the amplifier board. The processing subsystem, ADC subsystem, and communication subsystem are only on the data acquisition board. Both boards contain power and connector subsystems. While one data acquisition board is needed, the user can create as many amplifier boards as needed since each only has 4 channels.

4.2.1 Power Subsystem

As specified in the originating requirements, separate power supplies for the 5V digital, 5V analog, 3.3V digital, and 3.3V analog needs on the data acquisition board were created. That way interference could be avoided, especially between the digital and analo g components and even between components operating on the same ground plane. On the amplifier board, only a single 5V supply is needed to power its components. Two sets of ground planes were also created. One is for grounding the digital electronics on the board and takes up the lower half of the data acquisition hardware board. The other is for the components responsible for analog measurements (ADCs and voltage references). This ground plane takes up the upper half of the data acquisition board. The ground planes on the data acquisition board are connected with three 22 ohm ferrite chokes. Similarly, the amplification board employs a ground plane for all of its components.

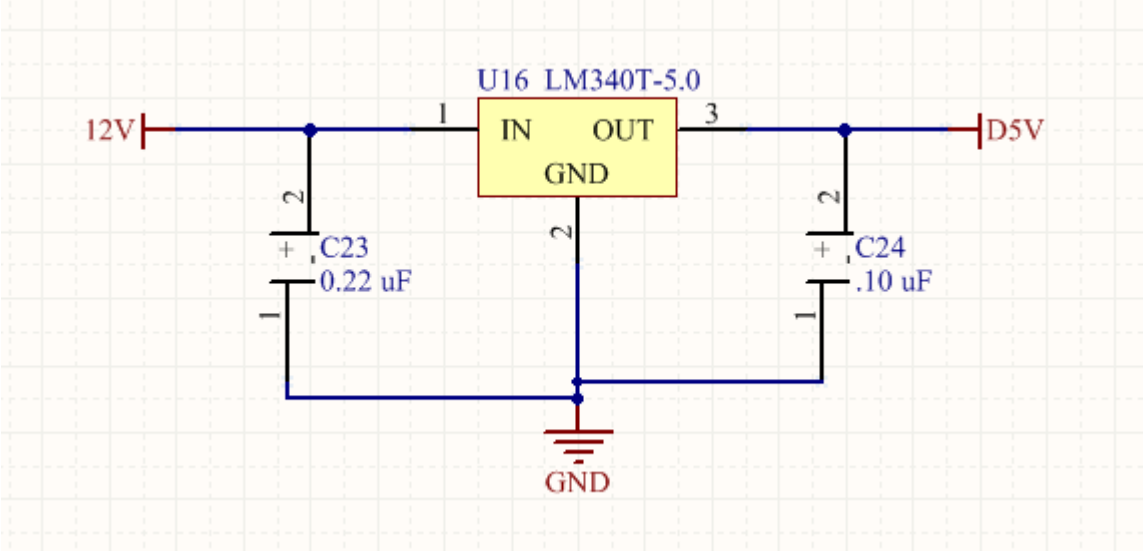

Figure 4: Schematic for LM340T LDO

National Instruments' LM340T linear regulators are used to meet the 5V digital and analog needs on the data acquisition board and amplifier board. These linear regulators are connected to the 12V power supply and step the voltage down to 5V. Each linear regulator has 3 pins, one for ground, one for input power, and one for output power. A .22 microfarad ceramic capacitor is attached from the 12V pin to the ground plane, while a .10 microfarad capacitor is connected between the 5V output pin and the ground plane. These capacitors ensure that linear regulators operate within their stability regions. Furthermore, the capacitors act as low-pass filters that remove spurious noise from the power source.

Texas Instruments' TL1963A LDOs are used to supply the data acquisition board with 3.3V power. A LDO is each allocated for the digital components and analog components. Much like the LM340T, the TL1963A also uses three pins, except it outputs 3.3V rather than 5V. Furthermore, the TL1963A is a SOT surface mount IC, unlike the through-hole LM340T. This means that the heat sink on the TL1963A can be soldered to the copper grounding plane on the PCB, making an excellent sink for the heat it generates. Similar to the LM340Ts, the TL1963As use 10 microfarad decoupling capacitors between the 12V input and 3.3V output.

From these power regulators, all power is sourced on both the amplifier and data acquisition boards. For ease of use, it is necessary to determine when there is power present on the board. Red chip 1206 LEDs are attached to the 5V digital, 5V analog, 3.3V digital, and 3.3V analog supplies. 1206 resistors of 499 Ω are put in series with these LEDs to limit current flow. This simple addition allows the user to quickly determine whether all parts of the boards are receiving power. Furthermore, the widths of the power traces on the PCB have been made as wide as the receiving component will allow. This allows more current to

flow through the traces and prevents overheating during use.

4.2.2 Processing Subsystem

The microcontroller chosen for this project was the NXP LPC1769. The LPC1769 is a 32-bit ARM Cortex-M3 based microcontroller that the Baja SAE team has used for the past couple years. Some features include the ability to communicate via the CAN protocol and Ethernet, which are overkill for this project. However, the LPC1769 does have the ability to address 4 UART channels and potentially 3 SPI channels. Another very useful feature is the microcontroller's ability to send PWM commands, which could be used to control devices such as servo motors.

In laying out the necessary circuitry and components for the microcontroller's operation, an oscillator and necessary capacitors were placed. The LPC1769 contains a phase-locked loop that converts the input from the 12 MHz oscillator up to 120 MHz for use in clocking the microcontroller. A 12 MHz crystal has been placed on the board along with 18 picofarad capacitors attached from each end of the crystal to ground. Noise considerations from the power supply are also taken into account. .10 microfarad capacitors are placed near each of the microcontroller's power input pins.

Of utmost importance is the programming port for the JTAG interface as the project will not work without it. This has been placed near the microcontroller and routed to the proper pins.

4.2.3 ADC Subsystem

The ADC subsystem of the data acquisition board contains two main parts. One part consists of the ADCs themselves, Analog Devices AD7927's, and the other part consists of the voltage references necessary for correct ADC measurements. For this project, the AD780 from Analog Devices was chosen. The AD7927 is an 8 channel successive approximation ADC with 12 bit resolution. The part operates from the 5V analog power supply on the data acquisition board. .1 and 10 microfarad capacitors are connected to the ADC voltage supply and supply from the MCU. This is to remove any transients or noise that may be on the power lines. A benefit of this ADC is that it can operate its digital inputs and outputs at 3.3V while operating overall at 5V. This enables the ADC to read analog channels varying in voltage from 0 to 5V while sending the data describing these channels at 3.3V back to the microcontroller. While the pins on the MCU are 5V tolerant, it was decided that operating in the 3.3V regime would be better from a hardware resiliency standpoint.

The ADC communicates with the microcontroller via the Serial Peripheral Interface Bus

(SPI). The SPI is an interface that facilitates communication between devices using 4 wires. SPI operates synchronously and bidirectionally. The first wire, SCLK, is used to synchronize data being sent over the data lines (MOSI and MISO). When data needs to be sent or received, square wave pulses are sent down the wire with the number of pulses corresponding to the number of bits being sent or received. These pulses tell the microcontroller and ADC (or other device) when to read the bits on the MISO or MOSI lines. The Master Out-Slave In (MOSI) line is the wire that is used to transmit bits from the master to the slave. In the configuration of this project, the master is the microcontroller and the slaves are the ADCs. The Slave Out-Master In (MISO) is used to transmit data bits from the slave back to the master. Finally, the Chip Select (CS) is used to select the chip with which the master is communicating. Unlike the other wires that are active high, the CS is active low. When communication is to be initiated, the line is pulled low for the duration of the communication. The CS returns to the high position once the communication has completed.

Each channel (0-7) of each ADC is routed to a connector on a terminal block at the top of the data acquisition board. These connectors will be discussed later in another section.

Each ADC uses an AD780 voltage reference. The voltage reference's output connects to the ADC's REFin pin and supplies a 2.5V reference source for the ADC. The voltage supplied to the REFin pin must be held at 2.5V with no more than $\pm 1\%$ deviation. The reference's job is to supply this precise voltage. Each ADC has its own dedicated reference. There are several decoupling capacitors connected to the reference to isolate it from the noise that can prevent it from outputting the reference accurately. A 1 microfarad decoupling capacitor is connected between the analog 5V power source and analog ground. There is also a 100 microfarad capacitor connected between the output of the voltage reference and the analog ground. Finally, a 100 nanofarad capacitor is connected between the temperature output pin and analog ground as the output is not used in this project. Tying the pin to ground prevents transients and other noise from interfering with the reference's operation.

Two AD7927 ADCs were cascaded to meet the 16 ADC channel specification in the originating requirements. Performing this cascade was relatively simple thanks to the way SPI operates. Both ADCs share the same SCLK, MOSI, and MISO channels. The CS line is the only part of the physical SPI specification that is different between the two. To access the first ADC which monitors channels 1-8, the pin GPIO 0.19 is pulled low. To access the second ADC, which monitors channels 9-16, the GPIO 0.20 pin is pulled low. The specifics of the commands sent over SPI to configure and read data from the ADCs will be discussed in greater detail in the software implementation section of this report.

4.2.4 Connector Subsystem

It was decided to use terminal blocks for all external connections that needed to be made with the boards. The terminal blocks facilitate easy and secure connections without the use of additional connectors. In essence, a terminal block is a wire clamp. There is a screw on the top of the terminal block that tightens the clamp. This behavior is desirable for a couple reasons. Firstly, it enables the operator to insert more than one wire into a connector. This is especially valuable for someone interested in attaching an oscilloscope to the system or powering multiple devices from the board. Secondly, the use of terminal blocks removes the need to attach connectors to wires that need to be attached to the system. One simply inserts the wire and screws the terminal block tight. Finally, the use of terminal blocks enables the operator to forgo worrying about different wire gauges. Some sensors might use different wire gauges than others, and the terminal blocks allows the operator to connect these sensors the same as others.

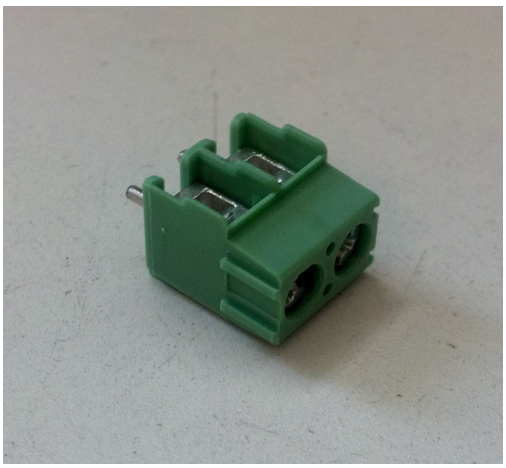

Figure 5: 2 Pin Terminal Block

Transient voltage suppressors are placed in line with the digital GPIO pins that are pinned-out on the data acquisition board. Semtech SRDA05-4 ICs are used to accomplish this voltage suppression. Each IC contains a network of diodes that steer the current to ground in an overvoltage or transient situation. This particular IC contains 4 channels and triggers for voltages above 5V. 5V was chosen as the MCU cannot tolerate voltages above 5V.

4.2.5 Communication Subsystem

To make the data acquisition useful for the team, the board needs to send the data gathered from the ADCs and other sensors to the computer for further processing. The previous DAQ made use of the USB HID drivers to communicate with the computer via native USB.

However, these drivers could be temperamental and tended to communicate slowly. Due to the challenges present in implementing native USB support, it was decided to use a serial to USB solution.

Serial to USB ICs enable the designer to use universal asynchronous receiver/transmitter channels to send data over USB to a computer. The serial to USB IC takes care of the drivers and communication standards for the designer, emulating a serial port on the computer. This lets the designer concentrate on sending and receiving data rather than the communication protocols necessary to use USB.

The IC chosen for the project was FTDI's FT232RL. The chip automatically configures itself for baud rates up to 3Mbaud and can source its own power from the USB port. This means that the FT232RL is a very easy device to use. The FT232RL also operates from 3.3V to 5.25V, meaning that it will work perfectly with the LPC1769, which operates at 3.3V. One simply connects components to it in the manner described in the datasheet and it just works.

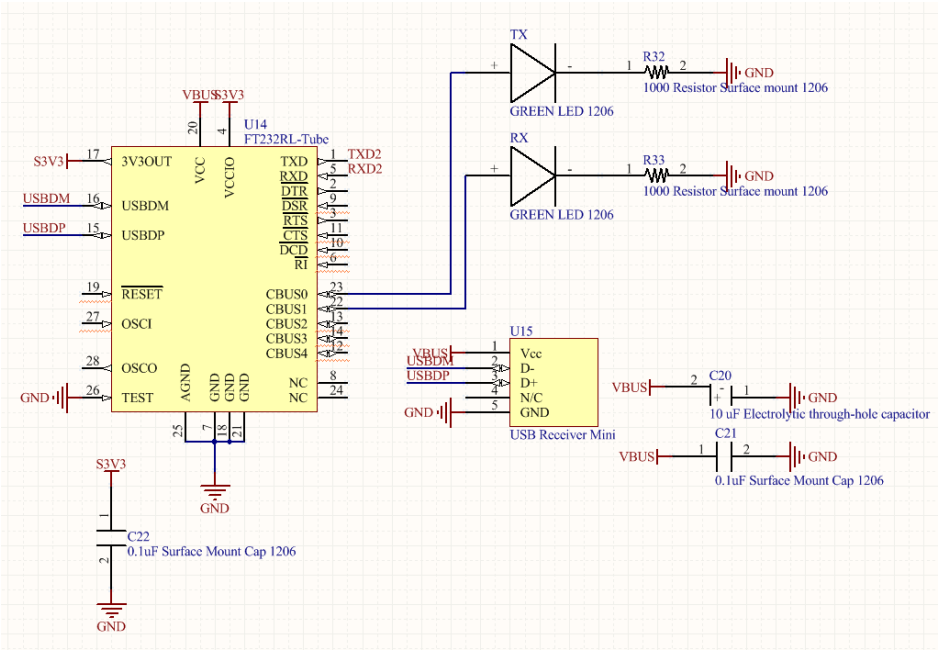

Figure 6: Serial to USB Schematic

In the configuration on the data acquisition board, the FTDI sources its own power from the USB port. This means that, apart from the TX and RX pins connected to the microcontroller, it is on a separate circuit from the board. The FTDI does share the same ground as the rest of the components on the board so that there is no mismatch in voltage levels. The TXD pin of the IC is connected to the RXD2 pin on the MCU. Likewise, the RXD pin of the IC is connected to the TXD2 pin of the MCU. These pins are responsible for duplex communication between the MCU and IC.

The USBDP and USBDM pins are connected to the data pins on the mini USB port. These pins carry the transmitted and received data. A mini USB port was chosen since it would fit nicely on the board and cables are easily accessible. 3V3OUT provides a 3.3V source of power for the chip. This pin is connected to VCCIO so that the TXD, RXD, and CBUS pins may be driven at this voltage as well. The presence of the VCCIO pin allows the designer to vary the operating voltage of these lines for different logic level needs. Finally, red and green LEDs are connected to the CBUS0 and CBUS1 pins to blink when data is sent over the TX and RX pins respectively. The power supply pins (VBUS and 3v3OUT) both have decoupling capacitors to connected to ground to prevent noise from interfering with the operation of the FTDI chip.

When the data acquisition board was first populated there was a peculiar problem with the FTDI chip. Data was being sent to the chip from the microcontroller, but the chip was not passing the data on to the computer. Strangely, the computer was recognizing the FTDI chip normally and communicating with it successfully when the TXD and RXD pins were shorted together to form a loopback device. Some scoping and examination of the schematics determined that the TXD2 pin on the MCU was connected to the TXD pin on the chip. This was also the case for the RX pins. In this configuration, both the MCU and FTDI chip were sending data to one another but neither was reading data. The TX and RX traces between the MCU and chip were cut and flipped using magnet wire. This ensured that TX on one went to the RX on the other. Once this was completed, the board started communicating with the computer correctly.

4.2.6 Sallen-Key Low-pass Filtering Subsystem

In order to remove noise from the outputs of amplified sensor data, it is sometimes necessary to employ a low-pass filter in order to attenuate high frequency signals that may accompany the signal we care about and to prevent aliasing. A passive low-pass filter, consisting of a simple resistor and capacitor ideally has a gain of 1. In practice, the gain is less than 1 since it is an impedance to the signal passing through it. While this would be fine for simple applications where the exact voltage value of a signal is not important, it is not practical for accurate data collection. Rather, a unity gain must be maintained for signals that are lower than the chosen cutoff frequency for the low-pass filter. A simple active low-pass filter is also not desirable because it does not attenuate frequencies above the cutoff frequency quickly enough. A simple single pole LPF attenuates signals at -20dB/decade on a logarithmic frequency scale while a dropoff of -40 or even -60dB/decade would be desired.

After some research, I decided to go with the Sallen-Key topology for an active low-pass filter:

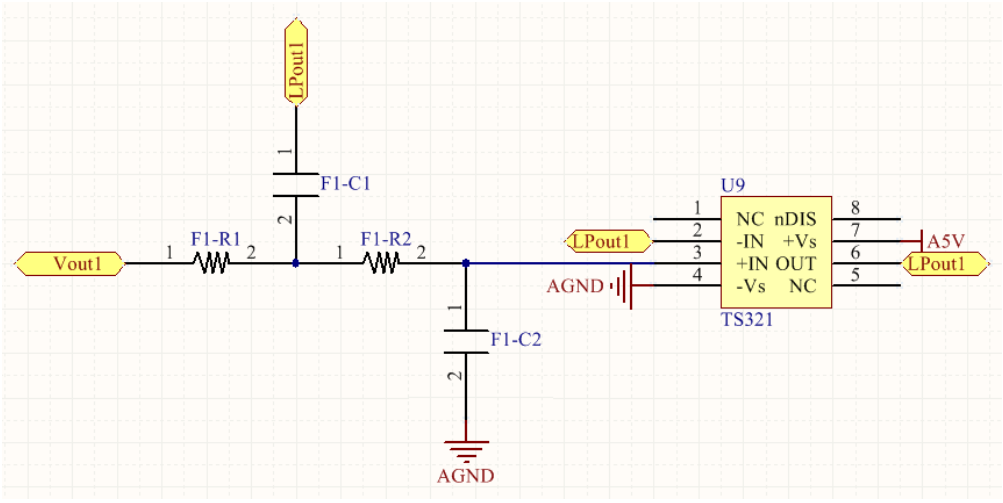

Figure 7: Sallen-Key Low-pass Filter

The Sallen-Key low-pass filter meets the requirement that the signal not be attenuated below the cutoff frequency as it provides unity gain via the direct connection between LPout1 and -IN on the operational amplifier. Furthermore, the topology attenuates signals above the cutoff frequency rather quickly at -40dB/decade thanks to the two poles introduced by the two capacitors connected to LPOut1 and AGND. The other end of these capacitors are connected to the +IN input on the TS321 op amp.

In using the the LPF, one needs to calculate the correct resistor and capacitor values to set the cutoff frequency. The transfer function of the Sallen-Key LPF is[1]

$$H(s) = \frac{\omega_0^2}{s^2 + 2\alpha s + \omega_0^2}$$

with the cutoff frequency described by[1]

$$\omega_0 = \frac{1}{\sqrt{R_1 R_2 C_1 C_2}}$$

and a Q factor of[1]

$$Q = \frac{\sqrt{R_1 R_2 C_1 C_2}}{C_2(R_1 + R_2)}$$

Plugging in $\omega_0 = 2\pi f_0$ and setting $Q = 1/\sqrt{2}$ to get the flattest response for frequencies below the cutoff, the designer can plug in chosen values of the resistors or capacitors to determine the values of the other components. Using a Q of this value creates a Butterworth filter[1].

Power-wise, the TS321 operational amplifier operates in the single-supply regime from 0 to

5V. This means that signals that are passed through the filter must be between 0 and 5V, or else the signal will be clipped. If the voltages going into the op-amp exceed 35V, the op-amp will be destroyed. However, the chances of a signal this strong going into the inputs of the op-amp are very low.

Initially, a different operational amplifer than the TS321 had been chosen for the Sallen-Key filter. The Baja team had plenty of OPA847s in its inventory so I decided that it would be a good choice to use in the filter. What I did not know is that the OPA847 is unstable at unity gain. Had I read the datasheet more closely, I would have seen this right away. Instead I struggled to determine why the output of the filter was so noisy and unpredictable. After a couple weeks, I asked Bruce Land about the problem and after a few minutes he determined that this was the case. This was definitely a humbling experience and just goes to show how important reading the datasheet closely truly is.

4.2.7 Instrumentation Amplifier

The instrumentation amplifier on the amplification board is responsible for taking a signal and amplifying it. Often times, the signals sensors output are in the mV level. While this is adequate to see with an oscilloscope, voltages of this level are particularly hard to resolve with ADCs. Therefore, instrumentation amplifiers are often used to bump up these signals. Instrumentation amplifiers are easy to use, only requiring a single resistor to set the gain of the amplifier. The value of the resistor can be calculated from the formula presented in the datasheet.

The INA827 from Texas Instruments was chosen as the instrumentation amplifier. The INA827 can provide gains from 5 to 1000 and can operate with the single-side power supply on the amplification board. For a given input with the part operating at 5V, the INA827 can output voltages from -0.2V to 4.1V. While this is a larger range than other instrumentation amplifiers, it is still possible to saturate the output if one is not careful in determining a gain to use for a given input. To calculate the resistor needed for a certain gain, the operator can use the following relation taken from the INA827 datasheet:

$$R_G = \frac{80000}{G - 5}$$

where R_G is the resistor value to use in Ohms and G is the desired gain. The output of the INA827 is sent both to a terminal block as an "unfiltered" output and to the Sallen-Key low-pass filter for filtering. The user may chose from one or both outputs when connecting them to the inputs of the ADCs on the data acquisition board.

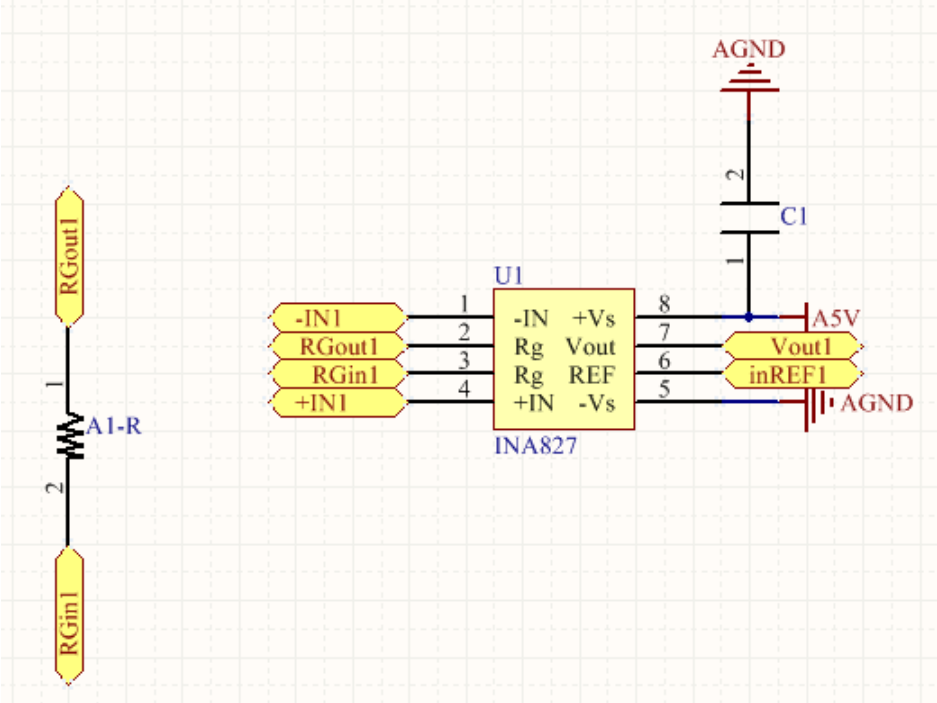

Figure 8: Instrumentation Amplifier Schematic

Connected to the REF (reference) pin of each INA827 is a jumper switch. With this switch, the user may decide whether the reference voltage is 0V or 2.5V. This is useful for offsetting the output signal by 2.5V. To accomplish this, the outputs from a REF3225 from TI were connected to one pin of the jumper while the other pin was connected to ground. The middle pin is connected to REF. Whichever pin is connected to this pin determines the offset used. Like other devices in this project, the REF3225 uses a .47 microfarad decoupling capacitor to remove noise and transients from its input power supply.

Interestingly, the amplifier board was shorting when it was first powered on. Some probing and examination of the schematics determined that the REF3225 was not, in fact, connected to ground. Some magnet wire was soldered between the ground pin on the voltage reference and ground to establish a connection. Once this was complete, the board did not short anymore.

4.3 Software

The software running on the microcontroller is essential for the operation of the data acquisition unit. Without the software, the project will be almost totally useless to the team. The software consists of 6 parts: initialization, the main `while` loop, DAQ configuration, calculating RPM data, gathering ADC data, and sending data over the UART. This order

also follows the flow of the program itself.

4.3.1 DAQ Initialization

When the DAQ board is first powered on, the function `timer_init()` is run. This function enables both timers 0 and 1 in the LPC1769. Timer 0 is used to blink the `MCU_LED` on the DAQ board as a check of MCU functionality. As such, the value 16666 is written to the match register (`MR0`) register. At this value with a clock divisor of 1 and prescaler of 1000 (set in the `LPC_TIM0` register) the timer will interrupt every 500 ms. Timer 0 is set to interrupt and clear on match by writing to the appropriate bits in `LPC_TIM0`. Finally, the interrupt for the timer is enabled and the counter is enabled, thus fully enabling the timer. Timer 1 is also initialized with the same parameters, except the `MR1` register is dependent on the data received from the configuration string. Specifically, the `MR1` register is set to `(int)(33.33*samplePeriodInt)` with `samplePeriodInt` derived from the configuration string. This adaptability allows the user to modify the sampling rate of the data acquisition unit.

After the timers are initialized, the external interrupts for gathering RPM data need to be initialized as well. This is accomplished though `eint_int()`. Each of the four interrupts are initialized in the same fashion, so only one will be covered here. First, each of the pins associated with the external interrupts need to be set to external interrupt mode. This is accomplished by writing `0b01` to the `PINSEL4` register. The interrupt flag is cleared for for good measure and the interrupt is set to trigger on the falling edge of a signal by setting bits in the `EXTMODE` and `EXTPOLAR` registers. Finally, the interrupt is enabled and the priority of the interrupt is set. The procedure is the same for each of the four external interrupts on the LPC1769.

With the external interrupts initialized it is time to initialize the UART. Specifically, the DAQ board uses UART2 to communicate with the data collection computer. The UART communication code in this project is Brian Toth's own work for the Baja team with some modifications of my own. UART2 is initialized as follows: Firstly, UART2 is powered on by setting a bit in `PCONP`. Next, the peripheral clock and pins are set by writing to `PCLKSEL1` and `PINSEL4` respectively. UART2 is then configured to send data in 8 bits with no parity and 1 stop bit by setting `DLAB = 1` and writing to `LCR`. Furthermore, the baud rate is set to 115200 baud by writing to the `FDR`, `DLM`, and `DLL` registers. The content of these registers is determined by a rather complex algorithm in specified in the datasheet. I have included the code for this algorithm in a Python script in the appendix. With those registers set, the `LCR` register is rewritten to set `DLAB = 0` since access to the registers to set the baud rate is no longer needed. The RX and TX queues are enabled and reset, and the Receive Data Enable Interrupt and the Transmit Holding Register are enabled. Finally, UART2's interrupt is enabled. When the interrupt occurs, `receiveConfigData()` is called.

Once UART2 has been initialized, the pin connected to `MCU_LED` is set to GPIO, the direction set to output, and the LED is turned on. Furthermore, the SPI interface is enabled via the function `SPIInit()`. This and other SPI functions were provided by Code Red Technologies under the GNU LGPL v2.1 license. I was able to simply port it into my project with no real modifications needed aside from changing some pins.

With the above peripherals initialized, the program is ready to proceed into the infinite while loop and wait for a configuration command from the user/data collection computer.

4.3.2 Main While Loop

The DAQ only starts printing to the screen once a configuration string has been received and processed. In the `while(1)` loop, this is checked by checking the status of `configStatus`. If it is 1 then the DAQ has received a configuration string and is free to print data. Nested in the if statement is a check for `configDataReady`. If it is set to 1 then a new configuration is ready to be implemented. `DAQ_Config()` is called to parse the configuration string into integer variables that are used to control the DAQ behavior. Timer 1 is disabled and reset so that the new sampling period can be written to `MR0`. The new `MR0` values is written and the timer is restarted. Finally, the `configDataReady` flag is set to 0 so that the configuration is not reset again until another configuration string is sent.

The `rpm_flag` variable is set to 1 when the timer 1 interrupt is triggered. This variable is checked in the `while(1)` loop. When set to 1, the allotted time has passed and a new data sample is generated and sent over UART. The number of teeth seen since the last data sample is calculated with `calc_rpm_data()` and the ADCs are sampled with `sampleADC()`. The data gathered from the previous two functions is parsed into a string to be sent over UART with the `asciiSensorStatus()` function. Finally, the data is sent via UART using Brian Toth's `UART2_PrintString()` function. Further action is prevented until UART2's buffer is cleared. If the buffer is not cleared before writing a new data set, strange errors can occur in the output. The main functions used in the `while(1)` loop will now be covered.

4.3.3 DAQ Configuration

The configuration string and characters can be received asynchronously as receiving data via UART is interrupt based. This means that the user/computer can reconfigure the DAQ even while data samples are being taken. When `receiveConfigData()` is called by UART2's interrupt, it checks to see if a configuration string is already in the process of being written. If '$' has been detected already, it records the characters received. Otherwise the characters are not recorded. The function checks if the received charac-

ter is '$' so that the configStringFlag may be set for data to be stored. Likewise, if '#' is detected the configStringFlag is set to 0 so no further characters are recorded. configDataReady and configStatus are both set to 1 so that the while(1) loop may start printing data. receiveConfigData() also checks for two special characters. If '@' is detected, configStatus is set to 0 so that the recording of data stops. On the opposite side, '&' sets configStatus to 1 to restart data collection and reset the time counter.

Now that the configuration string has been read and stored in the array rcvData, it is time to parse the data into data structures useable by the rest of the program. This is where DAQ_Config() comes into play. It is only run when configDataReady is set to 1 in the main while loop. The function takes in rcvData and parses it using the C function strtok() into its constituent parts. The string is broken up into three character arrays: samplePeriod, eintStatus, and ADCStatus. atoi() is used to convert samplePeriod into an int. Likewise, (int)strtol() is used to do the same for eintStatus and ADCStatus. The resulting variables integer conversions from these variable conversions are samplePeriodInt, eintStatusInt, and ADCStatusInt.

The strings DAQ_Config() receives are in the following format: $5,0x0A,0xAAAA# where the first number is the time in ms between samples. The second and third values are hexadecimal values that specify which of the 4 EINT and 16 ADC channels are enabled, respectively. These values are big-endian in the sense that the 16th ADC channel is the binary value furthest to the left. In the example above, every other EINT and ADC channel is enabled. The extra 0 in the EINT byte is simply there to make a complete byte and can be filled with whatever the user chooses.

4.3.4 Calculating RPM Data

When calc_rpm_data() is called in the while loop, the current counts of each external interrupt are placed in the DynoData struct's rpm0, rpm1, rpm2, rpm3 fields, which are all 32-bit unsigned integers.

Of course, this function is dependent on what happens when an external interrupt is triggered. When each interrupt is triggered, the bit holding the flag is cleared and the count is incremented by 1. The interrupt is intentionally simple in order to prevent timing errors. In general, the fewer commands running in an interrupt handler the better. When timer 1 matches the value in MR1, the sampling period has been met. Subsequently in the timer 1 interrupt handler these counts (count0, count1, count2, count3) are assigned to the calc_count variables which are sent to the computer via UART. Furthermore, the interrupt flag is cleared and the rpm_flag is set, signaling to the main loop that another round of data can be gathered and sent to the computer via UART. Finally, the count variables are

cleared so they are ready to collect a new sample of teeth counts.

4.3.5 Gathering ADC Data

Now that the values of the teeth counts have been gathered the system moves on to sample channels of the ADCs. This is done through the function `sampleADC()`. The function addresses both ADCs using SPI, sampling only from the channels specified in `ADCStatusInt`. A `for` loop iterates through the 8 channels of the first ADC. In each loop, the status of the channel is checked by looking at the particular bit corresponding to that channel in the 16-bit integer. If the value for that bit is 0, no further action is required and that channel is not read. Otherwise, two bytes are sent to the ADC via SPI as the ADC requires 12 bits of data using `SPISendN()`. At each SPI send, the `WRITE` bit is high along with the two Power Management bits, specifying normal operation. Bits 6-8 change depending upon which channel needs to be sampled. Furthermore, the `RANGE` bit is set low so that the ADC measures from 0 to 5V and the `CODING` bit is set high so the conversion is to straight binary. Between each channel, the only values that change are the bits specifying which channel is to be read. Due to this, an 8x2 array `send` was populated with the bytes already coded. With these "preprogrammed" one simply needs to iterate through the array to address the different channels. That is exactly what is done here. The SPI data is clocked back in through the use of `SPIRecvN()` from Code Red Tech. The two bytes need to be put into a useable form in one variable rather than two separate bytes. To solve this problem, the first 8 bits are left-shifted by 8 as they are the most significant. This 16-bit variable is then operated on by a logical and mask: `0x0FFF` to correctly reflect the data as the 12-bits output by the ADC. The result is stored in the `ADCdata` array.

This same procedure is repeated for the 2nd ADC, which addresses channels 9-16. Once this is complete, each value in the `ADCdata` array is assigned to the corresponding variable/channel in the DynoData struct.

4.3.6 Sending Data over UART

With all of the data collected, it is time to create the string of ASCII characters that will be sent via UART to the computer. To accomplish this, the program makes use of the `asciiSensorStatus()` function. This function creates the string and a checksum that can be used to check the integrity of the string on the other end.

The function starts by clearing all of the bytes in the `datastr` array by using the `memset()` function. An array is then created that places all of the variables in the `DynoData` struct into a single array. This makes it much easier to iterate through and process. The `time`

variable is incremented by `samplePeriodInt` as a sample occurs every `samplePeriodInt`. This variable is printed to a buffer variable using `sprintf()` and then concatenated into the `datastr` array using `strcat()`.

Once that has been completed, the function then iterates through the lower four bits of `eintStatusConv`. If the bit is high, the value of the interrupt counter is concatenated onto the `datastr` array. Otherwise, the data for that counter is skipped. After each value a space is inserted as a delimiter. A similar procedure is performed with the ADC data. An `if` statement is iterated over the bits of the 16 bit `ADCStatusConv` integer. If a bit is high, that channel is on and its results need to be printed. The data from that ADC channel is written to a buffer which is then concatenated onto `datastr`.

With the data now recorded to `datastr`, a checksum is created to guard against data corruption. Operating at 115200 baud can generate transmission errors. Therefore it is important to generate a procedure that the receiving computer can use to ensure that it does not have a malformed data string. To compute the checksum, each byte of the `datastr` array is XORed together. This checksum is easy to compute and very fast. One could naively send this character generated from the checksum via UART and it might work. However, terminal applications such as PuTTY take into account serial control codes. If the checksum happens to match one of these control codes then the data recording could stop or produce an unintended output. To prevent this, the checksum is brought into the range of "printable" ASCII characters. Firstly, the checksum is split into two nibbles of 4 bits each. The first nibble with the upper bits of the checksum is placed into the lower bits of `checksumByteOne` while the second nibble with the lower bits of the checksum is placed into the lower bits of `checksumByteTwo`. Each of these two bytes has their sixth bit set to 1. This ensures that the range of printable ASCII characters corresponds to decimal values 64 to 79 (@ to F). Lastly, these bytes are concatenated onto `datastr` along with newline and carriage return characters.

Finally, the data is sent over the UART via calling `UART2_PrintString(datastr)` in the main while loop. `while(UART2_is_not_Ready())` is executed right after to ensure that all of the data has been clocked out of the serial data register before continuing. Without this command the microcontroller had a tendency to execute UART writes too soon, calling a new one before the old data was finished being clocked out. Figure 14 in the appendix provides a screenshot of sample output from the DAQ.

5 Results

Pictures of the testing setups are included in the appendix as figures 15 and 16.

5.1 Testing the External Interrupts/RPM Sensors

In order to test the data acquisition board's external interrupt capabilities an oscilloscope and function generator were needed. A Tektronix CFG280 was connected to the EINT3 pin on the DAQ board. A 3 volt peak-to-peak square wave signal was generated and sent to this pin in order to confirm the board's external interrupt reading capabilities. Furthermore, a Tektronix TDS210 oscilloscope was attached to the EINT3 pin to check the output of the function generator.

The board will need to read RPM values between 0 and 4000 RPM as the engine itself is governed to 3800 RPM due to competition rules. Therefore, the plan was to sample the counts from the external interrupts at 0, 500, 1000, 1500, 2000, 2500, 3000, 3500, and 4000 RPM. To check that the data could be read accurately for various sampling rates, the counts from the external interrupts were sampled at periods of 5 milliseconds, 10 milliseconds, and 100 milliseconds. For reference, the current DAQ board in the dynamometer measures data at a period of 100 milliseconds. The data from the experiments is as follows:

	5ms/Sample		10ms/Sample		100ms/Sample	
Generated ±1%	Teeth	Calculated	Teeth	Calculated	Teeth	Calculated
0	0	0	0	0	0	0
500	3	600	5	500	49	490
1000	5	1000	10	1000	99	990
1500	8	1600	15	1500	149	1490
2000	10	2000	20	2000	198	1980
2500	12	2400	25	2500	248	2480
3000	15	3000	30	3000	298	2980
3500	18	3600	35	3500	348	3480
4000	20	4000	40	4000	398	3980

Table 1: Sample Output from External Interrupts

The current number of teeth on the gears the hall effect sensors use on the real dynamometer is 60. This makes the calculation of the RPM from the teeth/sample parameter very simple. Above, the calculation is simply the $\frac{teeth/sample}{samplerate}$.

From looking at the data, one notices two things. The first is that as the sample period decreases (frequency increases), the resolution of the measurement decreases. For example, the 3500RPM only generates 7 teeth counts every 2ms, while the same RPM generates 348 teeth counts every 100ms. This intuitively makes sense as the longer the time that passes between samples, the greater the number of teeth that pass by the hall effect sensor over that period of time. The second thing the reader will notice is that the calculated RPM values do not always match the generated RPM values. This is the case for a couple reasons. Firstly, the frequency generator is not perfect in generating a signal nor is the oscilloscope

in measuring a signal like this with accuracies around ±1%. This causes perturbations in the signal read with the MCU. In fact, the MCU could be accurately measuring those perturbations. Secondly, the number of teeth measured is a discrete value. This creates a loss in resolution where RPM values between certain teeth counts/sample are interpreted one way or the other. For example, 399 could be measured when 400 was sent due to this loss in resolution. This testing confirms that the data acquisition unit can measure revolutions per minute accurately.

5.2 Testing Frequency Response of Low-pass Filter

Part of the amplifier board's capabilities is to provide low-pass filtering of the output of the instrument amplifier. This is useful for suppressing high frequency noise that may corrupt the signal before it reaches the ADCs and preventing aliasing. To test the frequency response of the filter, a Tektronix CFG250 waveform generator was to the input of the first channel on the amplifier board. A gain of 5 was set on the instrument amplifier by not installing a resistor. In order to simulate a reasonable LPF configuration, I decided to filter frequencies lower than 3800 RPM. This would be useful for filtering out the noise generated from the spark plugs firing.

Frequencies above

$$\frac{3800\,revolutions}{minute} * \frac{minute}{60\,seconds} = 75\,Hertz$$

should be attenuated for this configuration. It was decided to double this frequency to 150 Hertz for testing purposes. To meet this requirement resistor and capacitor values needed to be chosen. Using $R_1 = 7.5K$ ohms and $R_2 = 7.5K$ ohms with $C_1 = .2$ microfarad and $C_2 = .1$ microfarad to achieve a Q-factor of a Butterworth filter, one can plug these values into the following relation that describes the undamped natural frequency of the Sallen-Key low-pass filter. At this frequency, the frequency response starts to decay at -3dB/decade before proceeding to -40dB/decade.

$$2\pi(150Hz) = \frac{1}{\sqrt{R_1 R_2 C_1 C_2}} = \frac{1}{\sqrt{7500\Omega * 7500\Omega * .2\mu F * .1\mu F}}$$

Using the function generator and oscilloscope it is possible to map the frequency response on a log-log plot. A frequency is set on the function generator and the peak-to-peak voltage of the output waveform is recorded from the resulting waveform. For a frequency range of 5 to 1500 Hz the following results were retrieved:

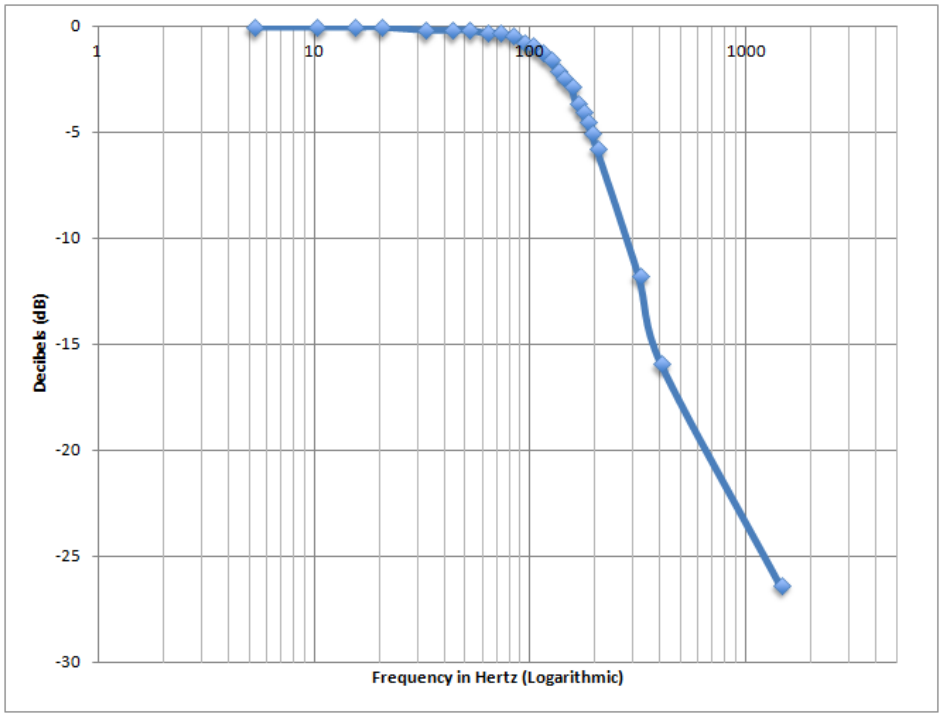

Figure 9: Bode Plot for Two Pole LPF with 150 Hz Cutoff

Extrapolating from the last few datapoints of the plot, we can postulate that the dropoff is indeed -40dB/decade. The very last datapoint is a bit higher than expected, but this can be attributed to the increased impact of noise in the measurements as the voltages measured become smaller and smaller. Furthermore, the Bode plot shows that the frequency response of the Sallen-Key LPF is nearly flat at 0dB, corresponding to a Butterworth filter. The filter starts to appreciably attenuate the input only after 100 Hz. This is important as it shows the filter is not adding a gain to the signal for the frequencies in which we are interested. The flat topology of the curve over the interested frequencies is also important as it suggests that the filter treats the gain of the frequencies relatively the same for these frequencies. Overall, this means that we can pass our input through the filter without worrying about it being appreciably attenuated below the cutoff frequency. If this were not the case, then the amplifier board would cause some inaccuracies in the signals being measured and sampled by the ADCs on the data acquisition board.

5.3 Testing ADC Sampling

The ADCs on the data acquisition board play a very large role in this project since most of the data gathered from sensors will be in an analog format. The ability to recreate sampled signals without aliasing was of particular concern. Aliasing results when a signal is not

sampled quickly enough to capture the "true data" of a signal. When aliasing occurs, the samples often represent a harmonic of the signal rather than the signal itself. To avoid this, engineers sample signals at the Nyquist rate (twice the frequency of the signal to be sampled) so that the Nyquist criterion will be met. Formally, the Nyquist criterion is[2]:

$$f_s > 2data_{bw}$$

To test the analog sampling capabilities of the data acquisition board, a Tektronix CFG250 function generator was connected to the 4th channel of the ADC inputs. A sine wave was generated starting at 100 Hz and decreased slowly by hand to approximately 1 Hz. The signal generated had a peak-to peak voltage of 5V with 0V as the bottom. The data acquisition board was directed to sample the signal at 500 Hz (2ms period). As expected, the DAQ board was able to sample the signal without aliasing:

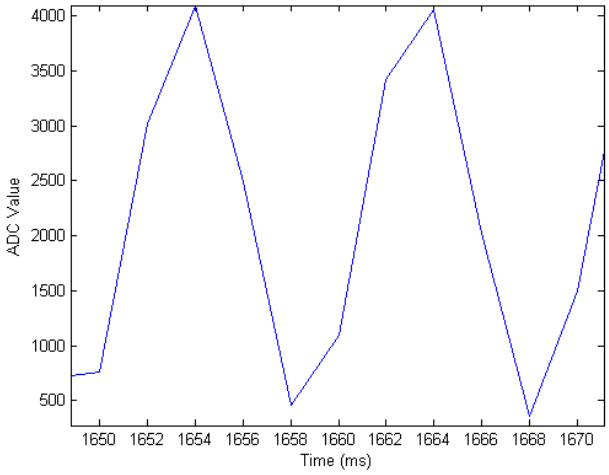

Figure 10: 2ms Sampling of 50 Hz Signal

From this graph, one can make out 10 data points that comprise the data making up the 50 Hz signal. By measuring the time between two peaks (1648 and 1668 ms, for example), one can confirm that this signal matches 50 Hz:

$$\frac{1}{1668ms - 1648ms} = 50Hz$$

There is the problem of data smoothness at this rate. With only 10 data points making up the sine wave, it appears rather "choppy" instead of the smooth function many of us are used to seeing. This choppiness can be improved by either lowering the frequency of the signal being sampled, or by increasing the sampling rate of the DAQ. However, there is a tradeoff between increasing the sampling rate and getting accurate RPM data. The RPM data relies on counting the number of teeth that pass by each sampling period. If one increases the

sampling rate, the period for counting the teeth decreases. With this decreased period, the system loses resolution since the number of teeth passing by each period decreases.

Therefore, the operator must be mindful of the balance that must be struck between an increased sampling rate (which benefits ADC sampling) and a lower sampling rate (which benefits the external interrupts). A 500 Hz sampling rate is well beyond the specification needed for this project as the team is planning on normally operating the board at a 20 Hz rate, which is double what is currently used in production. A more "reasonable" waveform sampled at 500 Hz is the following (still without aliasing):

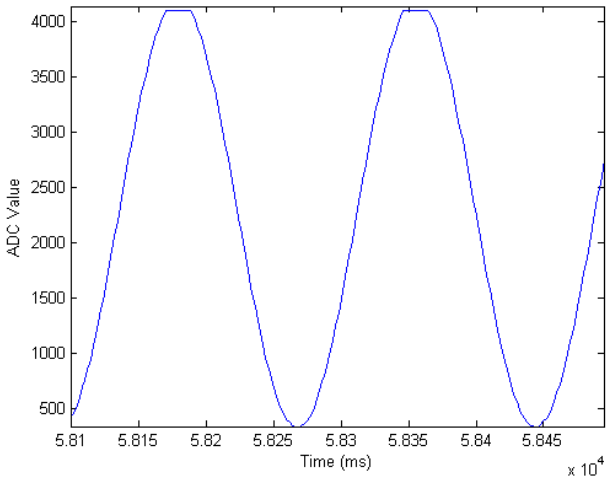

Figure 11: 2ms Sampling Rate of 5.7 Hz Signal

As the system operates well sampling at 500 Hz, I am very confident that it will be able to adequately sample signals at a 20 Hz rate. In this case,

$$\frac{20Hz}{2} = 10Hz$$

At a 20 Hz sampling rate, signals above 10 Hz will experience aliasing and most likely be unrecoverable. This should not be too much of an issue as the signals the team currently measures are relatively low frequency at only a few Hertz. If fast sampling rates such as 500 Hz are desired, then the operator is directed to turn off outputs from the external interrupts and only gather data from a few channels. The more channels that are gathered, the lower the maximum sampling rate achievable by the data acquisition board.

6 Future Goals

There are a few things that could be done with future iterations of this project. Firstly, independent sampling rates could be coded to decouple the accuracies of the RPM and ADC measurements. Many commercially available data acquisition units already have this feature.

Another feature that could be implemented would be automatic control of the throttle and/or braking system on the dynamometer. I have already written code that generates a pulse-width modulated signal to control a servo motor. The code has been tested and proven functional. It was decided to leave the automatic control out of this iteration of the project due to time constraints and the team's need for a functional dynamometer leading up to competition. Implementing the automatic control portion of the project would have occupied the dynamometer for a few weeks.

After finals, I plan on working with Brian Curless to integrate my data acquisition board and amplifier board into the current dynamometer. That way, the increased sampling rate, improved ADCs, and improved filters can be used to better tune the CVT for the competition at Rochester Institute of Technology June 6-9, 2013.

7 Conclusion

In conclusion, the improved electronics for the Baja team's CVT dynamometer were developed through a process of design, building, and testing. In the design phase, feedback was sought from the subteams that have an interest in this project. All groups involved came to a consensus on the requirements for the project. With the requirements at in hand, I set about designing a new data acquisition system, experimenting with various design proposals and encountering mistakes along the way. Once the system was built, I was able to test it and show that the data acquisition system did meet the requirements set. The system can successfully sample 16 ADC channels with 12-bit resolution and has the ability to communicate with 16 GPIO digital ports. Furthermore, the system can count the number of interrupts on each of its external interrupts, which is important for measuring rotation rates. A cascaded amplifier and Sallen-Key low-pass filter were created to both amplify and filter noise from the outputs of analog sensors. These circuits were placed on PCBs for robustness and code was written to take the data from these channels and output it over USB for a computer to process further. Using the data from this system, the team can better tune its CVT for improved performance and continued success at competition.

8 Acknowledgments

I would like to thank Dr. Bruce Land for his advice and wealth of knowledge. His insight helped shape my project into what it is today.

I would also like to thank the Cornell Baja Racing SAE team for allowing me to contribute to the team's continued success through an MEng project. On the team, I would like to extend my gratitude to Andrew Michaels, Brian Toth, and Brian Curless, for their knowledge about the LPC microcontroller and mechanics of dynamometers.

For the use of their LaTeX template for this report, I would like to thank Adam Shapiro and Tom Chatt.

Finally, I would like to thank my family for their support and encouragement as I worked to complete this project.

9 References

[1] Wikipedia: "Sallen-Key Topology" http://en.wikipedia.org/wiki/Sallen-Key_topology
[2] Wikipedia: "Nyquist Rate" http://en.wikipedia.org/wiki/Nyquist_rate

10 Datasheets

LPC1769: http://www.nxp.com/documents/user_manual/UM10360.pdf
FT232RL: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf
AD7927: http://www.analog.com/static/imported-files/data_sheets/AD7927.pdf
AD780: http://www.analog.com/static/imported-files/data_sheets/AD780.pdf
LM340T: http://www.ti.com/lit/ds/symlink/lm340-n.pdf
TL1963A: http://www.ti.com/lit/ds/symlink/tl1963a.pdf
SRDA05-4: http://www.semtech.com/images/datasheet/srda05-4_srda12-4.pdf
REF3225: http://www.ti.com/lit/ds/symlink/ref3225.pdf
INA827: http://www.ti.com/lit/ds/symlink/ina827.pdf
TS321: http://www.ti.com/lit/ds/symlink/ts321.pdf

11 Appendix A - Additional Figures

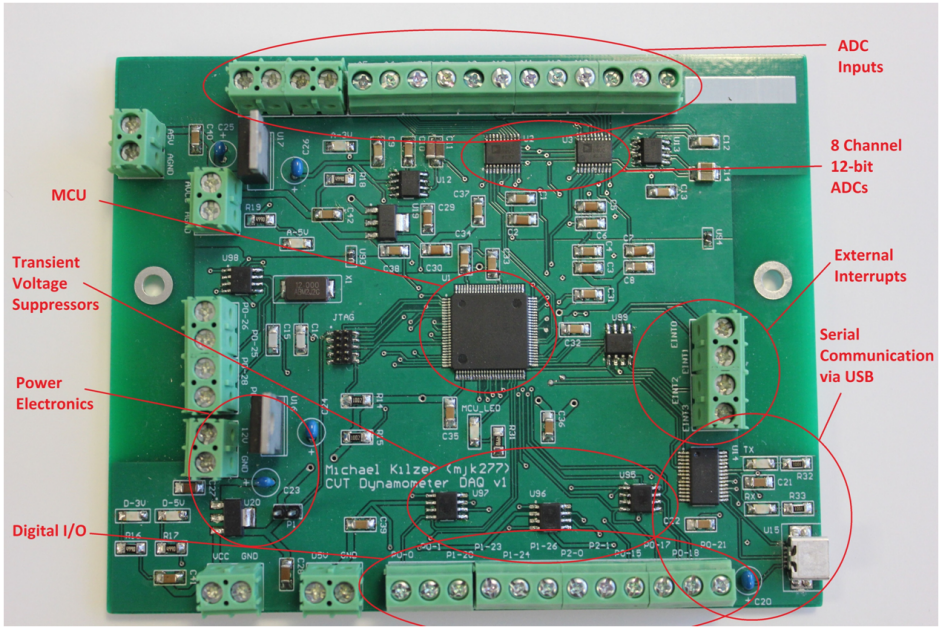

Figure 12: Annotated Data Acquisition PCB

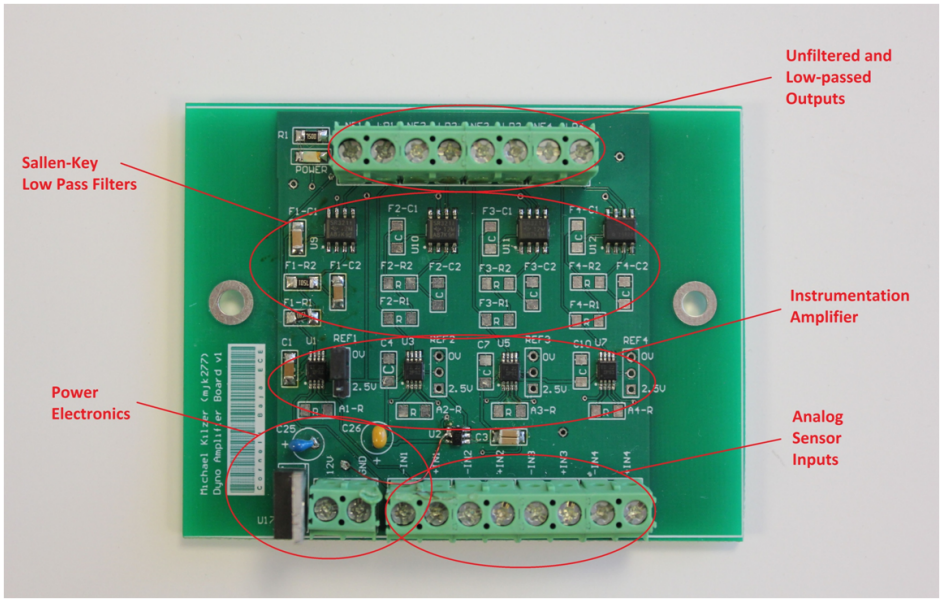

Figure 13: Annotated Amplification Board PCB

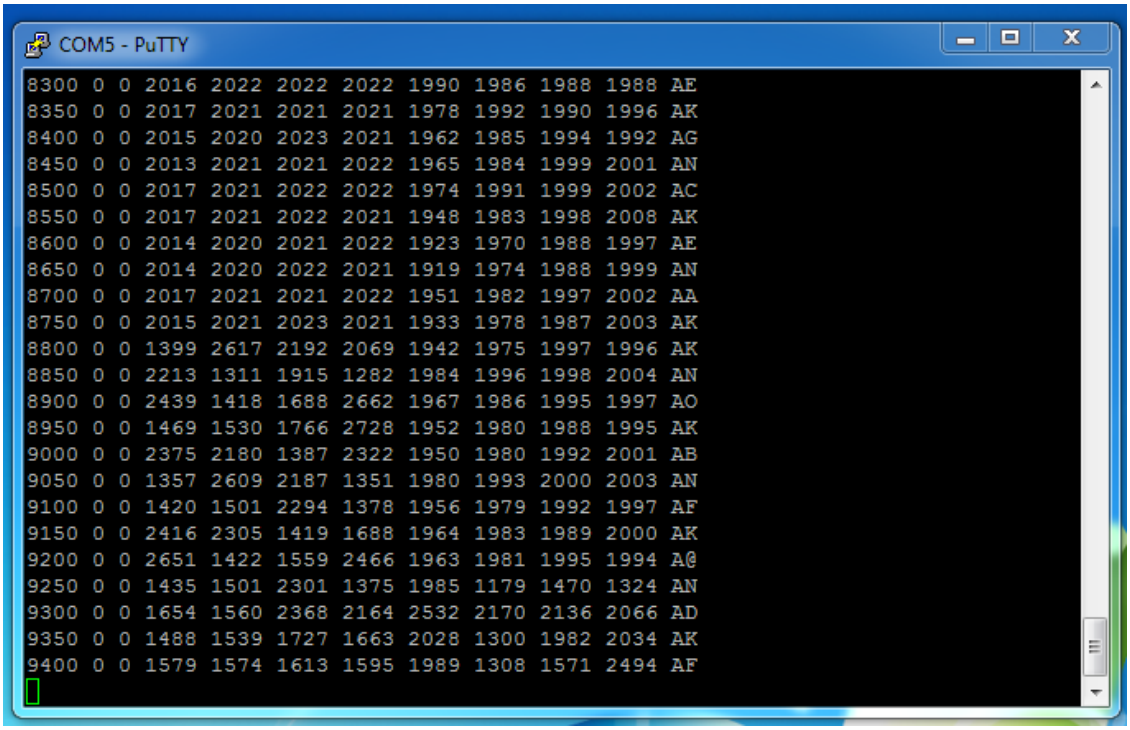

Figure 14: Sample DAQ Output

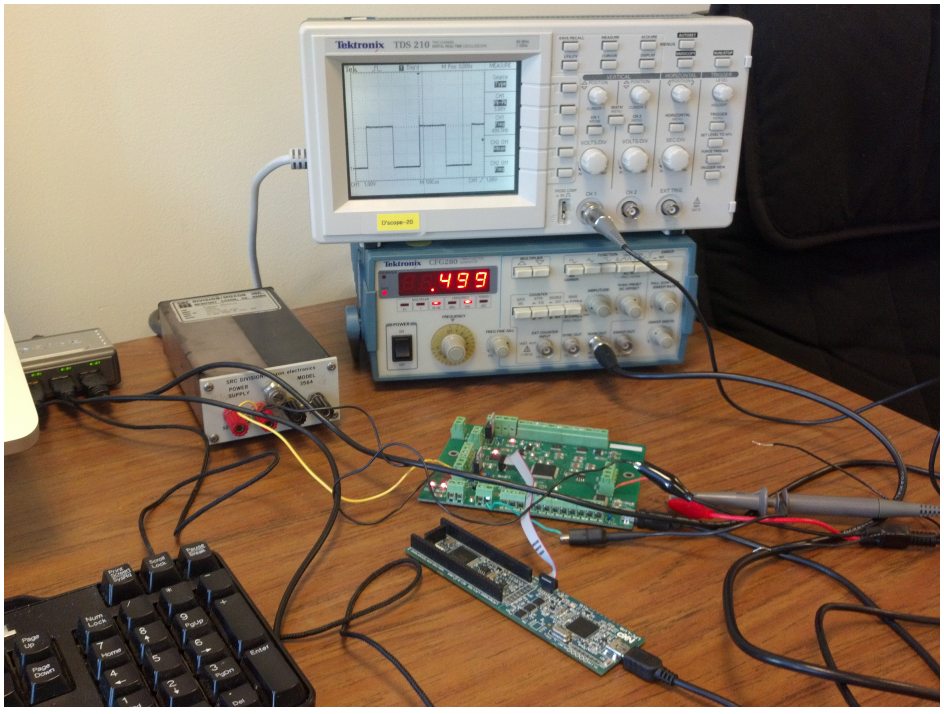

Figure 15: Testing Setup for Data Acquisition Board

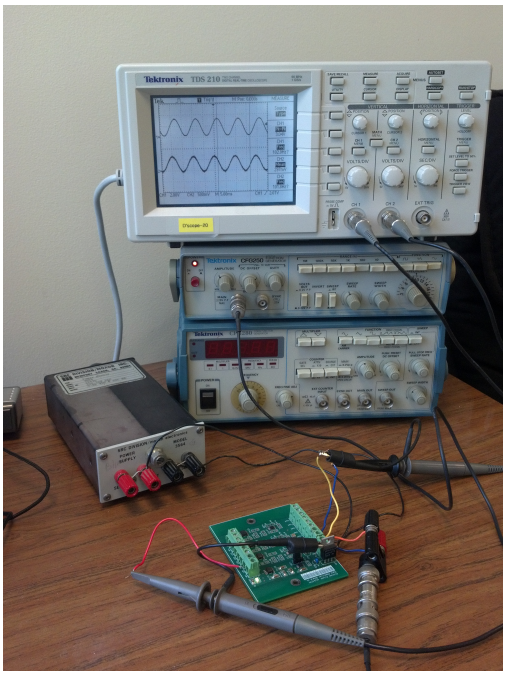

Figure 16: Testing Setup for Amplifier Board

12 Appendix B - User's Guide

12.1 Hardware

To use the data acquisition board, the user needs to supply 12V between the 12V and GND terminal block connectors on the board. The board will step the voltage down to the appropriate voltages it needs. The same situation is applicable for the amplification board.

Connect analog inputs with a maximum voltage of 5V to the ADC channels at the top of the board. Each one is numbered 1-16. 16 GPIO pins are broken out on the board too. These are labeled P(Port#.Pin#), for example P0.27. The pins chosen have many useful functionalities such as PWM, I2C, CAN, and more. Refer to the LPC1769 datasheet for the functionalities associated with each pin. Avoid connecting components that operate at 5V. The pins on the LPC1769 are 5V tolerant, but it will not output 5V for a 5V device. For device safety, transient voltage suppressors have been put in-line with the digital I/O pins. These should protect the ports of the MCU from overvoltage conditions.

The amplification board uses instrument amplifiers, meaning that the input to each of the four channels on the amplification board is a differential. In the case that a differential is not needed, tie the negative input of the channel to ground. Depending upon the input signal, you may need to increase the gain of the amplifier. Without a resistor in the A*-R position,

the amplifier will multiply the signal by 5. Gains of up to 1000 can be reached by using the following equation:

$$R_G = \frac{80000}{G - 5}$$

where R_G is the resistor value to use in Ohms and G is the desired gain. Furthermore, the offset of the output signal from the amplifier can be set by the jumper pins next to the amplifier. One position sets an offset of 0V while the other sets an offset of 2.5V.

The amplifier board outputs both an unfiltered and low-pass filtered output that can be passed into the ADC channels on the data acquisition board. The low-pass filtered output is useful when there is high-frequency noise impacting the integrity of the measurement an in preventing aliasing. One needs to calculate the corresponding resistor and capacitor values to produce the desired low-pass filter. One can use the following transfer function:

$$H(x) = \frac{\omega_0^2}{s^2 + 2 + 2\alpha s + \omega_0^2}$$

with the cutoff frequency described by

$$\omega_0 = \frac{1}{\sqrt{R_1 R_2 C_1 C_2}}$$

and a Q factor of

$$Q = \frac{\sqrt{R_1 R_2 C_1 C_2}}{C_2(R_1 + R_2)}$$

Plugging in $\omega_0 = 2\pi f_0$ and setting $Q = 1/\sqrt{2}$ to get the flattest response for frequencies below the cutoff, the designer can plug in chosen values of the resistors or capacitors to determine the values of the other components.

Ensure that both the amplifier and data acquisition boards use the same grounds to guarantee the integrity of the measurements. If both grounds are not the same, then mismatches in measured values can occur.

12.2 Software

A computer can communicate with the data acquisition board through the use of a terminal emulator such as PuTTY. The serial to USB interface operates at 115200 baud with 8 data bits, no parity bit, and 1 stop bit. Furthermore, flow control is off. The serial port name used on your computer will vary depending upon the operating system.

The data acquisition board will not collect data as soon as it is turned on. It is waiting for a command string from the computer. Command strings are in the form `$5,0x0A,0xAAAA#`

where the part before the first comma denotes the sample period in milliseconds. The 2nd and 3rd values are hexadecimal control codes corresponding to the external interrupts and the ADCs, respectively. A 1 designates that a channel is displayed while a 0 designates that a channel is not displayed. Only the least four bits of the external interrupt control code are used, with EINT3 corresponding to the left-most bit of the hex value. The ADC control code uses all four hex values with the left-most bit corresponding to the 16th ADC channel. Once the '#' character is sent in the above string, the DAQ starts collecting data. Sending a '@' will pause the recording of data and sending an '&' will restart the recording of data with the same configuration. One can also send a new configuration string to the DAQ. It will restart and record the data in the new configuration.

The output from the DAQ first outputs the time in ms, then the teeth/sample counts of any EINT channels selected, the ADC values of any ADC channels selected, and finally the two checksum bytes. Each part of the data is delimited by a space. For example, a made up output would be: `11260 0 2450 1599 1575 1571 1572 @@`. It is up to the operator to keep track of which outputs are external interrupt outputs and which are ADC outputs. This should be easy to do in the processing software on the computer, especially since it specifies the channels it needs and passes them on to the DAQ. Furthermore, the external interrupt values are always output before the ADC values.

13 Appendix C - Schematics and Code

Detailed schematics and code for the project are available on the project's website.