Purdue Vibrational Instrument Payload for Educational Research (PVIPER)

AAE 418
Marten Berlin, Dhruv Jain, Ryan Keith, Osvaldo A. Martin, Brady Walter

Faculty Advisor: S. Collicott, Ph.D

Purdue University, West Lafayette, IN, 47906, United States

Executive Summary

The goal of the AAE418 PVIPER project is to measure vibrations on a payload of a suborbital rocket flight for educational purposes. This experiment consists of a 3-axis accelerometer and data logger mounted to a modified Launchbox 2U sized payload. The Fall 2018 inherited the project with the hardware already selected and attached, and the code in progress. By the end of the semester, the device is able to properly collect and record acceleration, temperature, and time data. Before the device is ready to launch, the code must be integrated with the flight software, a new connector should be placed for seamless flow of data and transferred to a new baseplate.

I. Table of Contents

II. List of Figures .... 4 -

III. Introduction .... 5 -

IV. Objective .... 6 -

V. Background Information .... 6 -

V.A. SPI Communication .... 7 -

V.B. Vibration Measurement .... 7 -

VI. Experiment Details .... 8 -

VII. Payload and Electronics .... 9 -

VII.A. Why choose the ADIS16228? .... 11 -

VII.B. Why choose the Feather M0 Data Logger? .... 11 -

VII.C. Electrical Connections .... 11 -

VII.D. System Layout .... 13 -

VII.E. Aluminum Endcap Specifications .... 13 -

VII.F. Why Aluminum? .... 13 -

VIII. Coding Methodology .... 15 -

VIII.A. Arduino Fundamentals .... 15 -

VIII.B. Recording to SD Card .... 15 -

VIII.C. ADIS16228 to Feather M0 Data Logger SPI Communication .... 15 -

VIII.D. ADIS16228 Accelerometer Parameters .... 16 -

VIII.E. SPI Settings .... 16 -

VIII.F. Byte Addressing .... 16 -

VIII.G. Registers Used .... 17 -

VIII.H. Recording Mode .... 17 -

VIII.I. Sample Rate .... 17 -

VIII.J. Resonance Correction .... 17 -

VIII.K. Reading Data Output .... 18 -

VIII.L. Global Commands and System Checks .... 18 -

VIII.M. Spacecraft to Feather M0 Data Logger communication .... 18 -

IX. Code Walkthrough .... 19 -

IX.A. Opening Comments .... 19 -

IX.B. Pin Layout .... 19 -

IX.C. Library Inclusion and SPI Setting Setup .... 20 -

IX.D. Variable Declarations .... 20 -

IX.E. Setup() and General Settings .... 20 -

IX.F. SD Card and Data Log File Initialization .... 22 -

IX.G. Check accelerometer connection and Self-Test Accelerometer .... 23 -

IX.H. Setup Accelerometer .... 24 -

IX.I. Retrieve flight information from Nanorack........ 24 -

IX.J. Configuration, Read, and Calibrate Timestamp and Temperature .... 25 -

IX.K. Record, Calibrate, and Close........ 25 -

IX.L. User Defined Functions - writeCommand, readCommand, readFFT, dataread, configSPI ... 26 -

IX.M. Parse_serial_packet User Defined Function........ 26 -

X. Conclusion........ 27 -

XI. Recommended Next Steps........ 28 -

XII. Helpful Notes........ 28 -

XIII. Acknowledgements........ 28 -

II. List of Figures

FIGURE 1: ACCELEROMETER WIRED TO DATA LOGGER .... 9 -
FIGURE 2: FEATHER M0 DATA LOGGER .... 10 -
FIGURE 3: ADIS16228 ACCELEROMETER .... 10 -
FIGURE 4: ADAFRUIT FEATHER M0 CONNECTIONS TO ADIS16ACL1 BREAKOUT BOARD ...... 12 -
FIGURE 5: ADIS15ACL1 BREAKOUT BOARD PINOUT .... 12 -
FIGURE 6: SYSTEM LAYOUT.. 13 -
FIGURE 7: FLOW CHART OF ADIS PROCESS .... 15 -
FIGURE 8: PIN LAYOUT COMMENT SECTION .... 19 -
FIGURE 9: LIBRARY INCLUSION AND SPI SETTING SETUP .... 20 -
FIGURE 10: KEY VARIABLES THAT WERE CHANGED/ADDED IN FALL 2018 CODE.................. 20 -
FIGURE 11: SPI INITIALIZATION AND PIN ASSIGNMENTS... 21 -
FIGURE 12: CREATION OF .TXT FILE FOR DATA STORAGE .... 22 -
FIGURE 13: CHECK ACCELEROMETER CONNECTION AND SELF-TEST ACCELEROMETER......... 23 -
FIGURE 14: ACCELEROMETER SETUP .... 24 -
FIGURE 15: INFORMATION RETRIEVAL FROM NANORACK .... 24 -
FIGURE 16: TIMESTAMP AND TEMPERATURE READ AND WRITE .... 25 -
FIGURE 17: DATA CALIBRATION AND WRITING .... 25 -

III. Introduction

Monitoring vibration is a critical component of ensuring structural stability of components and accuracy of experiments during the flight. The need for understanding the extremes of vibrations during flight and the potential effects on payloads are critical to ensuring the success of rocket flights. The maximum vibration seen by a rocket typically occurs during the suborbital flight phase, so for this experiment it is crucial to use equipment that is robust enough to survive the extremes of launch, yet sensitive enough to detect the less extreme vibration environment of the upper atmosphere and space.

Based upon the need to measure and record vibration during flight, the PVIPER system has been developed to achieve this by reliably monitoring vibrations during suborbital flights inexpensively. The goal of the experiment is to develop a better understanding of vibrations during suborbital flights, including a full analysis of the vibration spectrum throughout the critical moments of the flight. Providing this information to industry engineers could help improve the design of payload and other supporting devices to minimize adverse effects and prevent failures due to vibrations.

IV. Objective

The main objective of the experiment is to monitor the vibration spectrum of the launch vehicle. This will be done by measuring the acceleration of the payload in the x, y and z directions. The secondary objective is to include the corresponding time and temperature data. This data will be saved to an SD card in such a way that in can later be easily opened in several programs in order to perform an FFT analysis on it. The nature and design of the selected components should allow for the addition of other components and objectives to the experiment such as the monitoring of the surrounding environmental conditions such as humidity or pressure.

V. Background Information

This report first provides an overview of the results of the team's research in preparing this experiment for flight and selecting the components needed to perform it. A number of subjects are covered to provide future teams with an overview of the various factors that must be considered for such an experiment as well as provide understanding why certain components and protocols were selected.

These include:

1. An overview of SPI communication protocols and programming requirements.

2. An explanation of vibrational recording and analysis, including limiting factors and design criteria rules.

3. An introductory guide to making use of Arduino systems in the experiment, with a guide on the current status of the code and its functions.

Furthermore, this report explains the overall requirements and details of the experiment, including manufactured parts and electrical connections, as well as reasonings behind the choices made by previous teams during manufacturing.

Following that is a walkthrough and explanation of the most up to date code. This will include explanations of what each function does and how it operates. Note that several address registers will be referenced in this section. A full list of the register names and what value they correspond to can be found in the ADIS 16228 user guide.

Lastly, the report concludes with a general overview of the experiment as a whole, with member thoughts on the best next steps for future teams, as well as a long-term vision for the experiment.

V.A. SPI Communication

SPI, or Serial Peripheral Interface, is the protocol used to control the electronics used on the payload. SPI is used for communicating between microcontrollers or peripheral devices quickly over short distances. Devices on an SPI loop are either the master or a slave, with the master being the device that controls the whole loop. SPI is broken into three lines: MISO (Master in Slave out), MOSI (Master out Slave in), and SCK (Serial Clock). MISO are the lines used to send data from the slaves to the master typically sensors sending data or microcontrollers reading values. MOSI are the lines used for the master to send data to the slaves, and thus can control the actions of the attached slave devices. SCK is the serial clock, which sends synchronized pulses from the master to keep all devices at the same point in time.

Communication works by each slave device having a SS (Slave Select) pin. The master can set this to either high or low; on low, the slave listens to the MOSI line, and on high, it ignores them. This allows the master to control which slave devices receive each set of commands, and thus allows the master to give different sets of instructions to different attached devices on the same MOSI line, one of the main benefits of SPI communication. The slave select pin is also used by the master to identify which device sent data through a MISO line.

V.B. Vibration Measurement

The way vibration will be measured in this experiment involves the frequency of the vibrations and the strength of them. The frequency tends to be measured in Hertz (Hz), or number of times a complete oscillation is made per second. When the accelerometer records data, it will measure in samples per second. To get a clean set of data, the sample rate of the accelerometer should be significantly larger than the vibration frequency encountered. From information provided by the flight provider, the maximum frequency of vibrations encountered can be expected to be 2000 Hz. Due to the Nyquist frequency, the sample rate of the accelerometer must be above 4,000 Hz. In order to add a margin of error, a common value is 2.5*frequency, meaning a sample rate of at least 5,000 Hz is recommended.

The strength of vibrations is measured in force units of *g*, or mass*acceleration of gravity, where the force of gravity is the gravitational constant at sea level for Earth and mass is assumed to be one, as it will cancel out in the vibrations experienced on the payload. From information provided by the flight provider, it is expected to encounter vibrations potentially up to 23 *g*, though standard vibrations will probably not exceed 3.6 *g*. It should be noted that another way strength is expressed is in ASD, with units g^2/Hz. This is a method of relating strength for different frequencies, if the data is available, and takes the form ($strength^2/frequency$) as expected by the units.

VI. Experiment Details

The given requirements for this project was laid out at the start of the Fall 2017 semester assuming the use of the NanoRacks Feather-Frame payload device. The information on the system provided the base requirements of operations, including fitting in the available limits of provided power. The Feather-Frame provides 0.9 Amps at 5 Volts through either a standard USB 3.0 or USB 2.0 Type B female interface (NFF Payload User's Guide 2016, p. 3). Additionally, NanoRacks provides a series of ASCII codes which can be used for signaling the payload at various points during the flight, including launch, descent, and landing times for automated triggering of components.

To provide a baseline of operation for a specific space vehicle, the launch and vibration data of a Blue Origin Shepard vehicle was used to create a basic flight profile and regime guideline. This translated to a total experiment time of around 2.5 minutes, experiencing around 3 G’s at launch (Blue Origin New Shepard Payload User's Guide 2017, p. 37). Additionally, the Payload User's Guide provided Table 1 with suggested spectrums from vibrational testing. This would provide the basis of our selection criteria for accelerometers.

Additional requirements included a system capable of collecting and storing all gathered data during the experiment, necessitating the need for data logging and sufficient memory storage. The entirety of the system is also required to fit within a standard 2U CubeSat payload bay as provided by the Feather-Frame rack, translating to an area of 10 cm x 10 cm x 22 cm and weighing no more than 0.5 kg. However, it was strongly preferred that the system should be capable of being reconfigured for a 1U CubeSat payload size (10 cm x 10 cm x 11 cm) to facilitate use in other payload platforms and launch vehicles. The system was built using the provided Launchbox standard design and can be modified to fit within a 1U Cubesat payload size.

VII. Payload and Electronics

Below is the list of components used for the build:

- Adafruit Feather M0 Data logger
- SanDisk Ultra 16GB microSDHC UHS-I card with Adapter - 98MB/s U1 A1
- USB cable - 3.0A/2.0 MicroUSB
- 24-Gauge Solid Core Wire
- ADIS16228CMLZ Digital Triaxial Vibration Sensor with FFT Analysis and Storage
- ADIS16ACL1/PCB Breakout Board

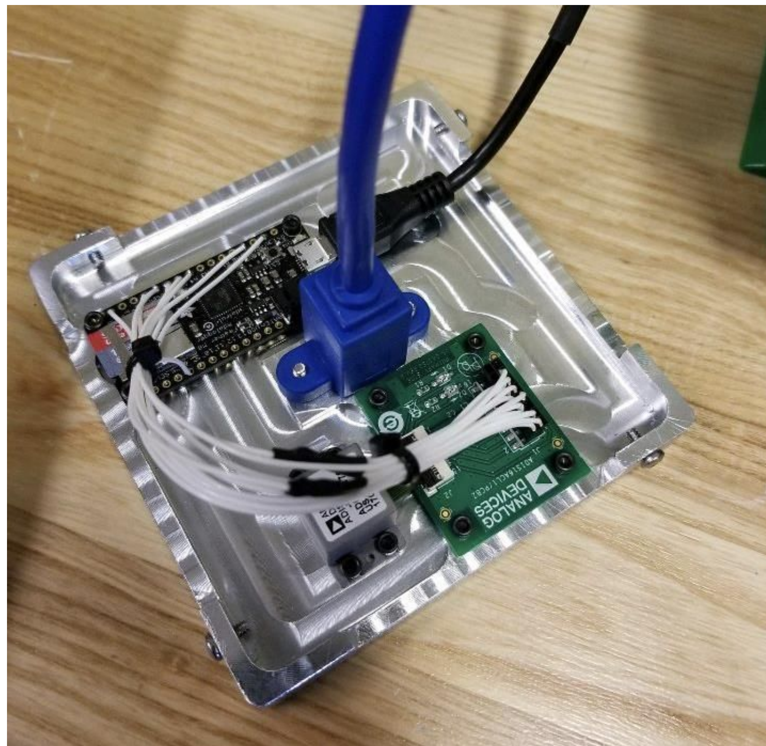

Figure 1: Accelerometer Wired to Data Logger

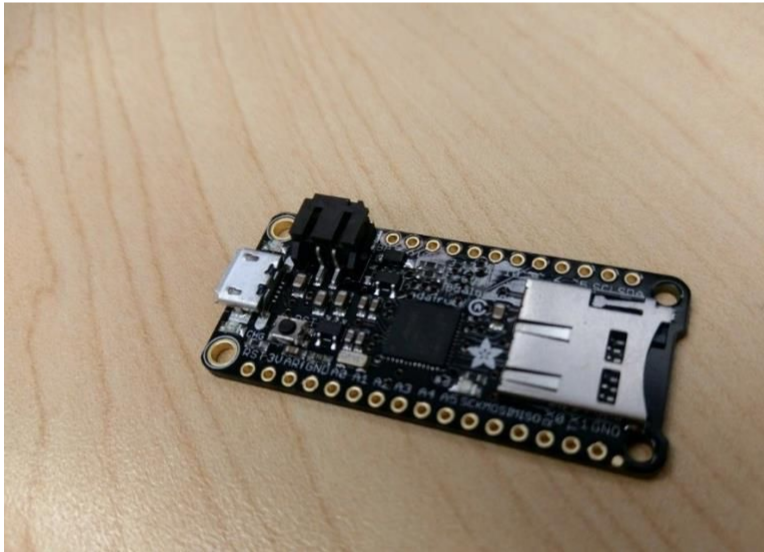

Figure 2: Feather M0 Data Logger

Figure 3: ADIS16228 Accelerometer

VII.A. Why choose the ADIS16228?

The ADIS16228 was chosen for a number of reasons, the main reason being the magnitude of error it had. For the highest available frequency of 20,480 samples per second (SPS), which is likely larger than needed, the peak noise per bin was 5.18 mg (milli-G's), which is very small compared to what would be encountered. For the anticipated frequency to be used of 5,120 SPS, the peak noise per bin would be 2.59 mg. Given that the expected range of normal vibrations is from -3.6 to 3.6 g, a 2.59 mg error represents less than 0.1 percent error on the vibration, a more than acceptable tolerance.

Another reason of choosing the ADIS16228 is due to the speed of processing. As already stated, it can record samples at up to 20,480 SPS, well above what is needed to capture a maximum of 2,000 Hz vibrations. Additionally, the ADIS16228 is an MEMS sensor (Micro Electrical Mechanical System), which have shown to be more accurate at higher vibration levels. Tests of this can be seen in the document "The suitability of low-cost measurement systems for rolling element bearing vibration monitoring", by Jarno Junnola. The author also does a market review of accelerometers, in which the ADXL001, also from Analog Devices, which scores very well for low cost sensors. The ADIS16228 is similar to the AXDL001 but has more accurate sensing at larger frequencies than the ADXL001, accompanied by the larger price tag.

VII.B. Why choose the Feather M0 Data Logger?

The Feather M0 Data logger was chosen for its size, ease of use, and cost. The Feather M0 has an SPI speed of 24 MHz, significantly above the maximum rate of the ADIS16228 (2.25Mhz). The integrated SD card makes for a small and integrated package, meaning no extra hardware to deal with as compared to other microprocessor and logging options. The Feather M0 also runs on the Arduino IDE and Arduino libraries, meaning there are numerous resources available to assist with development. Information on Adafruit Feather M0 and connecting to it using Arduino IDE is readily available on the Adafruit website.

VII.C. Electrical Connections

The system has been connected based on the wiring diagram shown in Figure 4. It is important to note that the pin numbers and their corresponding functions listed for the ADIS16228 are <u>NOT</u> the same as the pin numbers and functions on the ADIS16ACL1 breakout board. If connection issues arise, refer to the breakout boards pinout diagram, rather than the accelerometer itself. Worth mentioning is the reset pin, which is set "low" by default and therefore active. So, to not reset the board all the time, it has to set "high" in the beginning of the code.

The connector used on the ADIS16ACL1 breakout board uses 2mm spaced pins. This is smaller than the standard 2.54mm spacing and neither 2.54mm connections or individual 2mm

connections will fit on the board. Therefore, a 2x16 2mm spaced connector must be used to connect to all of the needed pins. Until Fall 2018 there was a crimped connector with stiff wires. This led to some problems, because a few wires lost connection when getting bent a little bit. To prevent this in the future, it was decided to get a new connector with more flexible wires and an improved stability at the connection point between connector and wires. An IDC Ribbon Cable complies these conditions. It was ordered at the end of the semester and has to be installed.

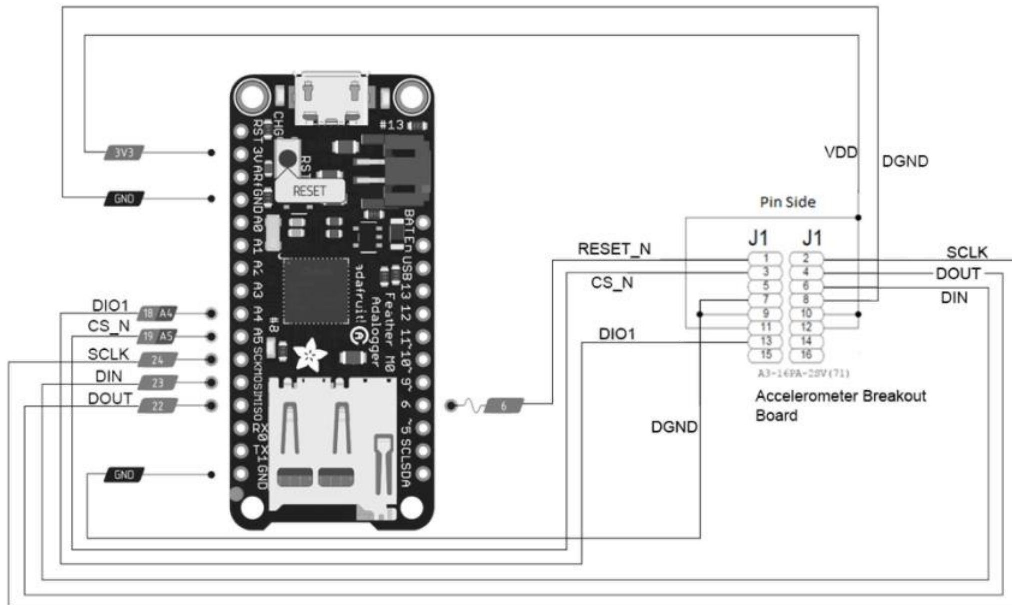

Figure 4: Adafruit Feather M0 connections to ADIS16ACL1 Breakout Board

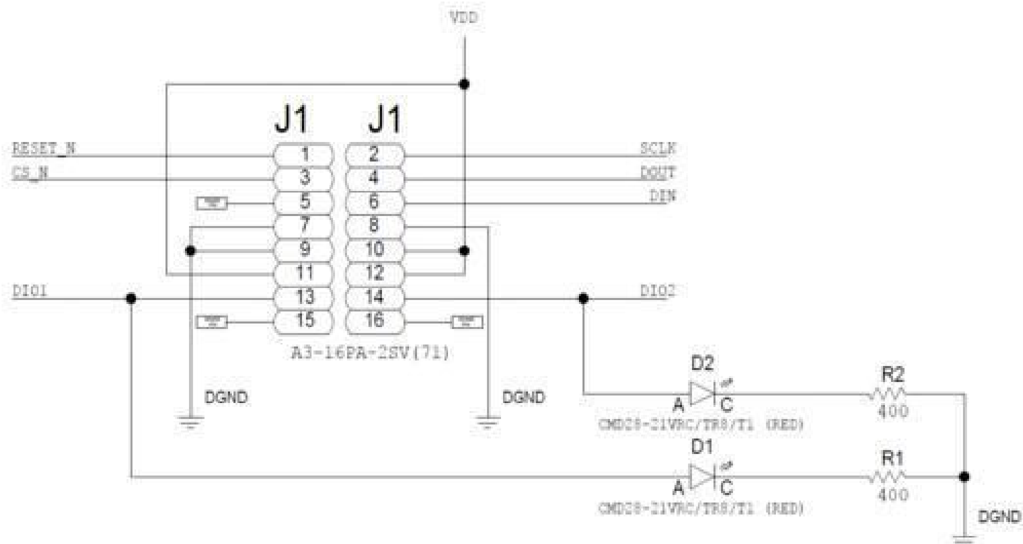

Figure 5: ADIS15ACL1 Breakout Board Pinout

VII.D. System Layout

The overall system and its components are connected as shown in Figure 6.

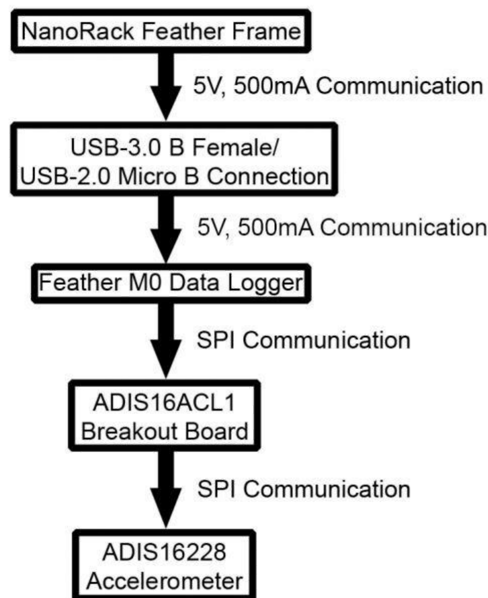

Figure 6: System Layout

VII.E. Aluminum Endcap Specifications

The aluminum endcap that the experiment is mounted to is designed specifically for this experiment and therefore has some variations from the standard endcap designed by the Launchbox team. Standoffs are built into the endcap for the data logger and breakout board. No standoff was designed for the accelerometer to reduce any potential error in vibration measurements. All electronic components are mounted using 3-56 x 3/16" screws. 3-56 was chosen due to the small mounting points on the accelerometer which would not accommodate 4-40 screws, therefore to be consistent all electronics were mounted using 3-56. 3-56 threads were chosen over the more common 3-48 due to the relatively short hole length allowed by the thin endcap. Normally, the endcap is mounted to the sheet metal side panels using self-tapping wood screws. In the aluminum endcap, the endcap mounting holes are tapped for 4-40 screws. Mounting holes for the USB-3.0 B are through-holes, with a diameter of 0.120in, as the USB connector itself contains threaded mounting points.

VII.F. Why Aluminum?

The endcap that is used for mounting the accelerometer and boards is made out of aluminum. This design choice is because the endcaps received from the 3D printers in

Armstrong Hall had poor quality. The aluminum has added benefits of being sturdier while also
being lightweight. There will be less cause for worry from the endcap failing if it were to vibrate or suffer a decent fall; however, if a different endcap is required for future endeavors, there are 3D CAD files located in the PVIPER folder. The Launchbox team was able to successfully 3D-print their endcaps and have a special process for working with the printers. This information can be obtained from their project report from the Spring 2018 semester.

VIII. Coding Methodology

There are three main challenges in coding the system: the use of the SPI library, understanding the required code and addressing of the ADIS16228, and understanding the output received from the NanoRacks Feather Frame. The following sections will explore each facet of the code, along with the basics of Arduino coding.

VIII.A. Arduino Fundamentals

In order to write code to the Feather M0 and ADIS16228, the Arduino IDE is used. The IDE requires specific plugins for the Feather M0, which can be found on the Adafruit website. Because the ADIS16228 is a simple SPI slave, no special plugins are needed.

Regarding the coding itself, the Arduino language is based on C++, despite the IDE being written in Java. The language features enhanced capabilities and many more complex features than standalone C++. However, because the languages are so similar, an external source code editor can still be used to write the code. But it is best to use Arduino IDE because of its user friendly interface.

VIII.B. Recording to SD Card

Writing to an SD card is rather straightforward in the Arduino language. After including the SD.h library, a single command to open or create a file is used, and to write to the card either write or print commands are used.

VIII.C. ADIS16228 to Feather M0 Data Logger SPI Communication

The communication between the accelerometer and the Feather M0 data logger is achieved through an SPI interface. Since the ADIS16228 contains the analog to digital conversion (ADC), the data output is digital (and thus easier to deal with). All of the ADIS16228 protocol information is located in the user manual and the main findings are discussed below. The general overview of the communication process can be seen in Figure 7 below.

As discussed previously, the use of the SPI interface requires the use of a master and a slave. In this configuration, the ADIS16228 is the slave. In order to send commands to the accelerometer, slave select must be set to low power to allow communication, then back to high power to process. Additionally, individual settings must be called before transfer can begin. For this system, settings include the processing speed, bit sending, and SPI mode. These are explained further in the report.

Figure 7: Flowchart of the SPI process

VIII.D. ADIS16228 Accelerometer Parameters

The greatest challenge when coding for this system is understanding the specific addresses, commands, and standards of the ADIS16228, as seen in the user manual. The Spring 2018 team has highlighted and annotated the ADIS16228 user manual to help future teams understand the system quicker (this can be found in the PDF folder in the PVIPER directory). Please note that as of Fall 2018, several of the written annotations concerning which time capture mode is used and many other aspects of the code are no longer relevant. However, the highlighted sections are still very relevant as these sections are mainly what the current code is based on. The following sections are presented in the same order they are presented in the manual.

VIII.E. SPI Settings

In order to communicate with the ADIS16228, SPI settings must be set. As mentioned previously, the settings for this system include the processing speed, bit sending, and SPI mode. These were set according to Table 07 in the user manual, with a bit rate of 2.25 MHz, MSB First bit sequence, and SPI Mode 3 communication. The bit rate is self-explanatory, however the MSB and SPI mode are harder to understand if there is a lack of experience. In short, MSB refers to Most Significant Bit, which is the left-most bit in an 8-bit sequence, or byte. Sending MSB first means this bit is sent first to begin communication. This does not affect the code syntax however. SPI Mode is based on the phase and polarity of the component clock, which is unique to each slave device. For the ADIS1628, Mode 3 is required.

VIII.F. Byte Addressing

To transmit data to the ADIS16228, a special technique called byte addressing is required. A byte address is a pointer that refers to the lower bite of a setting on the ADIS16228, such as REC CTRL1. Natively, the ADIS16228 natively interprets this address to retrieve the lower byte of the respective setting in order to read or write to it, where this project need to retrieve both upper and lower bytes from the address. The specific methodology 2018 Fall used is explained in Code Walkthrough. In general, in order to begin this transfer, the address must be sent first, followed by the value to assign or read. Because these values are binary, either binary or hex values must be used for communication. For a pictorial representation, please see Figure 09 in the user manual.

VIII.G. Registers Used

Due to the requirements of the system for this mission, only few of the ADIS16228's settings will be modified for simplicity. The registers that will be used for this are BUF_PNTR, X_BUF, Y_BUF, Z_BUF, REC_CTRL1, REC_TRL2, AVG_CNT, DIAG_STAT, GLOB_CMD, CAL_ENABLE, TIME_STMP_L, TEMP_OUT and DIO_CTRL. These will be individually explained in the following sections and can also be found on pages 10 and 11 of the ADIS 16228 user guide.

VIII.H. Recording Mode

The recording mode for the ADIS16228 is controlled by the register REC_CTRL1. There are four modes of recording available: manual FFT, automatic FFT, manual time capture, and real-time mode. For our mission, we desire the most sample points and granular data, so manual time capture was selected. This simply records the acceleration in each axis and does not spend system time performing FFT analysis. If unfamiliar, the Fast Fourier Transform, or FFT, converts a signal from either analog or electrical to both magnitude and frequency. This transform requires the recording to be interrupted, which is not preferable. In manual time capture mode, the output can be imported to a separate software such as MATLAB to manually perform FFT after the mission is over. To change the mode to manual time capture, REC_CTRL1[1:0] needs to be set to binary 10 or hex 1A. This means byte 1 has a value of 1 and byte 0 has a value of 0. In other words, the byte is read and assigned left-to-right, hence MSB first.

VIII.I. Sample Rate

When recording a signal, a continuous stream cannot be measured due to the nature of hardware. Instead, the hardware must take a discrete sample at certain intervals to put into a data array. Because a signal is periodic, a sample rate that is too low, will cause the corresponding data points to be non-periodic, inducing jumps in the recorded data. This effect is called aliasing. To prevent aliasing, a sampling rate must be higher than the the Nyquist Frequency, which is usually around double the frequency of the signal. As such, because the vibrations experienced during the mission don't exceed 2000 Hz, a sampling rate greater than 4000 Hz is needed. This rate corresponds to SR2, which has a sampling rate of 5120 Hz. Note that SPS, or samples per second, is the same as Hz. To set the sample rate to SR2, set REC CTRL1[11:8] to hex 02.

VIII.J. Resonance Correction

When a device is in resonance, it is vibrating at its natural frequency, which influences the frequency read by the accelerometer. As described in the user manual, the ADIS16228 has a resonant frequency of 5.5 kHz. Even though the max vibrations seen in the mission are 2.0 kHz, this resonance error is seen as low as 1.0 kHz, to counteract this error, resonance correction must be enabled. This function is default enabled by accelerometer.

VIII.K. Reading Data Output

Because of the limited memory of the ADIS16228, recorded data is buffered. In other words, the unit records each axis, loads into temporary memory until it is full, sends the filled memory to the master, erases the memory, and begins the process again. Appropriately titled, the buffers for each axis are X BUF, Y BUF, Z BUF and TEMP OUT. BUF PNTR starts at 0 and increments each time the buffer is written to full capacity. All of these registers are read from the ADIS16228, so no writing is necessary. The data from each buffer is output in two's complement format for the manual time capture mode.

VIII.L. Global Commands and System Checks

To command the ADIS16228, such as manually start/stop recording, self-test, etc., the register GLOB CMD is used. To use it, send a binary 1 to the corresponding bit as seen in Table 64 in the user manual. For the system, only bit 2, bit 3, and bit 11 are used, which are self-test, restoring axis calibrations, and start/stop recording. If any errors are present, they are written to the register DIAG STAT, which flags a 1 at the corresponding bit to indicate failures. For the system, SPI communication checks and self-testing are performed to ensure the accelerometer is functioning correctly. These errors are reflected in DIAG STAT[3] and DIAG STAT[5], respectively.

VIII.M. Spacecraft to Feather M0 Data Logger communication

NanoRacks Feather Frame provides a communication protocol between the spacecraft and data logger. Since we will record all the data once the power of the rocket is turned on, no information from the spacecraft is needed for this particular mission. However, the Fall 2018 team has yet to finish the function to retrieve data from spacecraft for future usage. If the future team is interested in this topic, the communication protocol could be found at NanoRacks Feather Frame Payload Users Guide. The document is located at C1 – Useful PDFs.

IX. Code Walkthrough

Fall 2018 team repaired the 2018 summer code since the program would output junk values for acceleration and temperature. A complete overhaul was done to the read and write processes that the summer 2018 code had, and many other changes were done and will be presented in this walkthrough. The addition of time stamps for the acceleration is present in the current code. After compiling all of the changes done in Fall 2018 the code is able to setup SPI communication, create file, read and write value into accelerometer, read acceleration, temperature and time stamp values and save the data into a SD card. However, there are few things still need to be tested and changed, which will be mentioned later in section XI, conclusion. Since the code is heavily commented, minimal writing will be presented in this report.

IX.A. Opening Comments

The opening comments detail the function and usage of the code. It explains the overall function of the code, accepted inputs, outputs, methods of data storage, potential errors and error reporting as well as general help and editor tips. Included in the header is team info and the most recent Blue Origin Payload User's Guide used for reference. Due to the length of this section, an image is excluded.

IX.B. Pin Layout

The next section of code is a simple diagram explaining the pin connections between the Feather M0 and the ADIS16228.

```
////////////////////////////////////////////////////////////////////////////
//                         S Y S T E M   L A Y O U T                      //
////////////////////////////////////////////////////////////////////////////
/*
  ACCEL PIN & DESC.       LINK     LOGGER PIN & DESC.
  --------------------    -----    ------------------------
  10: Reset               >>>>>    6: Reset (#9 on board)
  11: SPI Data Input      >>>>>    23: MOSI
  12: SPI Data Output     >>>>>    22: MISO
  13: SPI Serial Clock    >>>>>    24: Serial Clock
  14: SPI Chip Select     >>>>>    19: Chip Select (A5 on Board)
  15: Digital I/O #1      >>>>>    Digital IO #1 (Unused)
  3,4: Ground             >>>>>    Ground
  1,2: Power Supply       >>>>>    3.3v Power
  5,8: Ground             >>>>>    Ground (Unconnected)
  6,9: N/A                >>>>>    Unconnected
  7: Digial I/O #2        >>>>>    Digital IO #2 (Unused)
*/
```

Figure 8: Pin Layout Comment Section

IX.C. Library Inclusion and SPI Setting Setup

This section includes the libraries SPI.h and SD.h, and sets the SPI settings.

```
// CODE SETUP ////////////////////////////////////////////////////////////////

  // Library Declaration - - - - - - - - - - - - - - - - - - - - - - - - -
  #include <SPI.h>                    // Allows communication with SPI Devices
  #include <SD.h>                     // Allows usage of SD Cards


  // Ready SPI Settings (See ADIS16228 Table 7) - - - - - - - - - - - - -
  SPISettings settingsADIS(2250000, MSBFIRST, SPI_MODE3);
```

Figure 9: Library Inclusion and SPI Setting Setup

IX.D. Variable Declarations

Globally declare following variables: Nanoracks definition, Data logger pin value, Accelerometer register addresses, temporary variables, Nanoracks flight data structure. Due to the length of this section, an image is excluded. In Fall 2018 there were changes done to some variable declaration specifically the X, Y and Z buffer variable. The change was that the buffers are now an array of 512 values when run in manual capture mode. Then one value in the form of 16 bit variable is stored for temperature and time stamp for every 512 values of acceleration.

```
short int X_BUFF[512];          // Saves buffer value, acceleration, in X direction,
short int Y_BUFF[512];          // Saves buffer value, acceleration, in X direction,
short int Z_BUFF[512];          // Saves buffer value, acceleration, in X direction,
uint16_t tempCheck;             // Saves 16 bit of Temp_out and stores one value for
uint16_t timestamp;             // Saves 16 bit of Time_Stamp_L and stores one value
float temp;                     // Stores the temperature value after the conversion
```

Figure 10: Key Variables that were changed/added in the Fall 2018 code

IX.E. Setup() and General Settings

The setup function begins, which only runs once. The first portion of setup() begins both serial (USB) connection and SPI connection. The 115200 refers to the baud speed needed for serial communication, as set by the Feather Frame manual. Output pins are also defined in the ADIS 16228 Manual. In Fall 2018, the Reset Pin was set as an output and the DIO_1 register as input. One reason why the summer 2018 code wasn’t working was because the reset pin is in active low by default. This means that the accelerometer was resetting at the very beginning and wouldn't read any acceleration data or save any values. The reset pin is

now set to high, allowing the accelerometer to run and record data. DIO_1 is used to drive the interrupt line and set the data line busy when needed.

```
void setup(){

  // General ------------------------------------------------------------

  // Open Comm Channels - - - - - - - - - - - - - - - - - - - - - - -
  Serial.begin(115200);           // Opens Serial Communications
  SPI.begin();                    // Opens SPI Communications

  // Configure SPI Pin - - - - - - - - - - - - - - - - - - - - - - - -
  pinMode(spiSS, OUTPUT);         // Sets Slave Select Pin as Output
  pinMode(mosi, OUTPUT);          // Sets Master Out Slave In Pin as Output
  pinMode(miso, INPUT);           // Sets Master In Slave Out Pin as Input
  pinMode(sclk, OUTPUT);          // Sets Serial Clock Pin as Output
  pinMode(reset, OUTPUT);         // Sets Reset Pin as Output
  pinMode(dio1, INPUT);           // Sets DIO_1 register as Input

  digitalWrite(spiSS,HIGH);
  digitalWrite(reset,HIGH);       //Reset pin is on Active low so need to set to high

  // Initialize LEDs for Use - - - - - - - - - - - - - - - - - - - - -
  pinMode(ledG, OUTPUT);          // Initialize Green LED as output
  pinMode(ledR, OUTPUT);          // Initialize Red LED as output
```

Figure 11: SPI Initialization and pin assignments

IX.F. SD Card and Data Log File Initialization

Every time the code runs, the program will automatically create a new .txt document in numerical sequence, starting with 0 counting upwards 1, 2, etc. After creating the file, the program will check whether the document already exists and if not open the file in written mode, otherwise keeps on counting until reaching a new one.

```
// Initialize Data Log File -------------------------------------------

// Create txt file - - - - - - - - - - - - - - - - - -
String fileType = ".txt";

int i = 0;
int fileCondition;

do {

  fileName = i + fileType;                              // Define file name

  if (SD.exists(fileName))
  {
    i = i + 1;
    fileCondition = 1;                                  // If the file exist, i value plus one, and keep running the loop
  }
  else
  {
    vibeFile = SD.open(fileName, FILE_WRITE);           // If the file doesn't exist, create the file in write mode
    fileCondition = 0;                                  // Kill the loop
  }
} while(fileCondition);

// Print and Write File Status - - - - - - - - - - - - - - - - - -
if (vibeFile){                                          // Boolean true if file exists

  vibeFile.println();                                   // Writes blank line
  Serial.println("File initialized. Prepared to write.");

  bool sdCheck = true;                                  // Used as Check Later
}
else {

    Serial.println("Error creating or opening vibration data log.");

    bool sdCheck = false;                               // Used as Check Later
}
```

Figure 12: Creation of .txt file for data storage

IX.G. Check accelerometer connection and Self-Test Accelerometer

This portion of the code sends the command through GLOB CMD to self-test the accelerometer to ensure normal operation. The DIAG STAT register then checks for errors, which are printed to the serial monitor if found. Results are also written to the SD Card.

```
// Check for Accelerometer ---------------------------------------------

SPI.beginTransaction(settingsADIS);                                        // Begins Com w/ Accelerometer

writeCommand(GLOB_CMD, 0x80);                                              // Reset ADIS:
                                                                           //.. GLOB_CMD[7] = 1; DIN = 1000000
delay(100);                                                                //All the Delays are important to giv

spiCheck = readCommand(DIAG_STAT);    // Read DIAG_STAT value

Serial.println("Checking connection and status of accelerometer...");
vibeFile.println("Checking connection and status of accelerometer...");
Serial.println((uint16_t) spiCheck);                                       // Output DIAG_STAT to serial port fo
if (bitRead(spiCheck, 3)){                                                 // Check for SPI Comm failure

  Serial.println("\tSPI communication error w/ accelerometer.");
  vibeFile.println("\tSPI communication error w/ accelerometer.");
}
else {

  Serial.println("\tSPI communication w/ accelerometer PASSED.");
  vibeFile.println("\tSPI communication w/ accelerometer PASSED.");

  // Self-Test Accelerometer ---------------------------------------------
  writeCommand(GLOB_CMD, 0x04);                                            // ADIS Self Test:
                                                                           //.. GLOB_CMD[2] = 1
                                                                           //.. DIN = 100
```

Figure 13: Check accelerometer connection and Self-Test Accelerometer

IX.H. Setup Accelerometer

Settings are sent to the accelerometer using the assigned byte address and desired hex or Boolean value. The function writeCommand is a user function defined at the end of the code. After settings are sent, an appropriate message is sent to the console and the initial SPI connection is closed. The last command in the setup() is to close the file for smooth transition to loop().

```
// Recording Mode - - - - - - - - - - - - - - - - - - - - - - - - -
writeCommand(REC_CTRL1, 0x1102);                //REC_CTRL1 set to mode 2 of SR0 , Sets Manual Time Ca
                                                //mode set in AVG_CNT
writeCommand(REC_CTRL2, 0xF0);
                                                //.. REC_CTRL1[1:0] = 01 and REC_CTRL1[10] = 1
                                                //.. DIN = 000001000000000
delay(20);

writeCommand(DIO_CTRL,0x0F);                    //Set DIO1 line for busy/data line
delayMicroseconds(20);                          // delay
writeCommand(AVG_CNT,0x02);                     //Set mode 2 of SR0
delay(50);
writeCommand(CAL_ENABLE,0x04);                  //Calibration Enable to decrease the noise
delay(100);

spiCheck = readCommand(CAL_ENABLE);               // Read CAL_ENABLE value
Serial.println((uint16_t) spiCheck);


// Print Result - - - - - - - - - - - - - - - - - - - - - - - - - - -
vibeFile.println("System ready for vibration recording.");
vibeFile.println("X_acce Y_acce Z_acce time Temp");
Serial.println("System ready for vibration recording.");
```

Figure 14: Accelerometer Setup

IX.I. Retrieve flight information from Nanorack

This part of the code is modified based on the example provided by Nanorack. You could find the example at: B2 – Programming/ 03 / Testing & Sample Code / NRNSP Simulator / NRNSP_Sample_Payload / NRNSP_Sample_Payload.ino.

```
void loop(){


  // Connection with nanorack - - - - - - - - - - - - - - - - - - - - - - - -
 char buffer[MAXBUFSIZE + 1] = { 0};                  // Buffer for receiving serial packets.
 int res;                                             // Value for storing results of function ca

 NRdata flight_info;                                  // Struct for holding current flight data.

 memset(&flight_info, 0, sizeof(NRdata));             // Initialize the struct and all its data t

 res = Serial.readBytes(buffer, MAXBUFSIZE);          // Read in the serial data up to the maximu

 buffer[res] = 0;                                     // Null terminate the buffer
 res = parse_serial_packet(buffer, &flight_info);     // Update the current flight info with the

  // Record Data -----------------------------------------------------------

 //if(flight_info.flight_state == 'A' | 'B' | 'C' | 'F' | 'G' | 'H') {        // If want to record the data based on fli
```

Figure 15: Information retrieval from Nanorack

IX.J. Configuration, read and calibrate TimeStamp and Temperature

The configSPI() function is called to configure the SPI settings, and appropriate register data is written to GLOB_CMD register to start the manual capture sequence. Then dataread() is called to obtain the accelerations and the function in turn calls readFFT() to save the values in a 512 short integer variable. Thereafter the TimeStamp and temperature value are obtained and calibrated based on the examples for temperature given in the accelerometer user guide. It is important that each delay stays as it is if the value is decreased then it may lead to runtime error.

```
configSPI();
delay(20);

writeCommand(GLOB_CMD, 0x800);                                    // Start manual capture sequence

delay(250);

dataRead();
timestamp = readCommand(TIME_STAMP_L);                             // Read time value - lower byte only
tempCheck = readCommand(TEMP_OUT);                                // Read TEMP_OUT value
temp = (1278 - tempCheck)*0.47 + 25;                             // Calibrate TEMP_OUT to get temperature in degrees, de

//File in Write Mode
vibeFile = SD.open(fileName, FILE_WRITE);
delay(20);

// In Manual Time Capture Mode, there are 512 data in one run so we save the 512 acceleration by first convertin into integer and
// multiplying by the sensitivity factor. However, there is only one value for temperature for 512 values of accerlation
```

Figure 16: Timestamp and temperature read & write

IX.K. Record, Calibrate and Close

The data is calibrated by dividing it by 2^15 * sensitivity of data. Since we choose 1g so it is simply 2^15 but if the sensitivity is increased then the calibrating value has to be appropriately changed. The logic behind that can be found in the user guide. The values after calibration are sequentially written in the file stored in the SD card along with 1 temperature and 1 timestamp value for the for loop. Thereafter the file is closed to prepare for the next iteration of loop()

```
for (int i = 0; i < 512; i++) {

   // Printing to file for output
   vibeFile.print((short int) X_BUFF[i]*0.000030518);                  // Multiply by accel sensitivity
   vibeFile.print(" ");
   vibeFile.print((short int) Y_BUFF[i]*0.000030518);                  // Multiply by accel sensitivity
   vibeFile.print(" ");
   vibeFile.print((short int) Z_BUFF[i]*0.000030518);                  // Multiply by accel sensitivity
   vibeFile.print(" ");
   vibeFile.print(timestamp);                                         // TIME_STAMP_L value
   vibeFile.print(" ");
   vibeFile.println(temp);                                            // calibrated TEMP_OUT
   //Serial.println((short int) X_BUFF[i]*0.000030518);                  // To review important values i

 }

// Close SD and End SPI Communication once flight finished -----------------

vibeFile.close();                                                     // Closes File
```

Figure 17: Data calibration and writing

IX.L. User Defined Functions - writeCommand, readCommand, readFFT, dataread, configSPI

The writeCommand, readCommand, and readFFTT functions are adopted from ADIS16228 sample program found on GitHub. You could find the original code in the shared drive at: B2 – Programming/ 03 / Testing & Sample Code / ADIS16228_Teensy_Example/ ADIS16228_Teensy_Example.ino. With the help of Dr. Anthony Cofer we were able to modify the functions to read and write the desired value of the accelerometer. writeCommand function allows tehuser to set speicifc value of a register, the dataread function calls readFFT function which in turn calls readCommand function 512 times to access the 512 value of the acceleration in the three direction. readCommand function in general allows user to access the value of a specific register. configSPI function configure the SPI before the data is sent or received through it.

IX.M. parse_serial_packet User Defined Function

This part of the code is modified based on the example provided by Nanorack. You could find the example at: B2 – Programming/ 03 / Testing & Sample Code / NRNSP Simulator / NRNSP_Sample_Payload / NRNSP_Sample_Payload.ino. This function will allow user to access flight information provided by Nanorack.

X. Conclusion

When the Fall 2018 team was assigned the project, the main goals were to test the device using the Summer 2018 code to ensure it was functioning properly, to record timestamps, and to transfer the device to the new baseplate. However, the Fall 2018 team quickly realized that the recorded data was incorrect and that the accelerometer was essentially outputting 'garbage' values. Thus, the Fall 2018 spent a significant amount of time learning how to code for the ADIS 16228 and reworking the Summer 2018 code in order to get reliable data. One of the key issues found was that the default state of the reset pin on the board was 'on', which caused the system to reset each time the code was run. The fix for this was to simply detach the wire from the reset pin until we found a way to set the reset pin to 'off' within the code, allowing us to reconnect the wire. Secondly, interrupts had to be added to allow the accelerometer to have enough time to take samples and send that data to the datalogger. It was also discovered that the data types were inconsistent throughout the code, so all data types were changed to integers. The time capture mode of the accelerometer was changed from automatic FFT to manual time capture to allow for more samples to be taken, rather than have the microprocessor take that time to perform an FFT. An FFT analysis can be done post-flight using software such as MATLAB with the recorded data.

The assessment of the Fall 2018 team is that the PVIPER project is near completion. The device now successfully records and outputs correct accelerations along the x, y, and z axes, temperature, and timestamps. All necessary hardware for the experiment has been verified and assembled. The electronic components chosen by the Fall 2017 team have been reviewed and verified by the Spring 2018 and Summer 2018 teams. The main task that remains is to implement the Nanorack system within the code in order to have the system run autonomously during flight. Nanorack is not fully implemented in the Fall 2018 code, but an attempt to implement it can be seen in the Summer 2018 team's code. It is unclear if the Summer 2018 Nanorack implementation works correctly or not, but it may serve as a good basis from which to work. Secondly, the device must be transferred to the new baseplate and the new wire connector must be installed on the datalogger. Another thing to keep in mind is that on occasion, when the baseplate is kept flat and unmoving, the acceleration output in the X direction is around -0.5g so an effort needs to be made to correct this calibration error.

XI. Recommended Next Steps

There are few things need to be completed:

1. The acceleration data is saved in raw format, as integers, and it needs to be calibrated. The calibration documents could be found on the User Manual of ADIS 16228 Page 16.
2. A new wire connector must be implemented and the device must be rewired, since the current connector is faulty. Fall 2018 team has ordered the connector, so it should be available for use. It was decided to use a Ribbon Cable, which is more robust at the connector than the crimped one used before.
3. The device must be transferred from the old baseplate to the new, properly made baseplate.
4. Check if the acceleration in X direction makes sense when the base plate is kept on a flat surface.

XII. Helpful Notes

1. You will need to download Arduino IDE to open, edit and test the code, so it is recommended to use personal laptops or computers when working on the code. Here is the link to where you can download it: https://www.arduino.cc/en/Main/Software
2. Several drivers are required to get the datalogger to communicate with Arduino IDE. Step-by-step instructions on downloading and configuring them can be found here: https://learn.adafruit.com/adafruit-feather-m0-adalogger/using-with-arduino-ide
3. It may take time to understand SPI communication and how it works with respect to the components used in this project. A helpful in-depth explanation of SPI and programming for it can be found here: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all
4. Dr. Anthony Cofer was a huge help in progressing with this project thus far. Should you need help with the project it is recommended to reach out to him at acofer@purdue.edu.

XIII. Acknowledgement

The Fall 2018 team would like to sincerely thank Mr. Michael Hayashi, Mr. Steven Pugia, and Dr. Anthony Cofer for their invaluable inputs and precious time that enabled us to correctly obtained data from the accelerometer in spite of our lack of experience in working with MEMS and SPI communication. We would like to thank Prof. Steven Collicott for his constant support and input without whom this project would not have been possible.