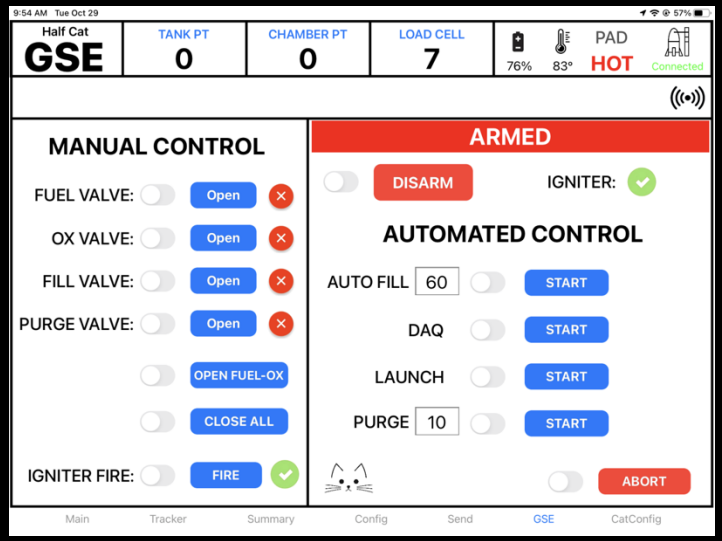
ROCKET TALK

HalfCat
Custom GSE Controller & DAQ



Overview and Design

Version 1.0 – February 2025

Mike & Preston Brinker

Why build a GSE controller for HalfCat Rockets?

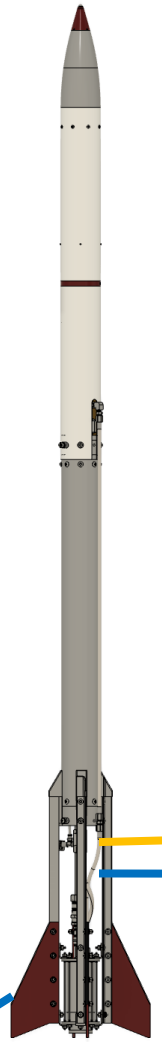
The commercial GSE controller design that is detailed in the HalfCat HCR-5100 Mojave Sphinx guidebook **works well**. It is battle tested with a combined 50+ static fires and launches. It is infinitely easier and cheaper to setup than building your own GSE Controller or DAQ system. HalfCat has always been optimized for low cost, ease and accessibility, and is a great entry point to learn liquid rocketry.

We built our custom HalfCat GSE and DAQ only because we had built ten previous versions for other liquid rockets. It only required minor edits to our existing PCBs, software, and iPad remote. Also, our overall "RocketTalk" project includes a full featured flight computer talking to our hand-held base, as well as the GSE, so we have one integrated hand-held remote talking across three radios to the pad and the rocket. This allows us to fill, launch, and track the rocket all in one, although the GSE Controller and code is separate and can be isolated, especially when doing static fires or testing not involving flight.

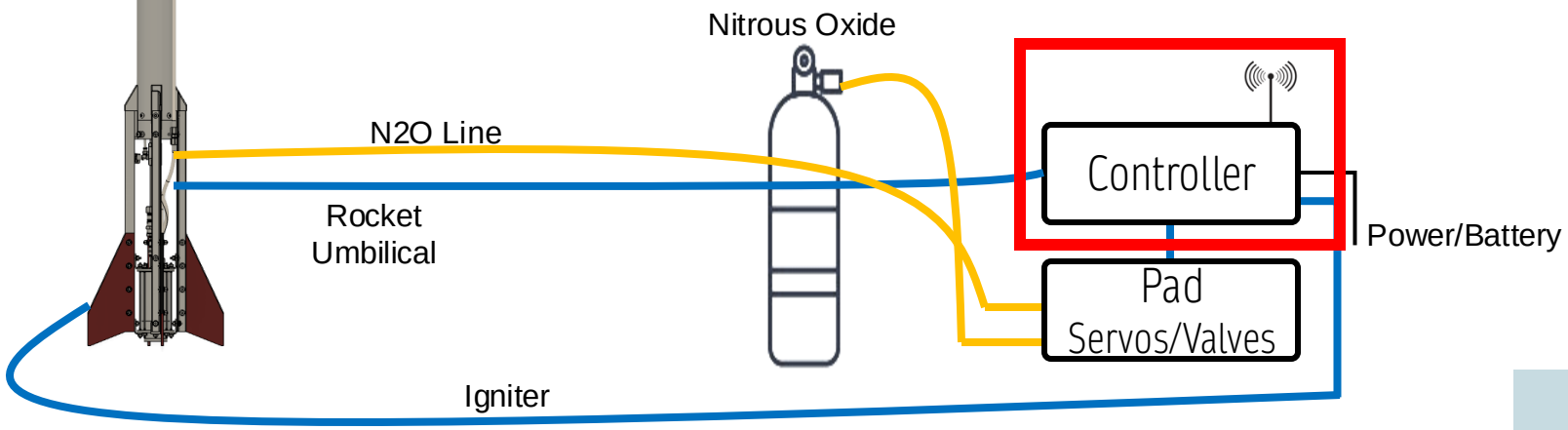
That said, if you have an amateur EE and some code junkies on your team and you want your own custom GSE controller or DAQ then our open sourced solution can provide you with some good ideas and a starting point. We intentionally built everything with the amateur maker in mind – including Arduino source code, EasyEDA schematics and PCB design, and Apple iOS Swift code. Building your own GSE controller could also be a fun upgrade for a team that has already built and launched their first Mojave Sphinx.

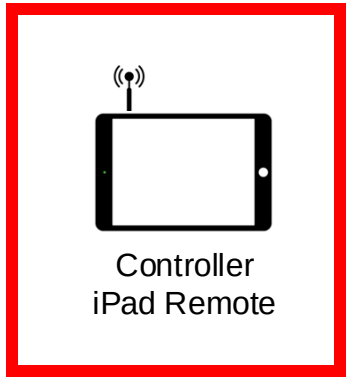
As an added disclaimer… Our HC GSE Controller was developed/modified over the course of 8-10 weeks to support our Mojave Sphinx build, but it is still very much a WIP. We've decided to take a snapshot and share our learnings, so student groups can learn, borrow, or reuse some of our ideas and our sloppy code.

GSE OVERVIEW



SAME PLUMBING: It should be noted that we used the exact same servos, valves, plumbing, and fittings as specified in HCR 5100 for the Mojave Sphinx GSE. The only addition in our setup is the custom GSE Controller with radio remote and a permanently mounted pressure transducer on the rocket tank (connected back through the umbilical).





OPEN GSE PROJECT

Below are the general design principles that drove the development of the controller and remote:

1. MINIMAL DESIGN – MAXIMUM FEATURES

Simplified and compact design with minimum features to load, pressurize, launch, and static fire a HalfCat rocket, integrating into the overall design of Mojave Sphinx.

2. BASIC COMPONENTS AND IDE'S

Developed using Arduino-based MCU (Teensy 4.1-Arm7), EasyEDA PCB layout, and an iPad tablet remote (Apple xCode IDE w/Swift).

3. SIMPLIFIED USER INTERFACE

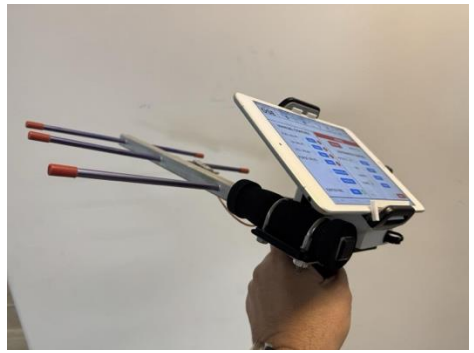
Custom remote provides simplified UI and error controls to minimize the human risks related to loading/pressurizing, and launching and DAQ data capture.

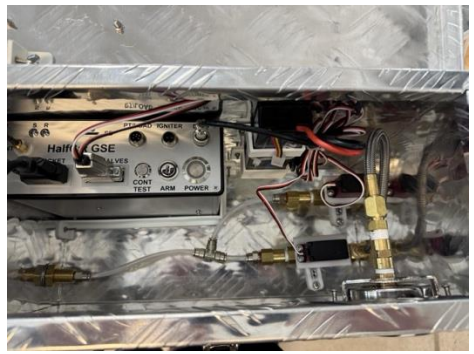
4. SAFETY AND ERROR SYSTEMS

Automated processes in the controller for loading and launch, over-current and fault protections, error checking, and more.

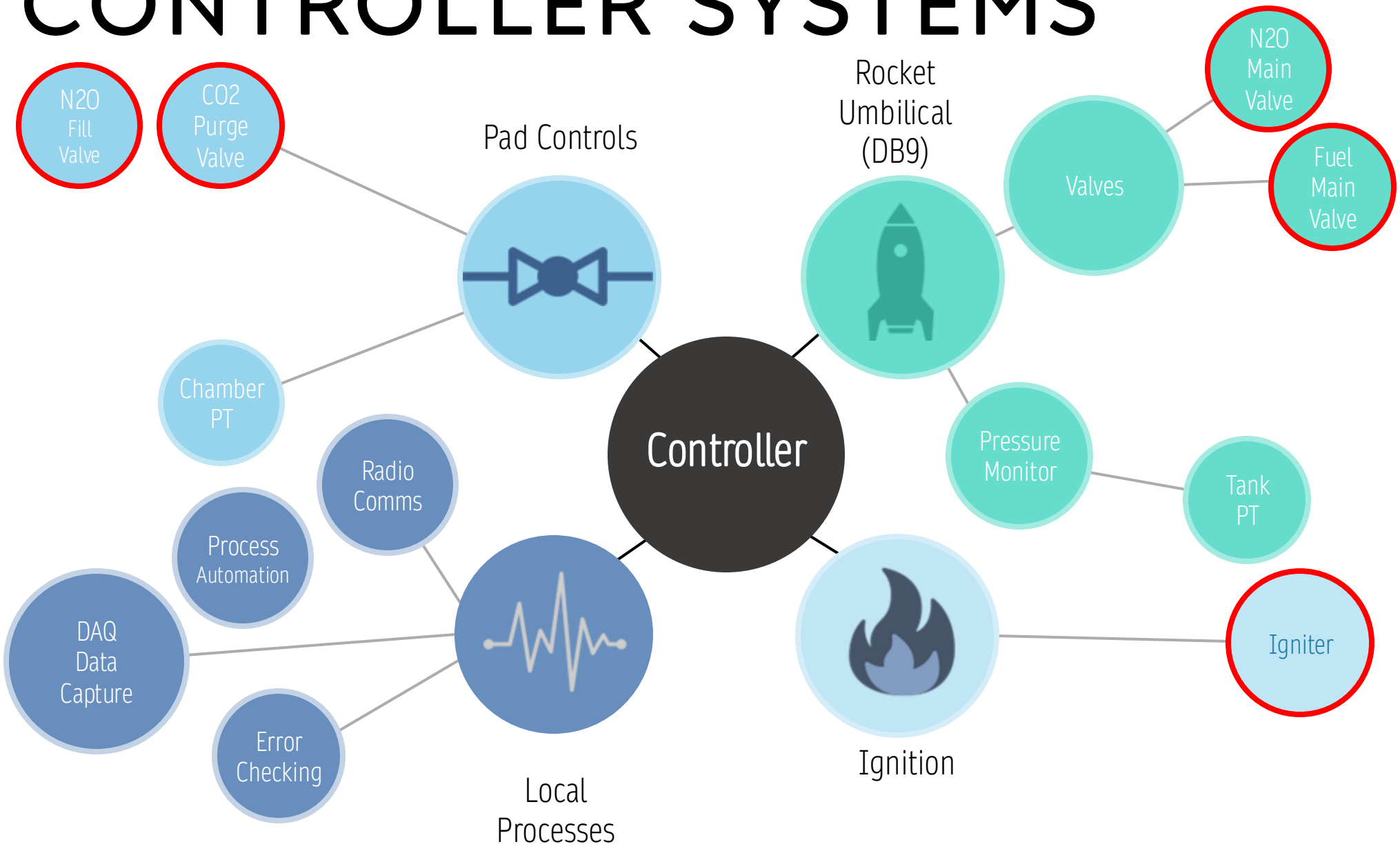
5. OPEN SOURCE

Code and hardware design available to US University student teams.
www.RocketTalk.net





CONTROLLER SYSTEMS

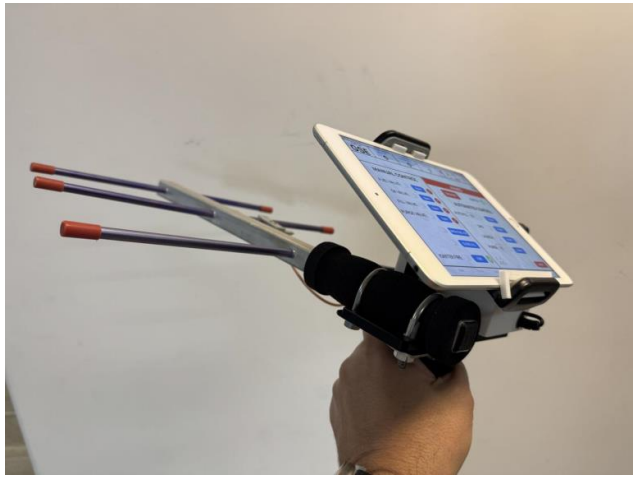


Hazardous

SYSTEMS FEATURE SUMMARY

GSE Controller

- Control of two pad servo valves for N2O fill and CO2 purging.
- Control of rocket N2O and fuel main valves
- Monitoring/recording of rocket tank PT
- Monitoring/recording of chamber PT (static fire)
- Igniter firing and local continuity check
- Full featured three channel DAQ recorder for static fire analysis (tank PT, chamber PT, load cell)
- Hazard power safety key
- Automated process tasks for loading and launching to ensure exact timing



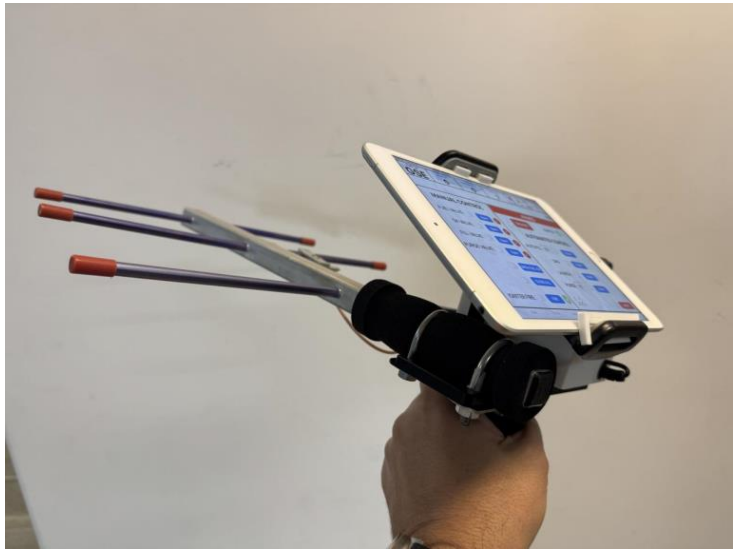
iPad Touch-based Remote

- Two tab-based control screens including:
- Main GSE Screen:
  - Manual valve and igniter control
  - Automated processes (launch and load)
  - DAQ control
  - Arming & Disarming
- GSE Configuration Screen:
  - Launch/Static fire mode
  - Set servo stops
  - Zero PTs
  - Set PT ranges

TECHNOLOGY SUMMARY

GSE Controller

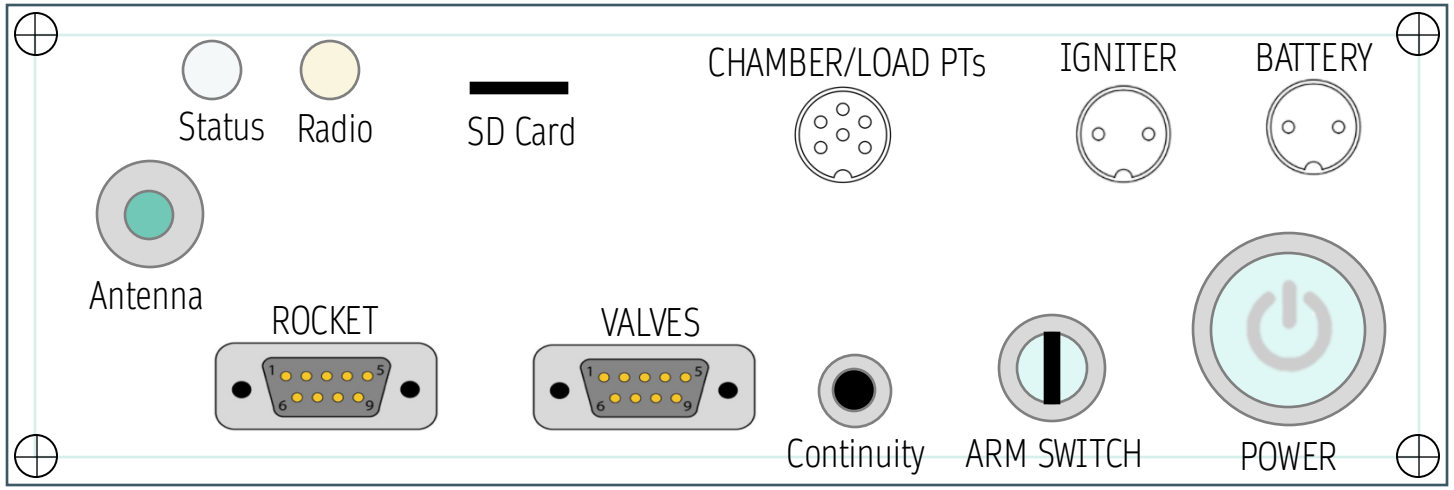
- Teensy 4.1 Arm7 Microcontroller
- SD card for DAQ and command logging
- Arduino IDE / Single C++ Sketch
- NiceRF LoRa 611Pro 433Mhz serial TTY radio (or use 900 Mhz)
- Uses 12.6v Lipo battery
- ADS1115 four channel 16bit ADC
- DB9 connectors for rocket umbilical
- DB9 connectors for pad valves
- 6 pin avionics connector for chamber PT and load cell amplifier (separate)
- CPU heatsink strapping for thermal management
- Custom PCB developed using EasyEDA and printed through JLCPCB



iPad Touch-based Remote

- Standard iPad mini (wifi only)
- Uses RedSerial cable to interface with NiceRF LoRa radio
- -or- Custom Bluetooth bridge using an ESP32 connected to a NiceRF LoRa 611Pro 433Mhz serial radio
- Apple xCode IDE with iOS Swift code
- 7.2v lipo battery and iPad battery

HARDWARE – EXTERNAL I/O



A word about servos

This was our first project controlling servos with an Arduino in a “rugged environment”. All of our past GSE controllers were directly powering solenoid valves with high current drivers or signaling using an ethernet bus. Servos use Pulse Width Modulation (PWM) to communicate between the MCU and the servo. The Teensy 4.1 has lots of examples and libraries, but ultimately there are variances and nuances with each servo that require you to set the stops for your preferred positions (like 0 and 90 degrees for opening/closing a valve).

The other issue you run into with servos is over-heating and burn-out, especially when working with quarter turn valves. When the MCU sends a command to the servo to move 90 degrees it will keep moving until its internal encoder says it has reached 90 degrees. So, if your hardware shifts a bit on the rocket and now when your valve is closed the servo is at 89 degrees, you will think the servo finished, but in reality it is sitting there quietly burning 2 amps and trying desperately to go another 1 degree. Eventually it will either burn-out or your battery will die. This is why servos are notorious for burning out.

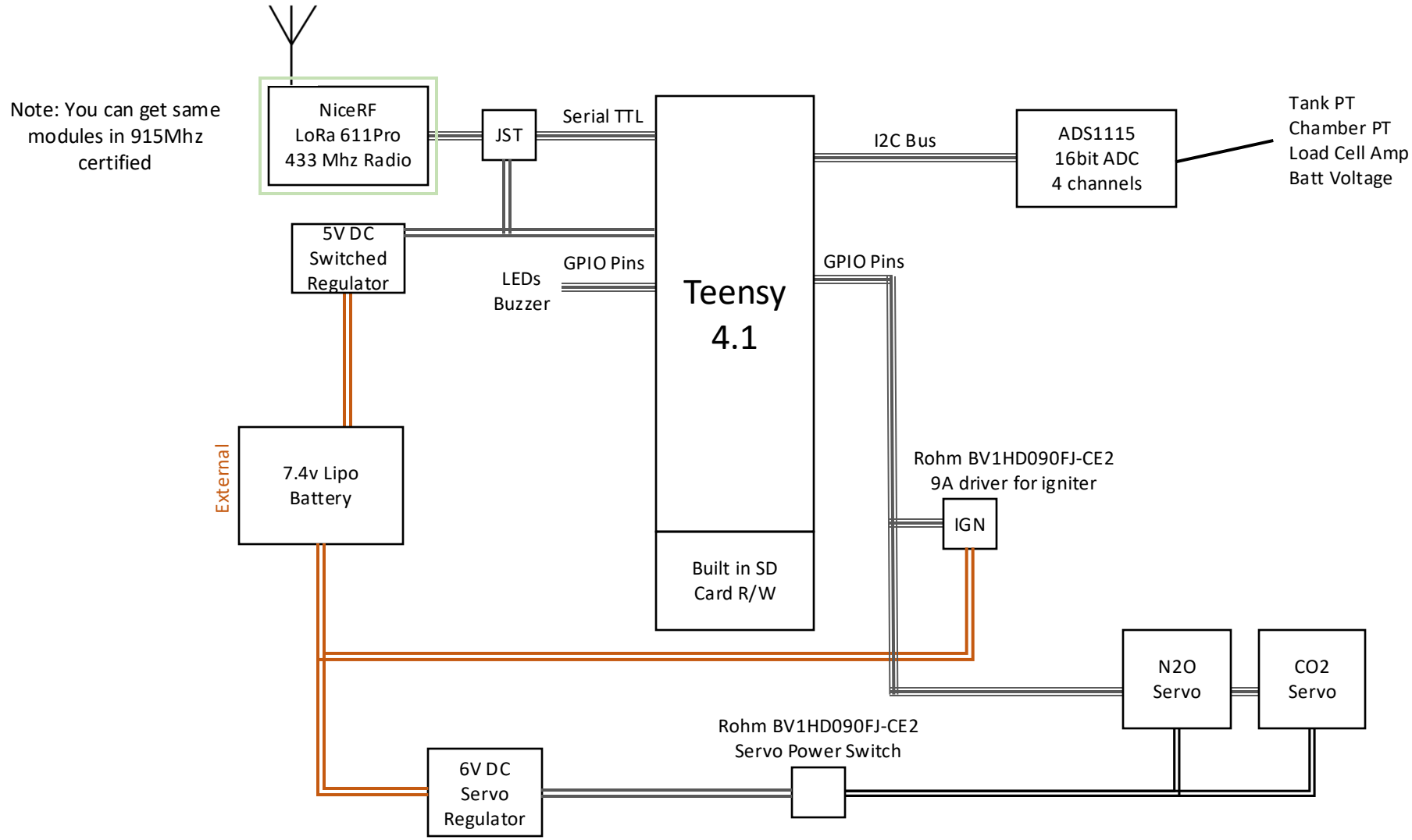
So we store a configuration file and make it easy in the configuration UI to **set the stops for each servo** on each valve. We also use a solid state switch to turn on the servo power when we want to command it and then turn the power off two seconds later, preventing the servos from overheating or consuming battery. The servos do not need to stay energized once the valves have opened or closed. This protects them from over-heating or drawing too much power.

Also, we found that the servos operated about 20% fast at 6v vs. 5v, so our GSE is using a 6v regulator. Every millisecond counts when opening the main N2O and Fuel valves on the rocket, to make sure they are flowing well while the igniter cartridge is still burning. The extra voltage lowers the open/close time by about 30ms.

ROCKET TALK

GSE CONTROLLER DETAILS

GSE Controller – Conceptual Architecture



Current Technology Choices – Controller PCB

- **Teensy 4.1:** The T4.1 with its Arm Cortex-M7 is just insanely fast for an MCU and it is loaded with a lot of versatile GPIO. The T4.1 also has a built-in SD card for easy data recording (used with an extender). The fast MCU allows us to sample four channels of data at 200Hz for static fire testing. We compile at a slower clock speed (384 Mhz) for cooler thermal performance.

- **Radios:** We have tried a number of different radios, but we are very happy with our NiceRF LoRa 611Pro radios on 433 Mhz (FCC amateur radio license required). You can get similar radios in 900 Mhz that don't require a license. The Lora technology is great and is very reliable. Order TTY versions and use on serial I/O.

- **ADS1115 16bit ADC:** We use one four channel ADC. One channel is for the tank PT, one for the chamber PT, one connects to the load cell amplifier (separate circuit), and one monitors the battery voltage. The ADC operates over I2C.

- **Igniter Driver - Rohm BV1HD090FJ-CE2:** This is a single channel 8A high-side driver used for triggering the igniter. It also provides excellent over-current protection, but doesn't shut-down when firing an igniter. It has a status pin that provides a safe continuity check.

- **Servo Power Driver - Rohm BV1HD090FJ-CE2:** We use another Rohm driver to switch the servo 6v power. Each time we move a servo to open or close a valve we energize it for two seconds and then de-energize it so there is no risk of overheating or damaging the servo.

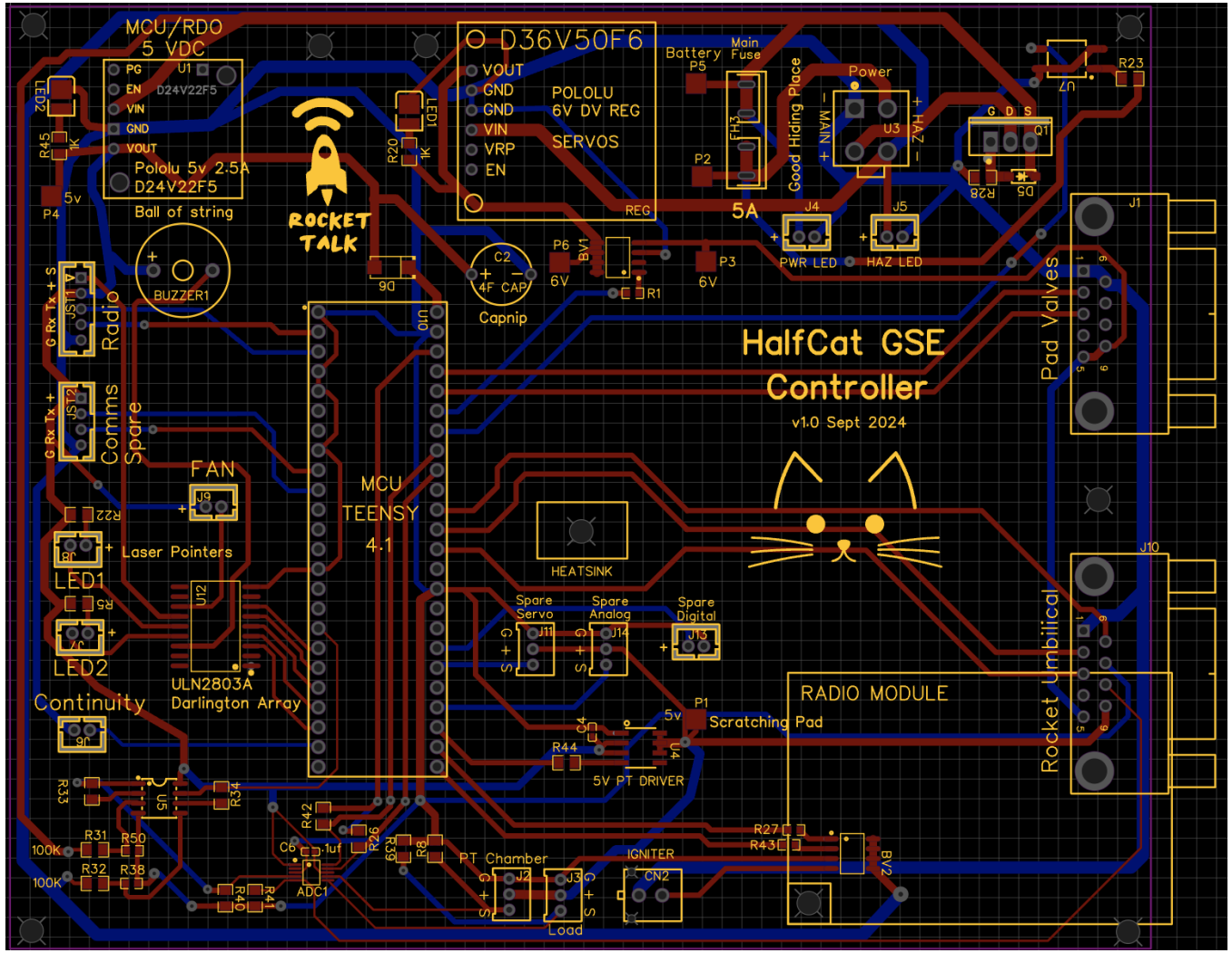
- **5V Regulator:** We use a Pololu 5v 2.5A switched regulators (D24V22F5) to power the Teensy, external PTs, and the Radio.

- **6V Regulator:** We use a Pololu 6v 5.5A switched regulators (D36V50F6) to power the servos. Running the servos at 6V helps them operate close to their maximum speed.

- **Reverse Polarity Protection**: We use a P-Channel Mosfet (IRF9Z24PBF), along with a Zener diode (SZMMSZ5246BT1G) and 10K resistor to provide reverse polarity protection for the PCB.

- **Other**: We use Vishay VO1400AEFTR Optoisolated relays to report fuse status for the one fuse

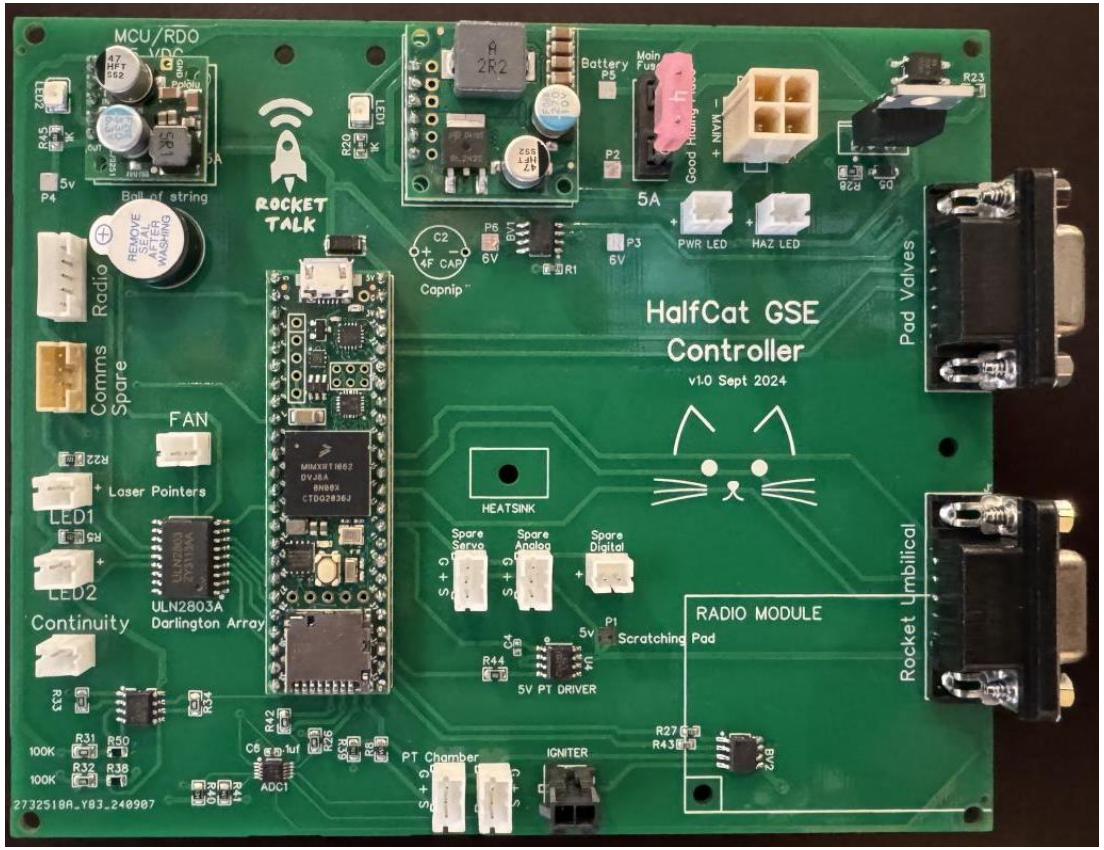
Controller Board – Custom PCB



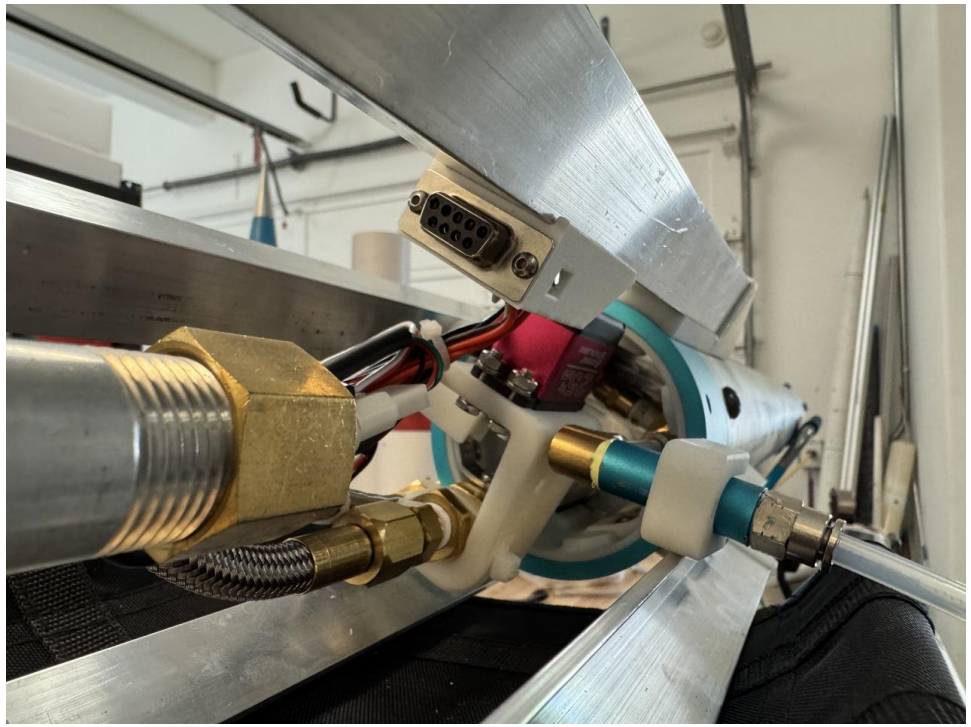
The schematic and custom PCB are designed in EasyEDA and printed and ordered through JLCPCB (via EasyEDA). It really is easy.

The PCB is a two-layer board printed with 2oz traces for added current capacity.

Controller Board – Custom PCB

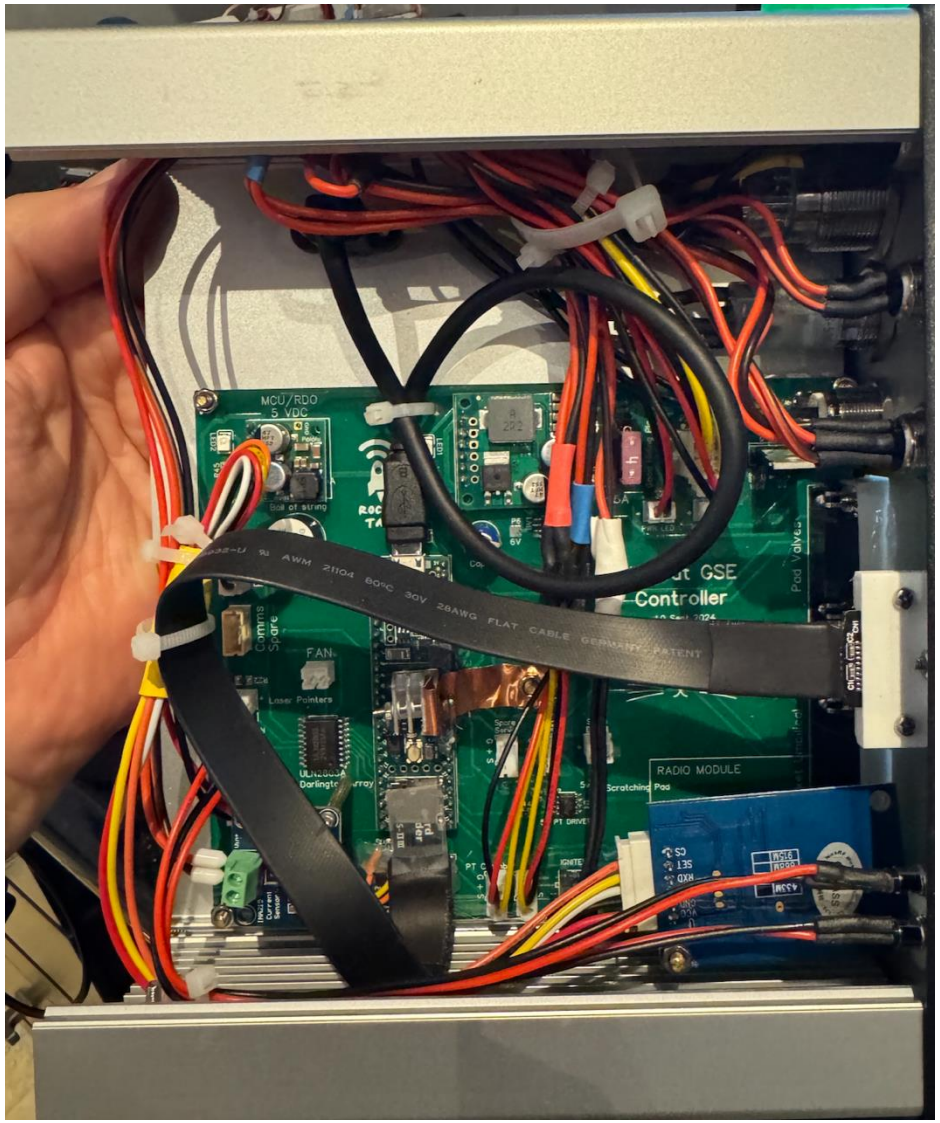


The loaded PCB. Note the radio module gets mounted in the lower right and the thermal strapping for the CPU (case bonded) is not yet added in this picture.



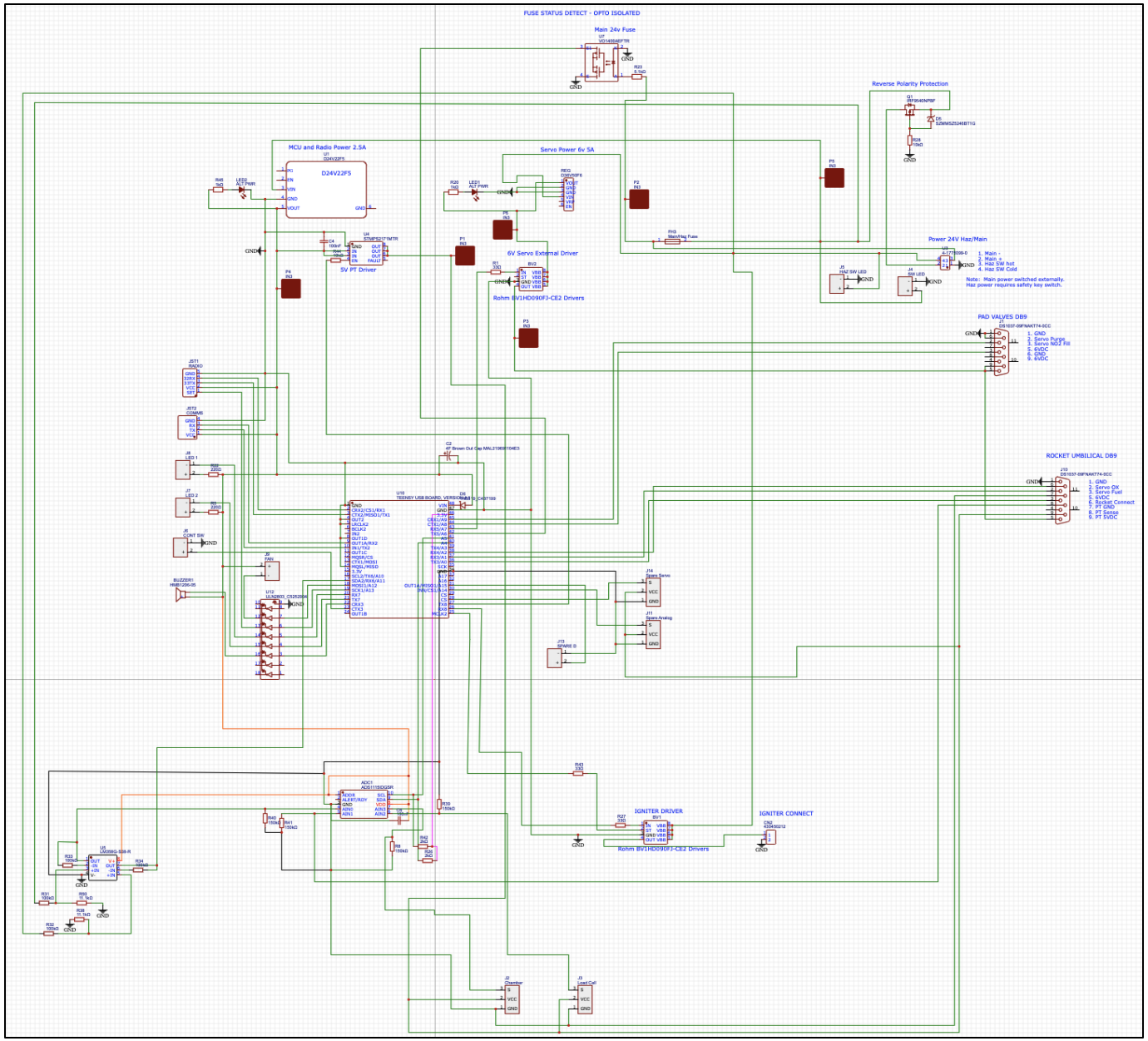
The DB9 connector on the rocket – connects to two servos and the tank pressure transducer

Controller Board – Custom PCB



Fully wired inside a Hammond aluminum box
(p/n 1455U1601 - 160mmx191mm)

Controller Board – Schematic



Controller Board – Code Design

The Teensy code is built from a single C sketch in the Arduino (Teensyduino) IDE. It utilizes some external libraries for the ADCs and logging, but is otherwise self-contained in a single code file.

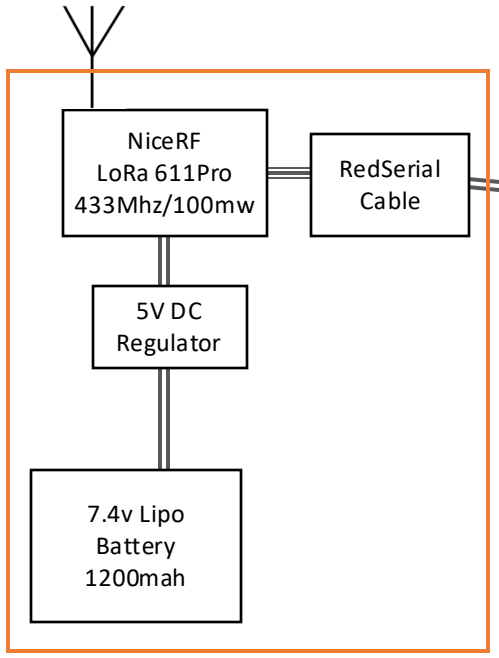
We used to use Teensy Threads to partition the code into three parts, but we've found the code is more reliable when run as a single sequential sketch and the Teensy processor is more than fast enough to sample and record at 100+ sps/Hz for DAQ logging.

Each of the activities in the main loop are throttled using timers, so the highest priority tasks, like sampling, process faster and more often. Here is the general order of operations:

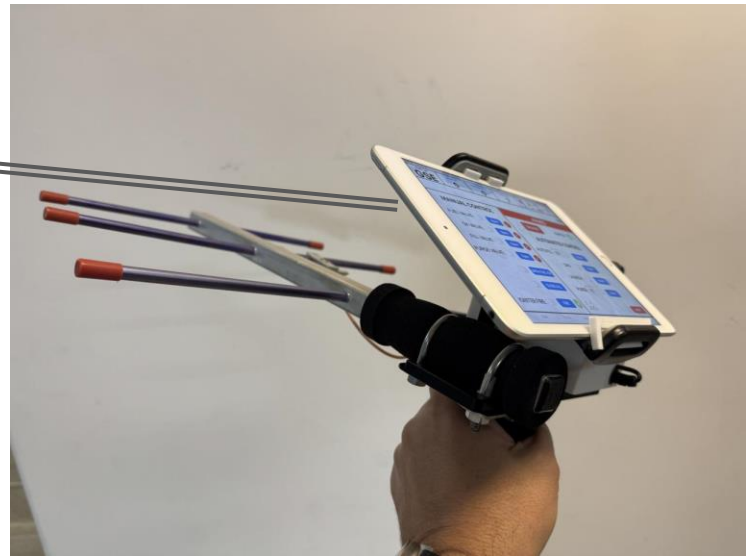
- Check radio (receive)
- Check radio (send)
- Sample
- DAQ Record
- Housekeeping
  - Check Pressures (for alerts)
  - Fuse check
  - Rocket connected check
  - Igniter continuity check
  - Voltage check
  - Manual continuity test switch
  - Check automated processes
- Error/Health check
- Send full status to remote (every 5 sec)

iPAD REMOTE

iPad Remote – Conceptual Architecture

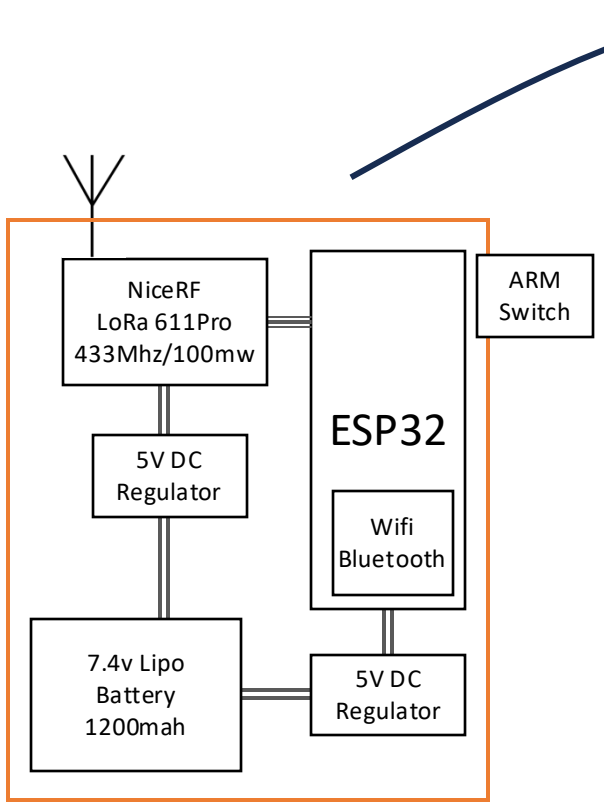


This version is using a direct RedSerial cable



Standard iPad Mini 4 (wifi only)

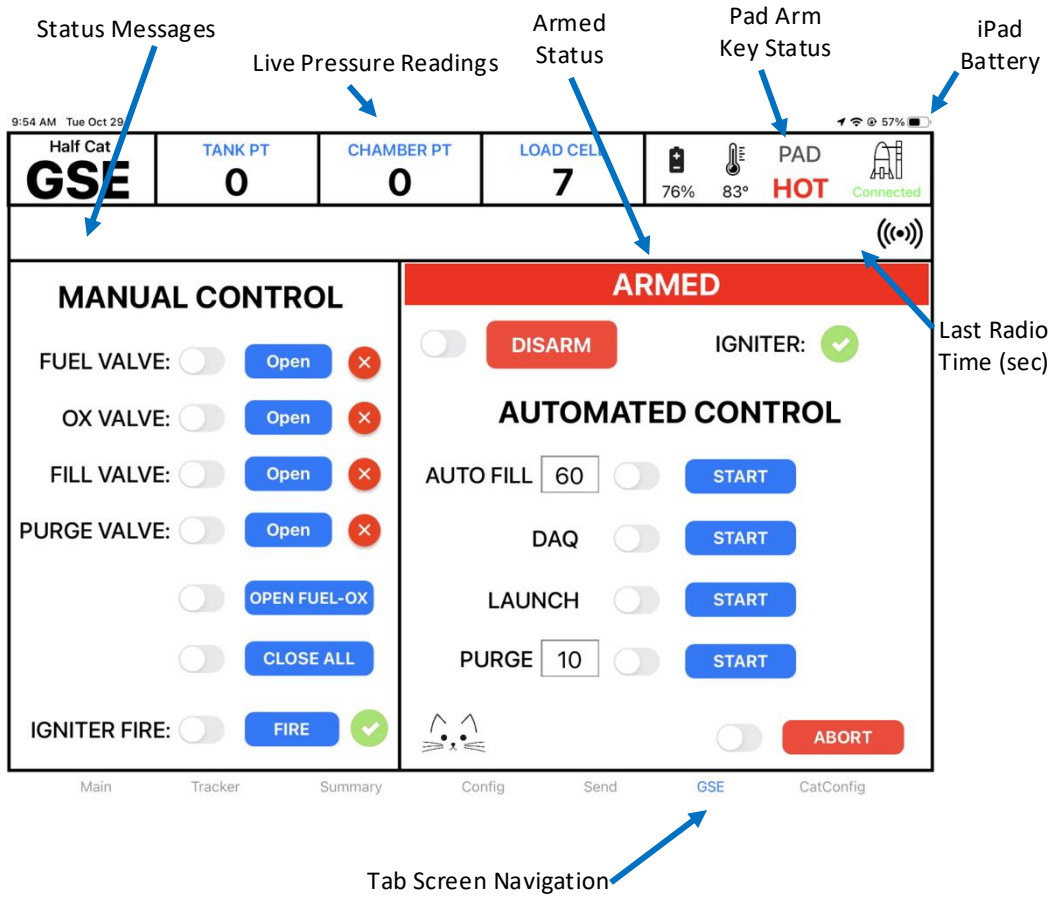
iPad Remote – Optional Bluetooth

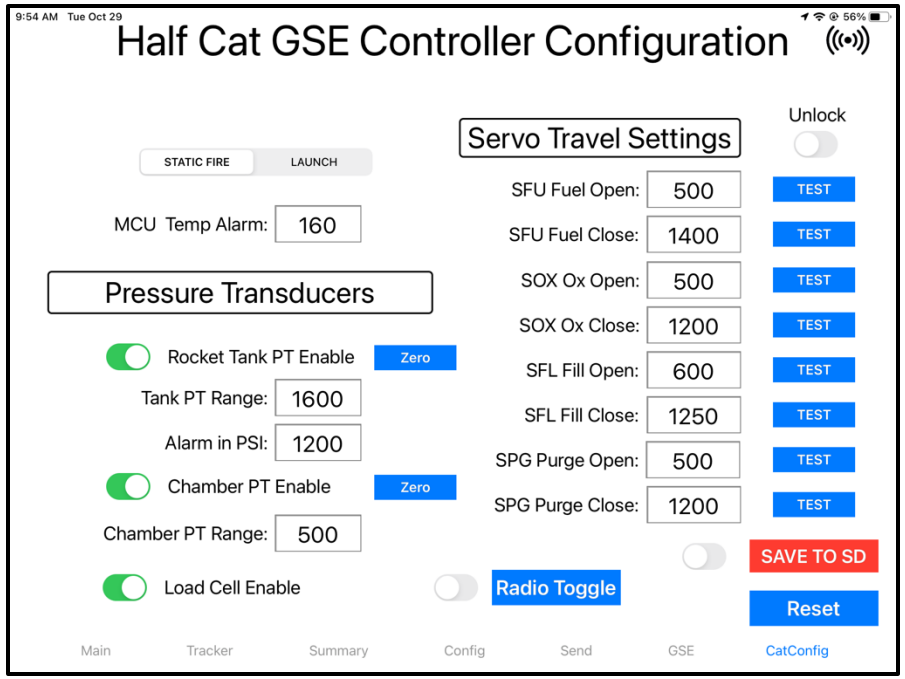


A Bluetooth / Radio Bridge Option is also available (contact for details)

Showing another version of our GSE for large liquid bipropellant rockets

iPad Remote – Screens





Current Technology Choices – Remote

- **iPad Mini:** When we decided to develop the remote using a touch-based tablet we chose an Apple iPad mini, due to the ubiquity and longevity of the Apple product line. It also provides an extremely intuitive UI which was a major design principle. The iOS Swift IDE (in Apple's xcode) is free and easy to learn, but to fully exploit Apple's development environment a $100/year developer fee is required.

- **Bluetooth Bridge Option:** Our original designs used a proprietary cable (Redpark) to connect the iPad to the serial TTY LoRa radio, but it is proprietary and difficult to get. The Redpark cable also was not Apple sanctioned and their primary solution was to interface to the iPad using Bluetooth. So we built a small Bluetooth "bridge" from an ESP32 using a small Arduino sketch. The ESP32 receives data from the LoRa serial radio and then passes it to the iPad via Bluetooth. This has proven to be extremely reliable. It also gave us the ability to add a physical key switch to the iPad for Arming the controller.

- **Battery with Bluetooth**: EMEPOVGY 2S Lipo Battery Airsoft 1200mAh 7.4V 20C

- **Other:** Our first version of this solution used a full-size 9" iPad, but we found that an iPad Mini worked just fine. Using a project box from Polycase (VM-35MMTT) provides a great one-handed solution. Also the iPad case from Seymac provides good protection in the rugged environment.

ROCKET TALK

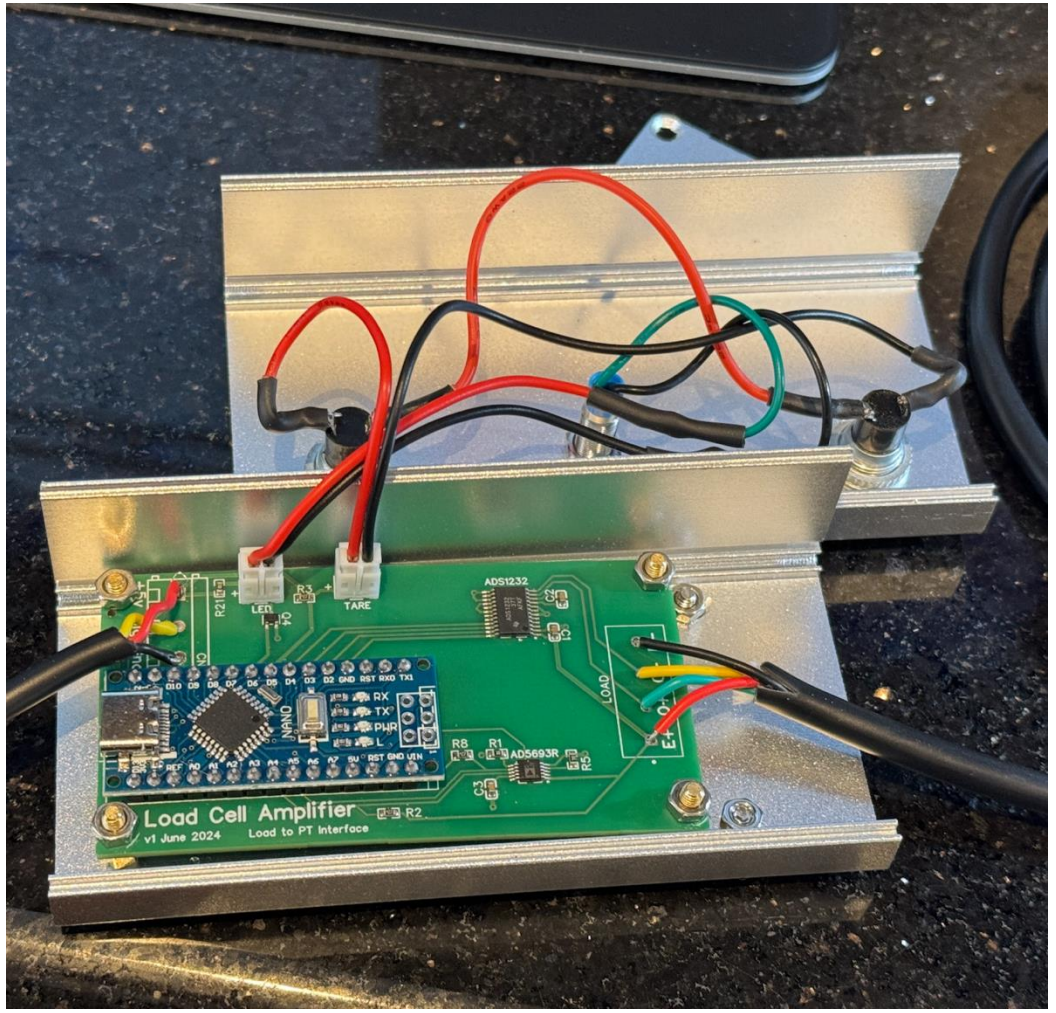
Load Cell Amplifier

LOAD CELL AMPLIFIER

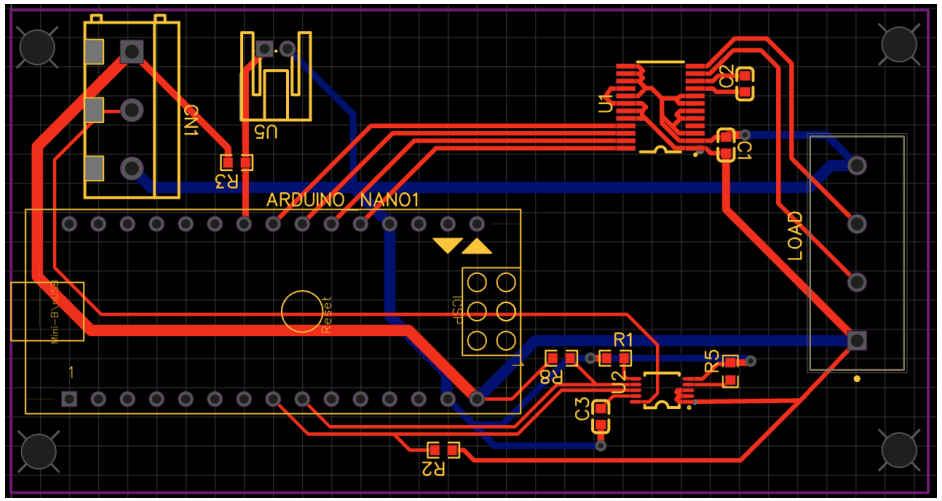
A strain gauge load cell requires an amplifier, since the output is in tiny millivolts, so you will need to wire an amplifier close to the load cell (within 2-3 feet) and then run a higher voltage signal line back to your DAQ.

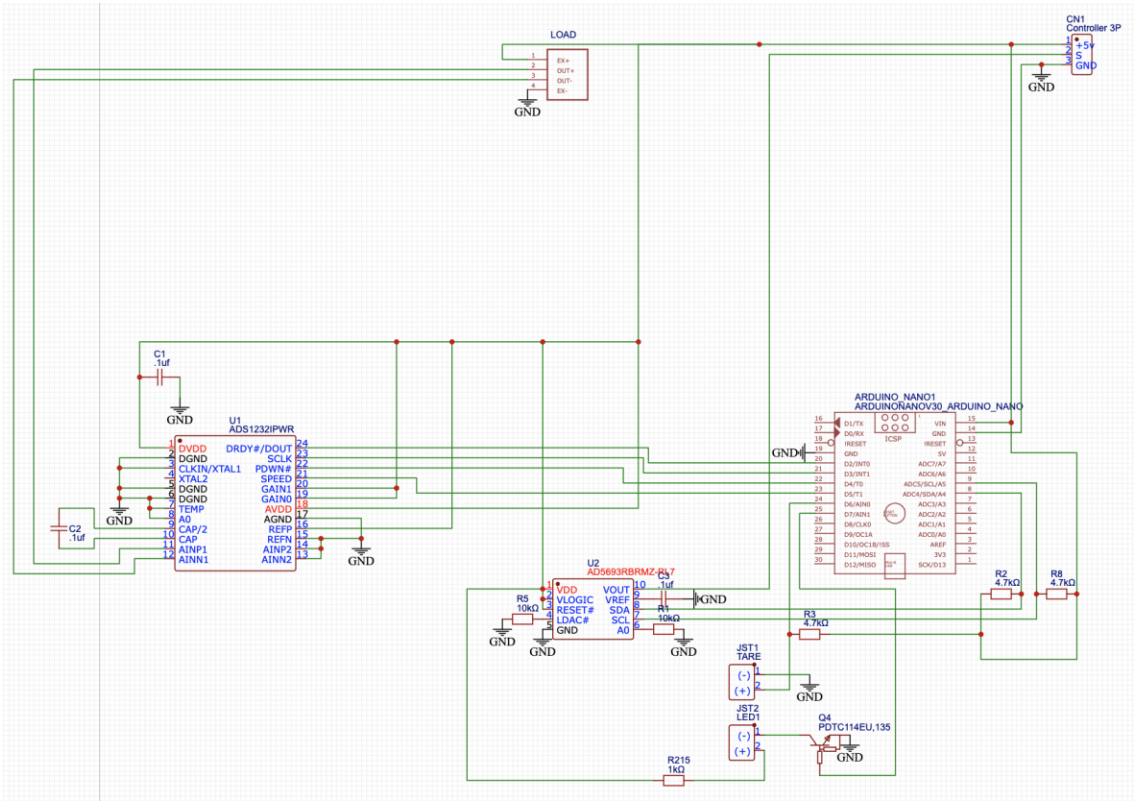
In our case, we made a custom amplifier circuit that will feed into our 5v ADC in the GSE Controller by sending 2.5v (2500mv) to the GSE when reading zero. It then increments one mv per pound for positive force and decrements one mv per pound for negative (hanging weight of the rocket/propellant). This allows us to better capture load taking into consideration the weight of the rocket and expending propellant over time. We log to SD 2500mv +/- 1mv for each pound of force.

It is built using an Arduino Nano and an ADS1232 ADC (connected to the strain gauge) and a AD5693R chip to send back the 0-5v signal to the controller.



LOAD CELL AMPLIFIER





Disclaimer:

The open-source code and hardware designs ("Materials") provided here are related to the development and launching of bipropellant liquid rockets. The authors and contributors of these Materials provide them for educational and informational purposes only.

WARNING: LAUNCHING LIQUID ROCKETS IS AN INHERENTLY DANGEROUS ACTIVITY. The use of the Materials involves significant risks, including, but not limited to, serious physical injury, disability, and death. Anyone using, modifying, or distributing these Materials must understand and accept these risks. If you choose to use these Materials, you do so at your own risk.

The authors and contributors of these Materials make no representations or warranties of any kind, express or implied, about the completeness, accuracy, reliability, suitability, or availability with respect to the Materials. Any reliance you place on such information is therefore strictly at your own risk.

In no event will the authors or contributors be liable for any loss or damage including without limitation, indirect or consequential loss or damage, or any loss or damage whatsoever arising from loss of data or profits arising out of, or in connection with, the use of these Materials.

The authors and contributors disclaim all responsibility for any injuries, damages, or losses of any nature whatsoever, whether direct, indirect, punitive, incidental, special, consequential, or any other damages, to persons or property, arising out of or connected with the use, misuse, or inability to use the Materials provided here.

BY USING THESE MATERIALS, YOU ACKNOWLEDGE THAT YOU HAVE READ THIS DISCLAIMER, UNDERSTAND IT, AND AGREE TO BE BOUND BY IT.

Questions?

@PAD33space on Twitter/X
@AllDigital on TRF
info@rocketTalk.net