**Renegade/BLT Vehicle & Static Fire Stand Avionics Architecture**

Top Level Requirements

* The system must have local nodes on the stand/rocket capable of the following
  + Control Solenoid Valves
  + Read Analog Sensors
  + Log Data
  + Communicate over a wired bus with external node(s)
* The system must have the ability to be controlled via nodes in Launch Control
  + Command Solenoid Valve actuations
  + Receive sensor data
  + Interface with Launch Control personnel for monitoring and control
  + Log Data
  + Communicate over a wired but with the onboard nodes
* The system must have the following critical safety features
  + Ignitor control is human in the loop only, not controlled in any way by the nodes.
  + All functions deemed a risk to personnel have hardware lockouts
    - Can be via nodes and their output states
    - Also have the possibility to cut power to safe state from launch control

Node design Rev. 1.00

* Teensy 3.5/6 based system
* Must be able to read the on board ADC
  + The inputs capable of reading differenced signal read Thermocouples (two pairs each Teensy)
* Must be able to read external ADC chips over SPI
* Must be able to assign arbitrary sample rates to each analog input (within possible hardware specs)
* Must perform basic bit chopping or conversions on data samples
* Must package samples into CAN frames and send on CAN bus
* Must listen to CAN messages for control inputs (actuate solenoid valves, toggle safety enables, arm autosequence, et cetera)
* Must send a state report on CAN at regular intervals (10 Hz preliminary rate)

Notes:

Teensy uses the FlexCAN library to access the built-in CAN controller. Which Teensy generation you’re using determines which FlexCAN. The original library is by teachop, the 3.x variation is by pawelsky, and there is another one for 4.x I don’t recall which one at the moment but it adds the CAN FD support. The APIs on each are somewhat different.

This is the original, not the one we are using in this case

<https://github.com/teachop/FlexCAN_Library>

This is the correct one for the 3.6

<https://github.com/pawelsky/FlexCAN_Library>

The built in Teensy ADC library is this:

<https://github.com/pedvide/ADC>

The Adafruit temperature sensor I followed their instructions to setup and read fairly closely with stripping out a little unnecessary code for my use.

My code is mostly based on the examples from the libraries I’m using and ripping out only the useful parts. I can help to further reconstruct what I’ve done but I will have to go back and look at what examples I followed specifically if that’s necessary.

The initial V1 system that is run with a pair of 3.6 nodes to run the static fire stand can fit all the requirements barely with the GPIO as is. The main challenge is the shared pins for analog inputs, PWM, and CAN.

* The analog inputs that are able to be read as differential must be used for that
* CAN pins must be used - Run everything over CAN0 on each node for now, but reserve pins for CAN1
* Each node gets one I2C temp sensor
* Reserve at least one SPI device pins
* Allocate 8 (each node) PWM outputs using pins 2,5,6,7,8,9,10,29,30
* Allocate other ADC inputs, currently only 6 (each node) in addition to the 2 differential TCs
* 2 general GPIO pins used as digital outputs to drive the solenoid valve chip input enable. There is a hardware pull down resistor, so this needs driven high when the solenoid valves can be active. Note-Teensy is inverted logic scheme (set low in code means drive high)

Non Teensy Node Notes:

Don’t worry about the arming and safety stuff yet. That’s all covered by the CAN code architecture we have set up and the specific operation of it I can add in later or we can work together on.

The Log Data requirement on the Teensy node is a future item that is not necessary on first revision. By the time we use this for BLT testing it will be required but not the initial static fire stand test for Renegade. This is to log the full unchopped data for everything locally for accessing after test/flight. We are assuming we will blow this up, hence the bus and telemetry systems, but in cases where it’s possible the best data for troubleshooting will come from recovering off the node itself.

I still need to provide the sensor functions as the last major set of requirements.