Maintenance Manual

ECE 492 - Spring 2016

Table of Contents

GLV

JGB Calibration

Calibration of inputs and outputs is done by adjusting the values in calibration.h and calibration.c in the source code and calling the get_calibrated_value method. With this method, the measurand must be provided and a calibration setting must be created and specified in the above classes; when called, this method returns the value, passed through the calibration scheme programmed in calibration.c.

JGB Software, Schematics, and Bill of Materials

The software, schematics, and bill of materials for this board are available on commit 6d8f7e11 via Git clone through:

> git@git.lafayette.edu:ece/ev-jgb.git
>
> Or
>
> https://git.lafayette.edu/ece/ev-jgb.git

Additionally, a copy can be found at:

> https://sites.lafayette.edu/ece492-sp16/files/2016/05/ev-jgb-master-6d8f7e117a3117a1c6c42e63b4ed99adb11533d1.zip

A provided README.md file in the software folder should prove helpful for programming, and before the initial programming, folders titled “bin” and “obj” must be created in the software folder. Programming was designed to be done with the Atmel-ICE programmer, although alternative programmers can be used, provided the makefile is altered accordingly.

The software used for the JGB relies on the following classes in the src folder:

- adc.c - enables, disables, and handles ADC and D2A conversions

- calibration.c - returns a 16-bit calibrated value given a 16-bit input and a calibration specification inside
- can_drv.c/can_frame.c/can_lib.c - handles CAN communication
- main.c - handles mode selection and general operation
- sensor_conf.c - initializes registers to known states and controls SSR and GPIO behavior
- timer.c - enables and disables the two onboard timers and the PWM signals associated with them
- config.h - defines constants used in the programming

Tractive System Controller

TSC Layout

The following pins are configured as such:

- ADC1 (Named ADC2 in code) = High voltage input (10-bit precision)
- ADC2 (Named ADC3 in code) = Physical throttle input (10-bit precision)
- ADC3 (Named ADC6 in code) = Motor controller voltage input (10-bit precision)
- ADC4 (Named ADC7 in code) = Mode selector input (MUST hook to GND for TSC operation)
- RELAY1 (Named SSR0 in code) = Safety loop fault (1 = open, 0 = closed)
- RELAY2 (Named SSR1 in code) = Physical throttle/drive mode light enabled (1 = closed, 0 = open)
- RELAY3 (Named SSR2 in code) = Precharge relay enable (1 = closed, 0 = open)
- GPIO0 = Drive button input
- GPIO1 = Buzzer output
- GPIO2 = AIR voltage present input
- D2A = Throttle output (10-bit precision, calibrated so a 0x0000 to 0xffff range corresponds to 0 to 4 volts)
- AMP0 = Current sensor input (10-bit precision)

TSC Operation

On startup or following a reset, the system checks the voltage present on ADC4; if 0V are present, it is set to TSC mode. In this mode, the onboard relays start closed and the throttle is disabled. Following the initialization, it reads in the measurands above and stores their values. Additionally, it checks if the throttle has been enabled and, if so, outputs the software or physical throttle value to the D2A (or 0 if the throttle is disabled) and, if the throttle is controlled by the driver's pedal, it enables the momentary buzzer and drive mode light. If the TSC receives a CAN remote transmit message addressed to 0x201, it transmits the relevant data from address 0x200. If the TSC receives a CAN data message, it reads it in, adjusts the throttle settings and

relays accordingly. In addition, it is designed to transmit a data message over CAN from 0x200 both in response to a received message and approximately every 1.048576 seconds. Also, if the throttle is enabled in software throttle mode and the board loses connection with SCADA for 1 second, the TSC ramps the throttle down to 0 and will not accept any throttle commands until the throttle reaches 0 and SCADA sends it a message telling it to disable throttle and set the software throttle to 0; this should take approximately 6.7 seconds from 100% and less time for smaller percentages. If there are no faults detected, there is voltage present on the AIRs line, the physical throttle is not pressed, the precharge voltage to the motor controller is present, and the drive button is pressed, the system will switch into physical throttle mode, as it would if SCADA were to tell it to do so.

TSI HIGH VOLTAGE

The TSI High Voltage section includes the pre-charge relay (24V AIR styled GIGAVAC relay), IMD (Insulation monitoring device), the ametes current sensor, (the forward reverse relay), and the high voltage isolation board.

System Description

HV connection pinout - This pinout describes the pinout of the cable from the high voltage TSI to the low voltage TSI.

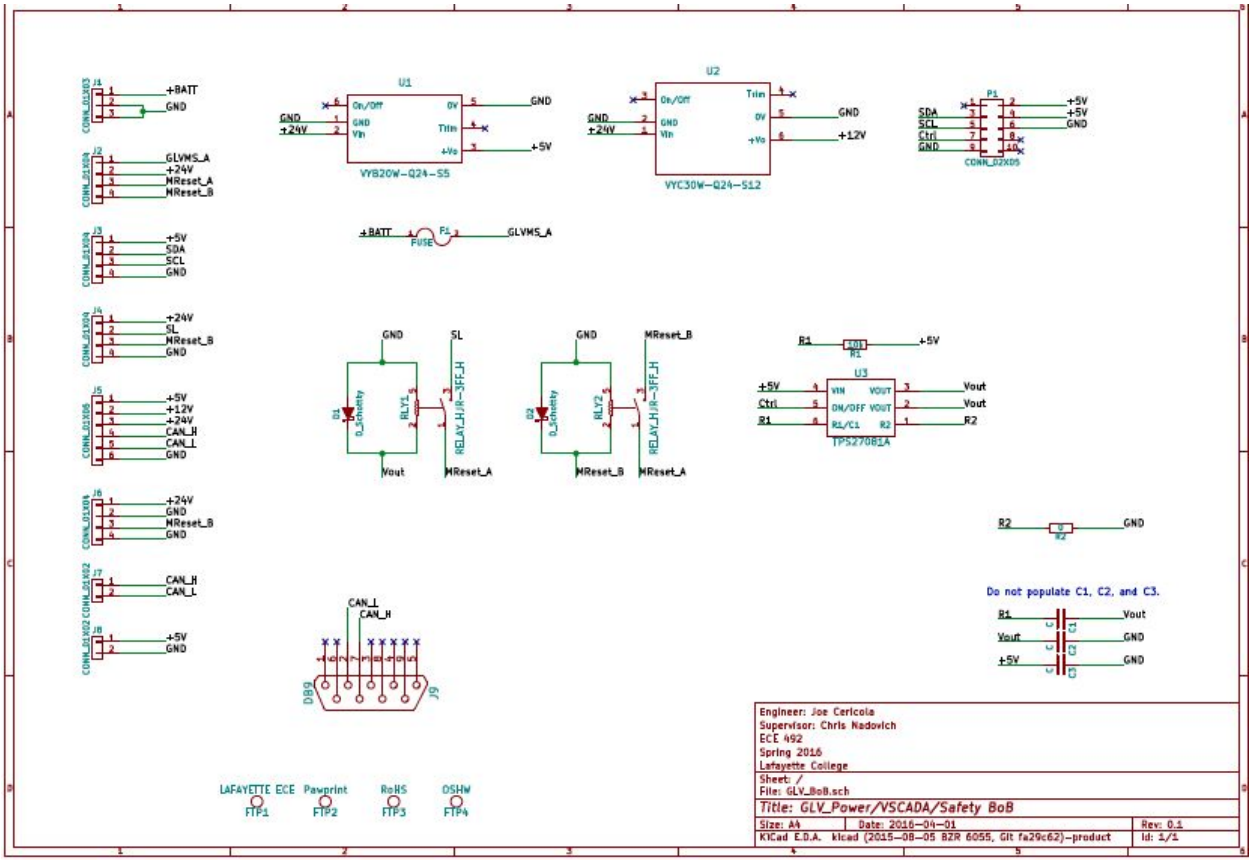
Pin1- +12V
Pin2- Current Sensor A1: This line of the current sensor goes to the AMPO+ and is measured through the diff amp internal to the JGB. (This is not hooked up to current sensor because it is no longer present on the TSI_HV section) (not calibrated)
Pin3- Output from IMD: The IMD_OK signal which is twelve or zero volts. (12 for good zero for ground fault) The IMD is not currently wired because of a ground fault issue with the motor controller. See QAR003d.
Pin4- Throttle Output: The throttle output from the JGB is wired to the TSI isolator board and then is output to the throttle input on the motor
Pin5- GND
Pin6- HV- : This is the minus side of the galvanically isolated 12V light which turns on at around 40volts which indicates that high voltage is present at the motor caps
Pin7- HV+: This is the plusside of the galvanically isolated 12V light which turns on at around 40volts which indicates that high voltage is present at the motor caps
Pin8- Output to precharge relay: This is the output to the precharge relay. ((No longer used no longer needed, see QAR003d)

Pin9- HV Voltage measurement: This is a voltage scaled from 0-140VDC to 0-5VDC which corresponds to the voltage at the caps of the motor controller see brandon's documentation on the JGB used for this application. (not calibrated)
Pin10- MC pre-charge output: This is the output from the optical isolator on the TSI_HV board this is routed back through the GLV section to determine the precharge state. (No longer used no longer needed, see QAR003d)
Pin11- Current Sensor A2: This line of the current sensor goes to the AMPO- and is measured through the diff amp internal to the JGB. (This is not hooked up to current sensor because it is no longer present on the TSI_HV section) (not calibrated)
Pin12- No connection

Schematic

To find the TSI High Voltage Isolation Board schematic go to the project website a can link can be found here.

http://sites.lafayette.edu/ece492-sp16/schematics/



To find the data sheet for the IMD you can go to this link.

https://www.bender-de.com/fileadmin/products/doc/IR155-32xx-V004_D00115_D_XXEN.pdf

To find the data sheet for the GIGAVAC relay go to this link.

http://www.gigavac.com/sites/default/files/files/catalog/spec_sheet/gx14_0.pdf

To find the wiring diagram for the Motor controller go here.

http://www.hpevs.com/Site/images/jpeg/Schematics/513-up/1234-1236-1238/update-10-27-15/auto1234-1236-1238_513%20up%20revE%20%2010-27-15.pdf

To find the data sheet for the necessary current sensor you can go here.

http://www.gmw.com/magnetic_sensors/ametes/documents/BBM-01Spec%20v3.pdf

Bill of Materials

There is a csv BOM in the TSI_High Voltage Zip.

Errata

See QAR003d

Huff Box DAQ/Dynamometer Controller

HB-DAQ Layout

The following pins are configured as such:

- ADC1 (Named ADC2 in code) = Torque sensor voltage input (10-bit precision)
- ADC2 (Named ADC3 in code) = Oil temperature sensor voltage input (10-bit precision)
- ADC4 (Named ADC7 in code) = Mode selector input (MUST hook to 5V for HB-DAQ operation)
- D2A = Load valve output (10-bit precision, calibrated so a 0x0000 to 0xffff range corresponds to 0 to 4 volts)

HB-DAQ Operation

On startup or following a reset, the system checks the voltage present on ADC4; if 5V are present, it is set to HB-DAQ mode. In this mode, the D2A is set to its maximum calibrated value. Following the initialization, it reads in the measurands above and stores their values. If the HB-DAQ receives a CAN remote transmit message addressed to 0x321, it transmits the relevant data from address 0x320. If the HB-DAQ receives a CAN data message, it reads it in, adjusts the load valve settings accordingly. In addition, it is designed to transmit a data message over CAN from 0x320 both in response to a received message and approximately every 1.048576 seconds.

GLV_BoB

The GLV_Power/VSCADA/Safety BoB (also referred to as the GLV_BoB for short) interfaces with the SCADA computer, routes major parts of the safety loop, and houses the 5V and 12V DC/DC converters. There is a 15A blade fuse to protect from potential sudden inrush currents from the 24V power supply. The board is currently in revision 0.1.

Schematic

Attached is a pdf of the KiCad schematic. The pdf is also available on the project website at the following address:

https://sites.lafayette.edu/ece492-sp16/files/2016/02/GLV_BoB_Schematic.pdf

Additionally, the raw sch file from KiCad is available on the GitLab repository at the following address:

https://git.lafayette.edu/ece/ev-glv-bob/blob/cb24e3d18a72272dd03da7061ad991c434417d5a/GLV_BoB.sch

Bill of Materials

A csv BOM generated from KiCad is available on the project website at the following address:

https://sites.lafayette.edu/ece492-sp16/files/2016/05/GLV_BoB_BOM.csv

Additionally, the same csv BOM is available on the GitLab repository at the following address:

https://git.lafayette.edu/ece/ev-glv-bob/blob/cb24e3d18a72272dd03da7061ad991c434417d5a/GLV_BoB_BOM.csv

Errata

The iteration of the GLV_BoB in use in the project during the Spring 2016 semester has a few minor issues which have already been addressed in the aforementioned documents. To better serve future users of the board, these will be briefly listed and explained.

VSCADA

Hardware

The main monitoring system will be a javascript applet similar to the one below. The applet will be available to any browser capable device connected to the same network as the Raspberry Pi. The dashboard will also be a javascript applet that you can navigate to on the 7" dashboard touch screen. The dashboard touchscreen's computer will be the SCADA Raspberry Pi 3. In drive mode the dashboard should not have any detailed diagnostics according to the formula EV so a warning light comes on when a problem is detected. If you want to know what the warning was check the web page or the SCADA logs. In maintenance mode you may use the touchscreen to navigate the diagnostic website.

Software

HTTP/WS Server

The HTTP/WS server is loaded and, by default, serves a minimalistic web application which dumps the entire system state. A configuration file can be used to serve files from a different directory, allowing a better user interface to be easily loaded. In the event that the system model cannot be created because of an invalid topology, this component is responsible for generating an error page. The state of the system model is transmitted to clients over WebSockets as JSON objects. For implementation details see "SCADA WebSockets API"

Document Logger

This component monitors the system model for changes in state--represented as dictionaries--and logs the changes as documents in a NoSQL database. The concept of a non-relational database can be frightening to some, but we believe this allows for simplicity of implementation and better normalization of data in our use case.

Message Bus

Although not mentioned above, the components communicate through a message bus. Any system can broadcast any kind of message to the other systems. This provides an elegant method of calling multiple handlers at once, and synchronizing the multiple concurrent entities. The broadcaster doesn't have to be aware of who is interested in the message, and listeners don't need to be concerned with where a message came from. In our case, the data flow is mostly unidirectional from the models to the logger and web api.

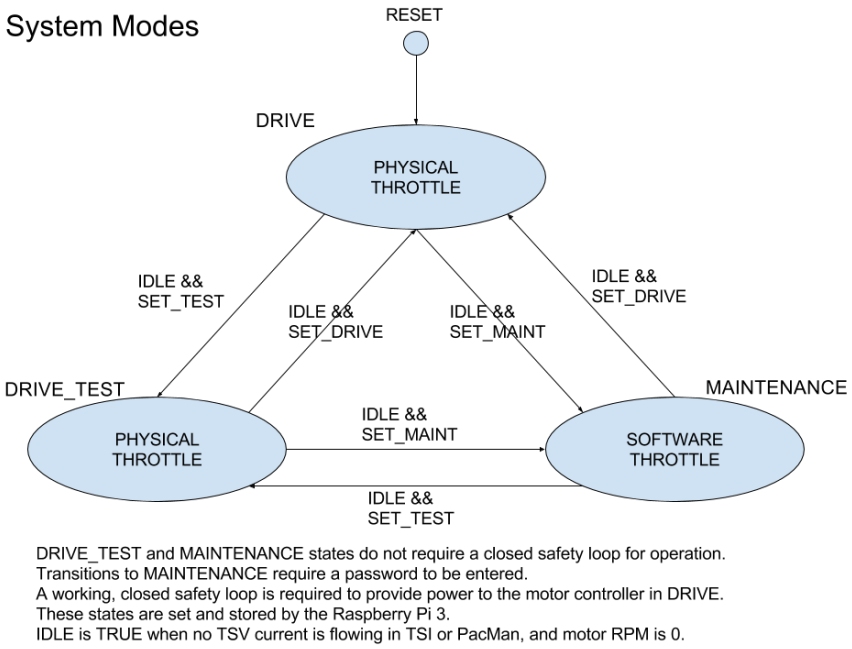
System Model

The system model is created on startup by looking at the `system_topology.yml` file. Details of the format of this file can be found in “SCADAd System Topology.” The system is divided into two classes of subsystem: ‘Virtual’ and ‘physical’. Physical subsystems have a real world counterpart and attempt to mimic the state of the counterpart. They are also associated with logic (methods) to control the state of the real world system. Virtual subsystems are where the system gets all it’s uniquely specific functionality. There is nothing saying a battery pack has to be part of a car, or that a car has to have a safety loop, until a virtual subsystem is written to manage that. Several virtual subsystems are proposed by this team. The subsystems are described below.

- **Battery Manager -** Aggregates all physical subsystems of type `BatteryPack` and broadcasts messages to all of them to coordinate global state.
- **Mode Manager -** This subsystem knows that the vehicle needs a power source. A BatteryManager or power supply could satisfy that. It knows that a software throttle should be used in maintenance mode, and the physical throttle used in drive mode. It knows that a safety loop should be present. Et cetera.

The Event Loop

A system like this has a lot of things happening at once. In order to make it easy to express concurrency, the software uses Python3.5's asyncio library. Subsystems typically have a loop which waits for data from CAN, and then processes it. These loops are expressed as asyncio coroutines. They are cooperatively scheduled, which means that the currently executing routine will only change when it is explicitly allowed to. This makes the software more easy to reason about than if it used a threaded approach, which would mean these routines are preemptively scheduled. Python's GIL prevents any true parallelism, so there is no downside from not using true threads. Some blocking libraries may need to be executed in a separate thread, but those can be managed by the event loop.



Other Information on Maintainability

The system topology is specified in a YAML configuration file. The file maps to a hash table and supports lists, numbers, booleans, strings, and comments, which make the configuration file self documenting. There will be no auto detecting hardware because then there would be no way of knowing if the system is completely online.

If the syntax of the configuration file is incorrect, the daemon will serve an HTTP 500 error page, to indicate that the system model cannot be created. If the user neglects to describe parts of the system in the configuration then the model will also be lacking. If the user describes subsystems that do not exist then the model will contain them, but they will identify as “offline” until contact is made. The configuration file can be modified by connecting to the machine running the daemon over SSH and editing the file with a text editor (nano, vim, etc.)

To clarify the consequences of misconfiguration: Incorrect syntax will result in an error page and no functionality. A subset of the real system will result in an online, but incomplete system, with full SCADA support for the subsystems described. A superset of the real system will result in offline components and full SCADA support.

Software will run on the heavily supported Ubuntu 15, be coded in python, and managed by system.

TSV Accumulator

Calibration

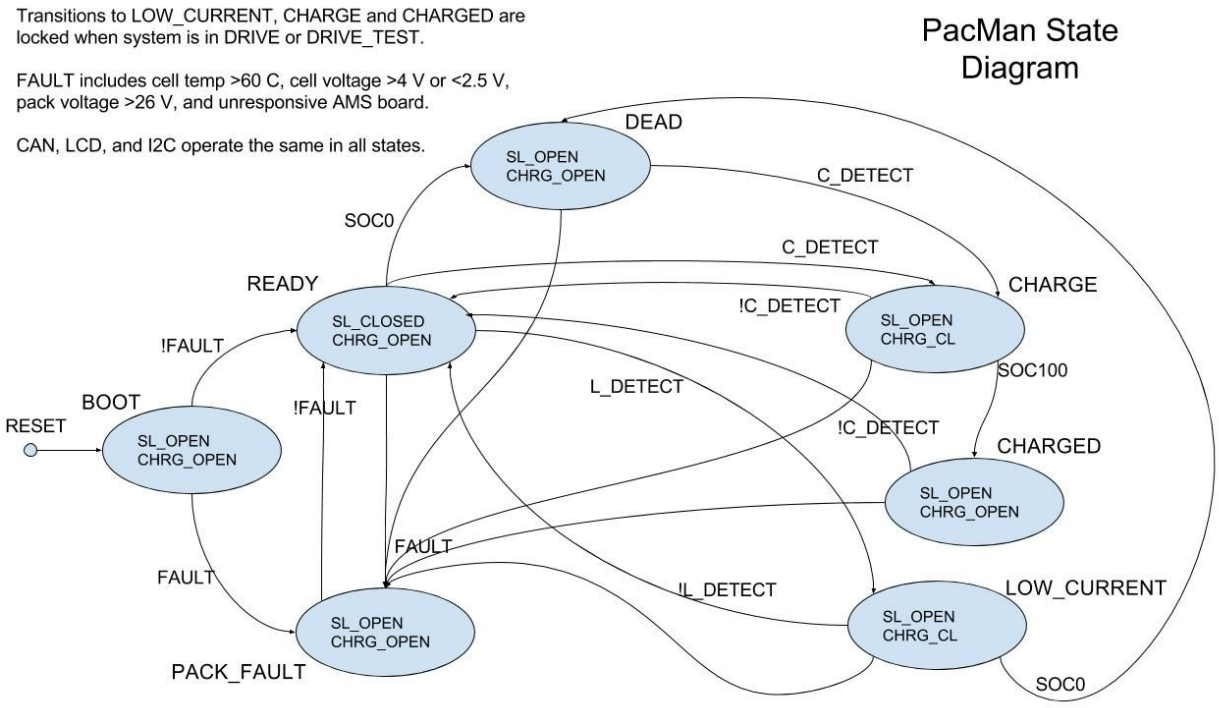
Calibration of sensors (pack and AMS sensors) is done by adjusting the values in params.h in source code.

Slope and offset values are used in all cases. Sensors should be calibrated in the test stand with lab power supplies providing power. To calibrate a sensor, its slope cal factor should be set to 1.0 and offset to 0.0. Output on the LCD screen will then be the raw sensor value. Data points at the high and low end of a sensor should be taken and compared with values measured with a calibrated device to calculate slopes and offsets.

Data collected for calibration (and the calculations of cal factors) can be found in xlsx format at the following address:

https://sites.lafayette.edu/ece492-sp16/files/2016/05/CalData.xlsx

State Transition Diagram



PacMan Software

The current version of PacMan software is v 0.14 and source code is available at the following address:

https://sites.lafayette.edu/ece492-sp16/files/2016/05/pacman_software_v0_14.zip

The tool chain is unchanged from previous versions of PacMan and details about it are available at the following address:

http://sites.lafayette.edu/ece492-sp15/files/2015/12/PACMAN_Programming_manual.pdf

The software is built on the Atom Threads RTOS. Atmel TWI and CAN libraries are utilized to achieve communication. All configurations (I2C addresses, CAN addresses, calibration factors) are stored in params.h

The code in main.c sets up tasks listed in tasklist.c and starts the RTOS. Functions that generate LCD screens are detailed in lcd.c. Functions that utilize TWI libraries to perform I2C communication are detailed in i2c.c. The remaining c files detail tasks that run continuously:

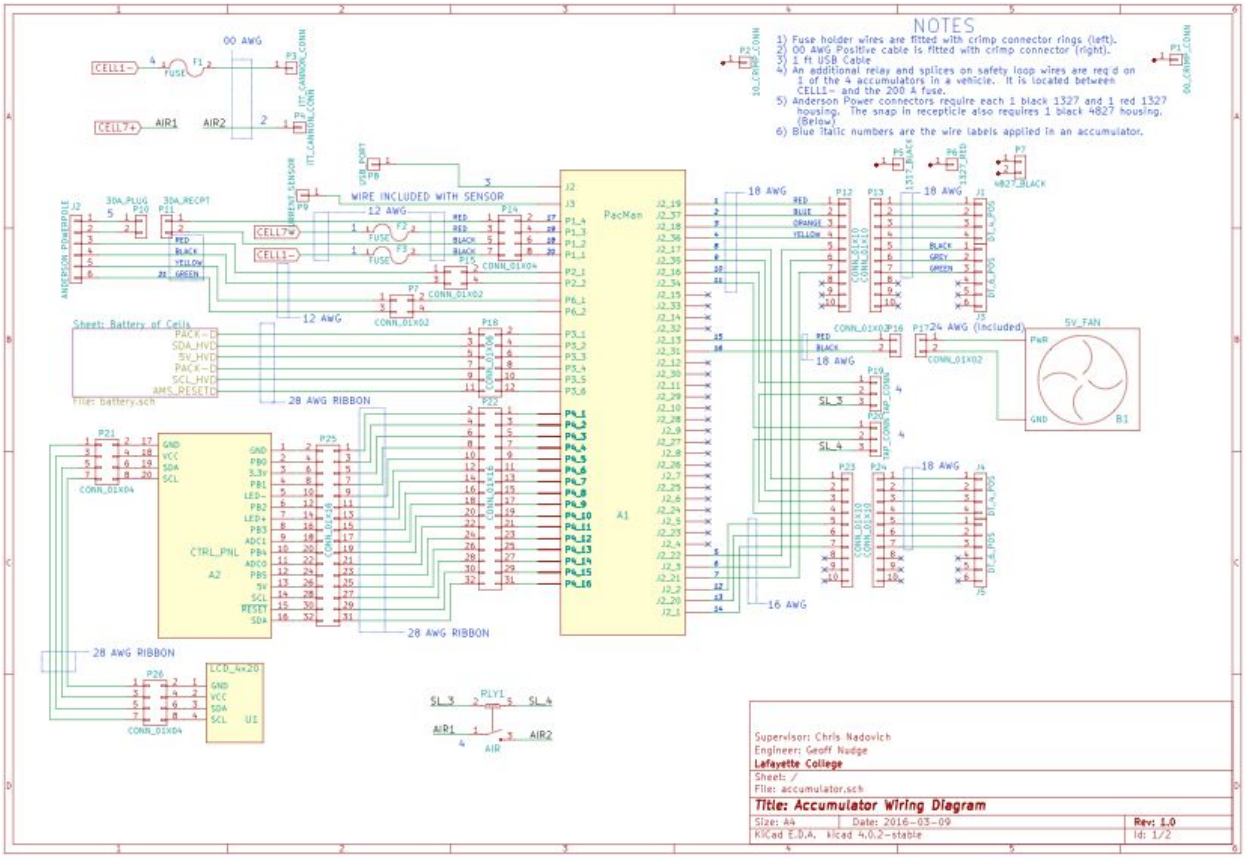
- task_button.c - retrieves button presses on the control panel
- task_can.c - transmits CAN messages
- task_charge.c - performs integration of current and calculates state of charge
- task_config.c - performs state transitions
- task_ gui.c - calls function in lcd.c to set the display output
- task_hearbeat.c - blinks an LED on PacMan to indicate the computer is operating
- task_i2c.c - calls functions to perform I2C communication tasks
- task_safety.c - opens and closes the safety loop relay
- task_watchdog.c - resets off chip watchdog

Schematics

PacMan

Attached are schematics generated from KiCad. The KiCad project is available at the following address:

https://sites.lafayette.edu/ece492-sp16/files/2016/05/pacman_hardware_rev_0_5.zip



Accumulator

Attached are schematics generated from KiCad. The KiCad project is available at the following address:

https://sites.lafayette.edu/ece492-sp16/files/2016/05/accumulator.zip

Bill of Materials

PacMan

A csv BOM generated from KiCad is available at the following address:

https://sites.lafayette.edu/ece492-sp16/files/2016/05/pacman-main.csv

Accumulator

A csv BOM generated from KiCad is available at the following address:

https://sites.lafayette.edu/ece492-sp16/files/2016/05/accumulator.csv

Dyno

Calibration

Refer to the torque calibration document. This lists the entire procedure involved with calibrating the torque sensor.

Maintenance

Water Cooling System Maintenance:
In its current state the water cooling system is not maintainable. The water reservoir can not be emptied without removing the entire motor controller stand from the dynamometer. This issue must be addressed next year. A possible solution to consider is to move the main water basin off the dynamometer. This will let the water be changed more easily as well as air would be able to escape the system.

Dynamometer Oil Maintenance:
The oil in the dynamometer does not need to be changed or replaced unless there is a spill. In the case of a spill there is oil absorbent located in AEC 402. After consulting an advisor proceed to apply the adsorbent followed by a clean up of the room when the oil has been absorbed. Once the new oil has arrived to replace the oil you must get the oil pump from the mechanical engineering workshop. They will instruct you on how to use this to refill the dynamometer with oil.