EE459 - Group 17

Christian Borao, Tyler Holloway
Hannah Lau, Taylor Stroobosscher

Spring 2019

Recreational Vehicle Monitoring and Real-Time Data Analysis

Abstract

Our project involves reading the standardized diagnostic information broad casted by all vehicles homologated for sale in the US after 2007, capturing live positional and geographical data, and presenting the aggregated data as a performance branded vehicle enhancement. This system will effectively give an end user access to professional grade vehicle performance monitoring equipment in a single, complete package.

Purpose

Our intention is to record readily available diagnostic information combined with sensory data such as location and acceleration to a vehicle enthusiast and recreational racing participant for monitoring and analysis purposes. With this system, users will be able to identify driving characteristics that can produce real-time and logged data that can then be carefully analyzed to optimize for any driving condition. With sensory data this generic, it would also be able to monitor driving conditions that can lead to safety hazards, premature vehicle and engine wear, and provide immediate and specific maintenance notifications (ex. your check engine light goes on, what caused it?).

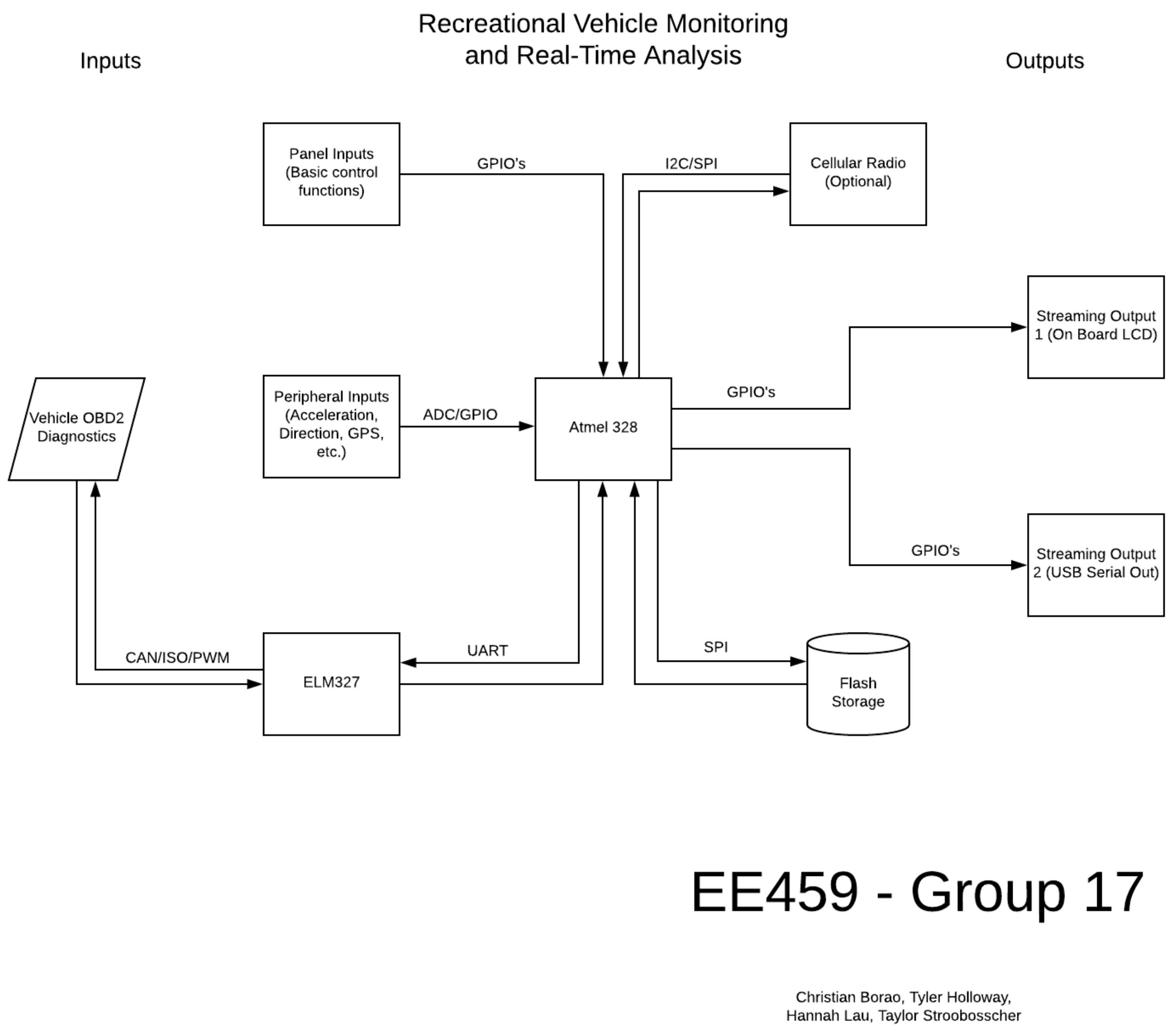
Usage

This system will function as a standalone unit that interfaces with the vehicle's 16 pin OBD2 port underneath the dash, and can then be mounted to the center console where a display and button panel can provide controls and streaming output to the user. Since the OBD2 standard provides 12V DC power, our only connection once inside the vehicle will be to the same diagnostic port. The user can configure and select data streams and choose when to enable logging for later analysis. The log files can then be offloaded and then parsed with custom software.

Major Pieces

Firstly Our system is modularized around the ATMEGA328 processor. The processor will handle all input, asynchronous and synchronous, as well as output streaming to any peripheral displays, serial ports, or memory units. All vehicle communication is facilitated with the diagnostic module, employing the ELM327L. This chip provides multiple interfacing options for vehicle analysis and can read a wide range of makes, models, and years of vehicles. The diagnostic module has a built in UART interface and will meet the ATMEGA328's UART immediately. No logic translation is needed since the ELM327 can be configured to run 5V logic. Next, any control modules such as buttons or controls can be routed to GPIO's on the main processor. Any peripheral sensory units have the option of interfacing with the processor though digital logic, or we can use one of the on-board ADC's that comes with the 328. On the output side, the processed data will mainly be sent synchronously to the various memory or streaming devices. SD Flash uses its own co-processor and data is typically exchanged over an SPI connection although some logic translation may be necessary. A board-mounted LCD will provide control interfacing and data streaming that can interface with the 328 using an array of GPIO's. Lastly, a generic serial communication output can provide us the option to mount our device to an external machine and provide a high level of abstraction for a more complex software implementation of real-time data analysis and presentation (ex. sending our data stream to an android tablet that can display a much more complex array of data and control abilities).

Block Diagram


Recreational Vehicle Monitoring and Real-Time Analysis
Inputs
Outputs
Panel Inputs (Basic control functions)
GPIO's
I2C/SPI
Cellular Radio (Optional)
Streaming Output 1 (On Board LCD)
GPIO's
Vehicle OBD2 Diagnostics
Peripheral Inputs (Acceleration, Direction, GPS, etc.)
ADC/GPIO
Atmel 328
GPIO's
Streaming Output 2 (USB Serial Out)
CAN/ISO/PWM
UART
SPI
ELM327
Flash Storage
EE459 - Group 17
Christian Borao, Tyler Holloway, Hannah Lau, Taylor Stroobosscher

Time Line

By the end of February, Our diagnostic module should be done and we should have a functioning data output (whether its over a serial data line or output to an LCD). Even if we don't currently have the parts to put down on the perf-board, we currently have a functioning development kit for the ELM327 and Arduinos, so we can begin developing the software while we wait for parts.

Also by this time, we should have the rest of the modules designed and have a parts list finalized. At this point, the goal is to have a minimal end-to-end prototype of our idea.

The month of March will be dedicated to expanding on our prototype by adding extra inputs (ex. sensors) and outputs (ex Flash memory). If we find ourselves meeting our design goals quickly, we can also look into developing a cellular data interface and look into pivoting our idea as an IoT product. This would open up another level of engineering that would certainly exhaust the rest of our extra development time.

Then April can be spent debugging, optimizing, and testing the product. More than likely we will have to do some aggressive optimization to get all of our asynchronous, task-oriented code to run smoothly on the 328. Expecting 2 weeks of code review and compression seems realistic. Then we can spend the rest of the time testing and evaluating our design, and preparing for the final presentation.

Major Tasks

- Order parts for our minimum prototype idea.
    - ELM resources, as described in the datasheet
    - LCD, we have experience with these guys from 109
    - Input controls (push buttons or any other type of control we can interface with)
- Write the routines to initialize and handle data transfers from the 327, essentially finalize the software to communicate between the 328 and the 327
    - AT commands and expected returns, This will effectively mean coding up the datasheet for the 327's software interface.
    - parsing data streams and sending that data to a stack buffer
- Order parts for the major sub-modules
    - GPS
    - Flash
    - Accelerometer
- Write an LCD driver, decide which pins to dedicate to it
- Write a flash driver that goes one step above just the SPI interface and actually interfaces with the part's filesystem
- Write an SPI driver for the main processor.
    - Flash will probably need it
    - Any extra communication device (generic serial driver?)
- Write a flash driver that goes one step above just the SPI interface and actually interfaces with the part's filesystem
- Write interrupt handlers to take care of the asynchronous operations.
    - RX complete
    - ADC complete (if we are using analog sensors)
- Write engine error code handlers that can process generic OBD2 Faults (this is going to be a ton of data)
- Mount all of our designed modules to the perfboard and debug
- Design a mount for a vehicle center control console (cupholders, vents, etc)
- Go to the track and test!
- If we have time, some python code that can parse and display our data graphically would be pretty cool