PROJECT **INDUZIONE**

TEAM ALWAYS BLOCK
Long Beach, CA

TABLE OF

CONTENTS

TEAM ALWAYS BLOCK

A Letter from the Team:

With the class now over, we hope to continue to make Project Induzione something real.

We would like to thank Bob Ward and CECS 490B for bringing us all together and helping us form this idea.

We have been excited to work on this project and give it the amount of attention to detail and excellence it needs for it to take wings.

Love,

TEAM ALWAYS BLOCK
hello@teamalwaysblock.com

(This page intentionally left blank.)

PROJECT SUMMARY

An overview of our project.

(This page intentionally left blank.)

// SUMMARY

Project Induzione is simple:

01. Plug our peripheral into your car's OBD-II port (every car since 1996 has one).

02. Drive.

03. Connect to our smartphone application and have fun!

ESSENTIALLY :

Our product consists of (1) a "plug-and-forget" component that you plug into your car that logs information about your car (including error codes and diagnostics) and (2) a smartphone application where you can access this information.

We seek to create the missing bridge of interactivity between you and your car.

// SUMMARY

Our team ultimately shares a broad vision based upon expanding your use of your car's data beyond viewing data-logging / diagnostic information.

// WE IMAGINE:

01. Having a social hub where you can compare your stats and compete with your friends based upon those stats

02. 'Leveling-up' based upon your driving habits and having the ability to both add virtual upgrades to the car you drive and race your friends virtually based upon those stats

03. Compete in weekly challenges with other users. (i.e., drive 200 miles this week or check all of the vitals for your car within the application).

This document serves as an introduction to Project Induzione, the technology behind it, and the features that make it a compelling product.

PREVIOUS WORK

Work already completed towards our project before our class' endeavor.

(This page intentionally left blank.)

// PREVIOUS WORK - OUR STORY

NEW BEGINNINGS

Our project was defined by us as a team, and there were was no previous work done associated with our project. A fresh start.

(This page intentionally left blank.)

AVAILABLE TECHNOLOGY

Existing solutions similar in scope and functionality to our project.

(This page intentionally left blank.)

// AVAILABLE TECHNOLOGY

There are several products existing today that are similar to our product. An overview of these products is listed in the table below.

TORQUE	ANDROID

Product Description:

Torque is an application that allows users to view information about their car via an OBD-II peripheral that connects to the vehicle and links to the user's phone via BlueTooth. In Torque, you are presented car information such as MPG, coolant values, and are alerted of fault codes.

WAZE	ANDROID/IOS

Product Description:

Waze is a traffic and navigation application that is community-based. Essentially, users collaborate to provide traffic information on highways across the country. As you drive more, you collect pellets using the phone's GPS that increase your running score within the application. Users are able to provide alerts to other users based upon current traffic conditions.

MAVIA	IOS

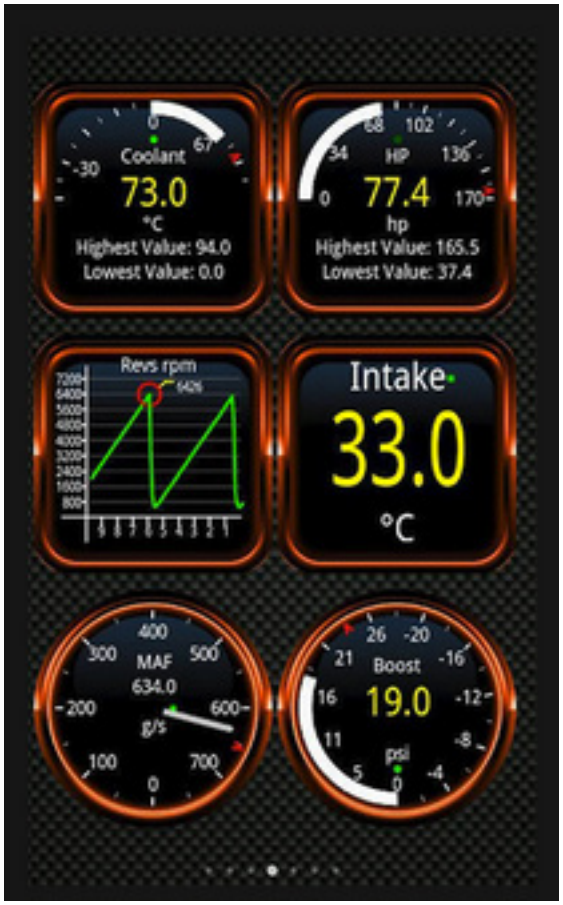
Product Description:

Mavia is a car-connection tool that uses the OBD-II port to provide a driver information about their vehicle. It provides deep analytics in a user-friendly manner, and uses social aspects to enhance the driver's connection to their surroundings.

// AVAILABLE TECHNOLOGY

TORQUE

An application designed to allow users to view information about their car via an OBD-II peripheral that connects to the vehicle and links to the user's phone via BlueTooth.



// HOW IT IS SIMILAR

Torque uses a BlueTooth peripheral to connect the user's car (via OBD-II) to their phone. This connection is the same kind we will be using for our product. Also, Torque displays information about your with gauges and skeuomorphisms that make the app look like the kind of presentation you would see on your dashboard.

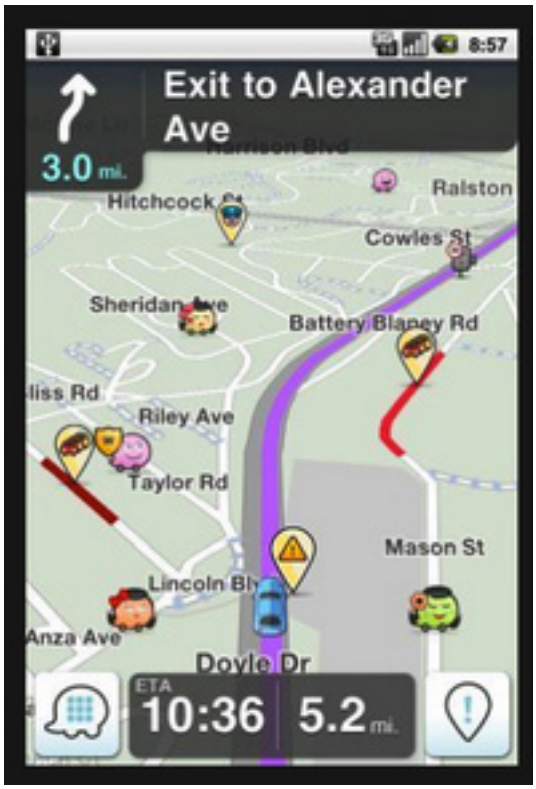
// HOW IT DIFFERS

This application gathers all of the necessary information about your car, but does not present it in a way that is by nature social or gamified. You do not get rewarded from the values read on your car, and although you can post information to Facebook or Twitter, you are not connected to your friends who are also using the app.

// AVAILABLE TECHNOLOGY

WAZE

A traffic and navigation application where users collaborate to provide traffic information on highways across the country.



// HOW IT IS SIMILAR

Waze is a traffic and navigation application that is community-based. Essentially, users collaborate to provide traffic information on highways across the country. As you drive more, you collect pellets using the phone's GPS that increase your running score within the application. Users are able to provide alerts to other users based upon current traffic conditions.

// HOW IT DIFFERS

Waze is focused on traffic and it's user-base mainly uses the application for traffic information. Also, Waze does not utilize the car's OBD-II port. While our app focuses on connecting the user with their car in a unique way and letting them 'level-up' and feel connected to their car, Waze is about giving users useful traffic information for their daily drive.

// AVAILABLE TECHNOLOGY

MAVIA

A car-connection tool that uses the OBD-II port to provide a driver information about their vehicle. It provides deep analytics in a user friendly manner, and uses social aspects to enhance the driver's connection to their surroundings.

// HOW IT IS SIMILAR

Mavia is our biggest competitor, and is the product that is most similar to what we are creating. It provides analytics to users in a friendly format with a well-designed iOS UI/UX. It encourages users to be social by connecting to outside social networks such as Facebook, Twiter, and Foursquare.

// HOW IT DIFFERS

Mavia does not present a gamified experience to users. Also, Mavia does not connect users socially directly through the app: it instead directs users to post on an outside network. Unlike our application, Mavia focuses on analytics, geofencing, locating, and reminding, but does not have any of the gamified aspects that our product does. It also does not provide analytics as deep as we intend to (such as average/real-time RPM, tire pressure, etc.)

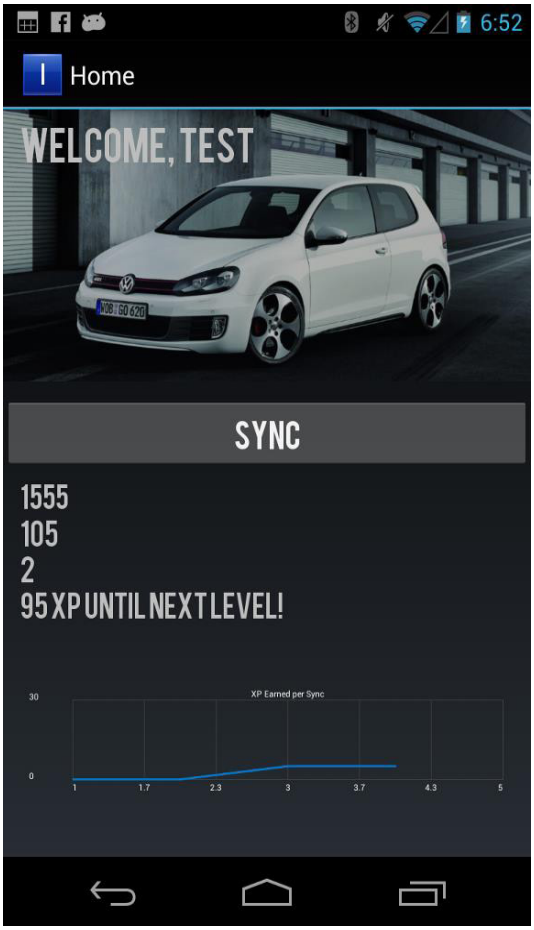
PROJECT DESCRIPTION

A full description of our project.

(This page intentionally left blank.)

// THE APPLICATION

From the Android application our team developed you can see your car's logged data. The application has a tabular interface displaying data-logging and car information.



// APPLICATION DETAILS

The Android application has a tabular interface displaying:

- Home (shows info on your car / weekly challenges)

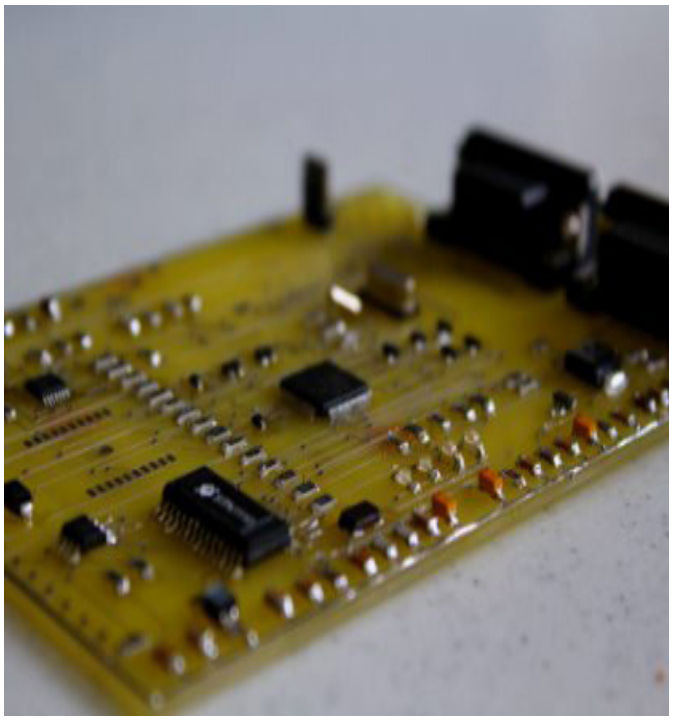
- Social Hub (car club and competitive, connect with friends and family)

- Live data-logging / synced data displaying graphical gauges (analog / digital)

- Diagnostics (diagnostic trouble codes, reminders for oil changes, tire pressure)

// THE PERIPHERAL

The hardware peripheral that plugs into the car houses a custom, team-designed printed circuit board (PCB). The idea behind this peripheral is that it will be plugged into the OBD-II port of car and left alone to gather data.



// LOGGING METRICS

The peripheral will be able to logs the following metrics:

- Fuel consumption
- Fuel used or remaining
- Fuel wasted while idling
- Distance
- Location
- Engine RPM
- Vehicle speed
- Engine load
- Air intake temperature
- Throttle position
- Battery voltage
- Error codes

PROJECT SPECIFICATIONS

A list of specifications our project was designed to meet.

(This page intentionally left blank.)

// SPECIFICATIONS

Specifications for this project were chosen to provide the best user experience possible. We found it crucial to focus on designing a product that we ourselves would want to use, and our product specifications reflect that.

PRODUCT SPECIFICATIONS

Since our product is divided into two pieces (peripheral and application), the specifications we have created are divided into two parts. The specifications of our product are described below. Major project requirements are designated with two stars (**).

PROJECT INDUZIONE PERIPHERAL

- Product Size (WxL): 1.8" x 3" compact design **
- Power Source: Requires 12V - 16V from OBD-II port **
- Built in GPS
- Built in BlueTooth **

MOBILE APPLICATION

- Android application required to communicate with the peripheral **
- Free Android application available on Google Play marketplace **
- Sync data stored in the peripheral to the application via BlueTooth **
- Tabular view showing interactive sections of application
    - Home (shows info on your car / weekly challenges) **
    - Social Hub (car club and competitive, connect with friends and family
    - Live data-logging / synced data displayed (graphical gauges) **
    - Diagnostics (show diagnostic trouble codes, reminders for oil changes, check tire pressure)

// SPECIFICATIONS

DID WE MEET OUR SPECIFICATIONS?

Project Induzione is an ambitious project. Our group has worked diligently in meeting each specification we assigned for ourselves, however, we did have some issues that we're the result of us being a small-shop with limited time. The mobile application we were able to meet all of our specifications. We had some trouble with the PCB, however, detailed in the section below.

PCB

We constructed two PCBs, however, we were unable to program either of them! We were unable to resolve this issue. We do know that all of the peripherals (GPS, BlueTooth) on our board function correctly. The specifications we met as a team are shown below:

PROJECT INDUZIONE PERIPHERAL

- Power Source: Requires 12V - 16V from OBD-II port
- Built in GPS
- Built in BlueTooth

MOBILE APPLICATION

- Android application required to communicate with the peripheral
- Sync data stored in the peripheral to the application via BlueTooth
- Tabular view showing interactive sections of application
    - Home (shows info on your car / weekly challenges)
    - Social Hub (car club and competitive, connect with friends and family
    - Live data-logging / synced data displayed
    - Diagnostics (show diagnostic trouble codes, reminders for oil changes, check tire pressure)

TASK LISTING

Tasks necessary for the project.

(This page intentionally left blank.)

// TASK LISTING

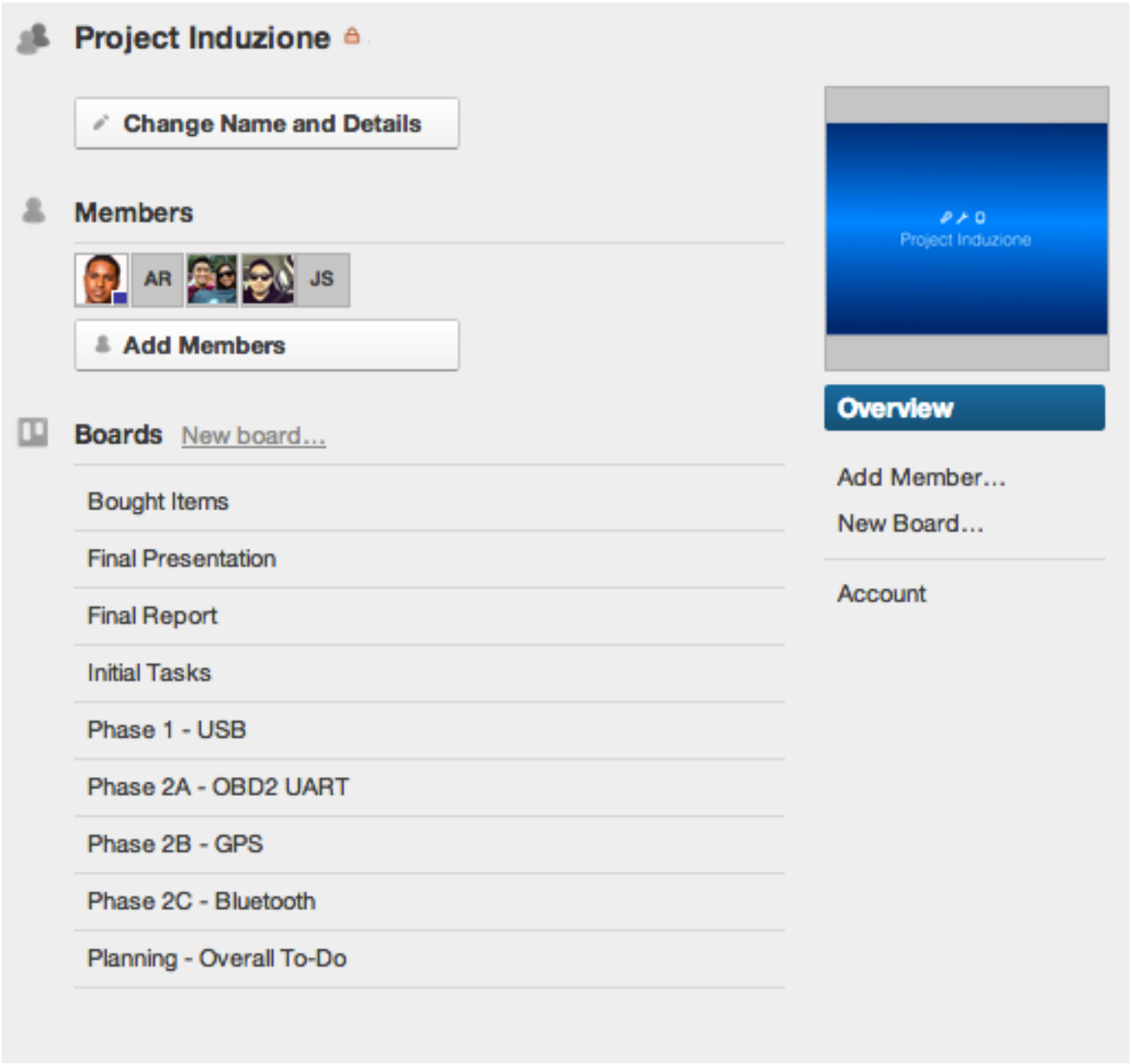
Hardware and software tasks required for our project, when broken down, can be seen as the following:

01. Purchase all required modules / hardware components
02. GPS data-logging
03. GPS data parsing and storing to SD card
04. OBD-II data logging
05. OBD-II data parsing and storing to SD card
06. BlueTooth UART communication
07. Purchase surface mount version of the modules / hardware components
08. Create board and schematic using Eagle CADsoft
09. Create gerber file and begin prototyping
10. Solder board and test
11. Plastic case enclosure research
12. Order plastic case enclosure
13. Application phase: determine development environment
14. Application phase: wireframing/screens/UX-UI flow
15. Application phase: begin prototyping application, divide application into segments
16. Application phase: bring and store data locally into an Android phone via BlueTooth
17. Application phase: server-side implementation
18. Complete application

// TASK LISTING

TRELLO TASK MANAGEMENT

Our team used a task management system called Trello. Trello allows teams to create boards with tasks and assign them to individual members or the group as a whole. It greatly helps in helping the team stay on schedule, as each team member can see the progress of another. A screenshot of Trello is provided below.



POTENTIAL CUSTOMERS

A brief overview of the people our project would appeal to.

(This page intentionally left blank.)

// POTENTIAL CUSTOMERS

Our product can be a little confusing - we understand. The following sections contain use-case scenarios from several Project Induzione users, who each use Project Induzione for different purposes.

MEET JODY

Jody buys Project Induzione for her new Fiat 500c Convertible.

Jody isn't all that concerned with the game aspects that Project Induzione provides, but she is interested in knowing the metrics of her car. Since she has a new car, she wants to make sure that everything is constantly A-Ok and looking normal before she would ever get a check engine light. If she ever does get an error code, she is comfortable in the fact that the Project Induzione component will diagnost the error code her car is throwing and record it so that is better able to tell the dealership what the problem is.

Jody has logs of her coolant temperature, fuel gauges, tire pressure, and other useful metrics in the palm of her hands at all times. She's an avid Project Induzione user for the ease of monitoring it presents her.

// POTENTIAL CUSTOMERS

MEET PHILLIP

Phillip commutes from Los Angeles to Corona daily. That's a lot of driving!

Phillip buys the Project Induzione physical peripheral, plugs it into his OBD-II port in his car, and downloads the Project induzione smartphone application.

Phillip is now connected to the nation-wide community of Project Induzione users. Project Induzione recognizes the car he's driving (a 2010 Nissan Altima) and assigns it a base performance index (PI) of 56.

He goes to race another driver virtually (who drives a 2003 Honda Acoord), and loses badly! He needs to improve his stats.

Phillip gains 400 experience points every weekday for his round-trip commute of 80 miles. Also for this week, one of the Project Induzione weekly achievements is to get 35 MPG for the week. Phillip gets this achievement and is awarded an extra 500 experience points. Sweet!

Phillip quickly levels up his car, buys virtual upgrades to his car with the virtual points he's earned (he chooses to get a new transmission) to improve his PI, and once again challenges the person who had the Honda Accord who burned him earlier.

He wins! Phillip is engaged and now wants to drive more, hit landmarks, get more achievements, and win more races, all while having valuable information about his car readily available! W00T!

// POTENTIAL CUSTOMERS

MEET JESSIE

He is your automotive enthusiast.

He's the guy that checks his oil and tire pressure every week and does not put anything less than 91 octane in his 2000 BMW 325i. He knows a little about engine management and is a good wrench. He can work his way around a car and diagnose what's wrong with his and his friend's vehicles.

Jessie buys the Project Induzione peripheral and installed the Android application to make it easier for on him to troubleshooe the many problems he is now encountering with his high mileage car.

He opens the live data-logging feature and not only finds that one of his cylinders is having a misfire and triggering a check engine light, he also discovers that his oil temperature is higher than usual. He replaces his spark plugs, cleans his mass flow sensor, and finds an oil leak. These small fixes help him keep his car on the road, and not broken down on the side of the highway.

// POTENTIAL CUSTOMERS

MEET RONALD

Ronald just bought a used 2001 Honda Civic for his son. a new driver.

Ronald buys the Project Induzione peripheral to keep track of his son's driving habits. He enables Project Induzione's blackbox logging feature to see if his son is ever speeding or driving recklessly. He also keeps track of his son's car for him, and takes care of any maintenance that the car needs when he notices something is wrong with the car.

Ronald handles the car vitals, while his son uses Project Induzione for the gaming aspect of it, leveling up his car as he drives and challenging other users.

COMPONENTS

The nuts and bolts of our project.

(This page intentionally left blank.)

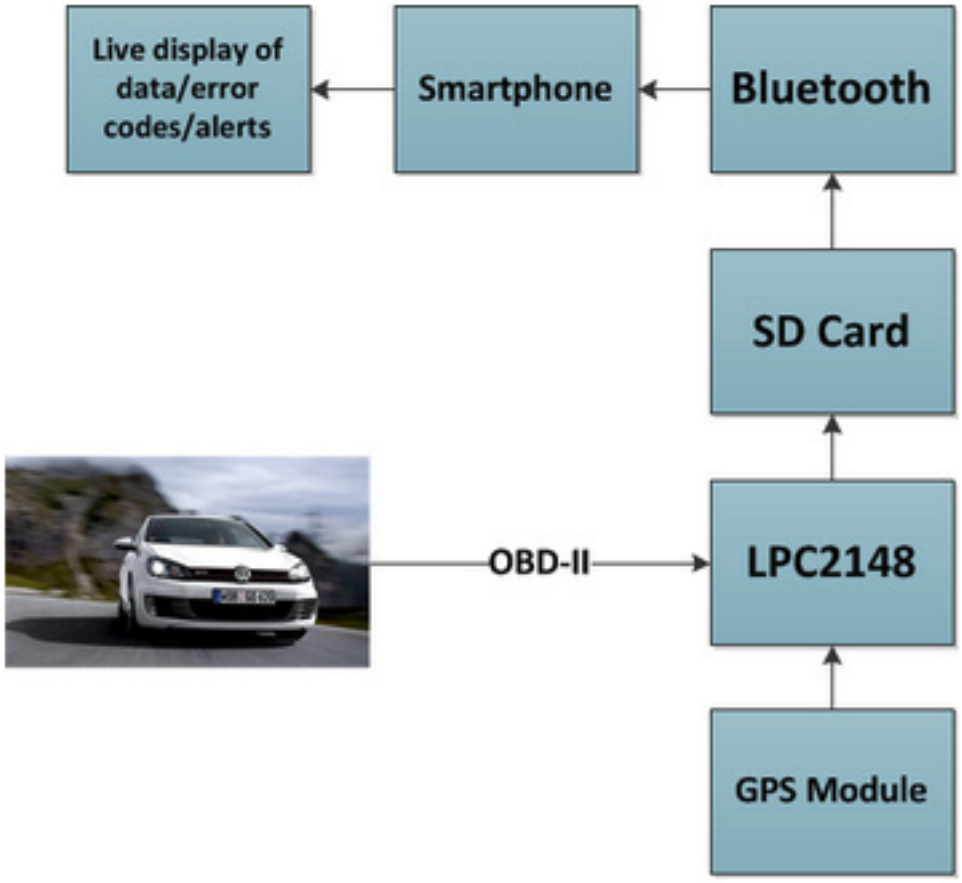
// COMPONENTS

Many drivers today are left at the mercy of car dealerships and/or questionable auto mechanics. Most, if not all people have no idea what exactly is going on under the hood of their car. For example, if a check engine light goes on, the driver typically does not know the reason why. With the features of our tool, users are able to diagnose any problems that arise with their vehicle, allowing them to know exactly what needs to be done when approaching a mechanic.

DESIGN CHART

In order to create this solution, our product implements many pieces of technology. There are various modules and peripherals that are required for successful completion of our project. A top-level block overview of the prototype of our project is provided below.

Note: The diagram below is strictly for prototyping purposes. All components of our project are finalized on a PCB designed by our team.



// COMPONENTS

LIST OF COMPONENTS

As seen in the previous diagram, the components of Project Induzione consist of the following:

01. Car with OBD-II Port
02. OBD-II to Serial Interface
03. LPC2148
04. SD card
05. GPS Module
06. RN-41 SM BlueTooth Module
07. Smartphone

Every car manufactures since 1996 has an OBD-II port, where various parameters about the car can be read and analyzed. We will retrieve data from the car through the OBD-II port. An ARM LPC2148 will handle the data we retrieve from the car via a scheduling technique while the user is driving. This data will be stored on an SD card and transmitted to a smartphone device using BlueTooth technology. A GPS will be responsible for logging location data as well as mileage count. All of this data will be displayed on an Android application.

In essence, the flow of technology in our device is as follows:

01. Microprocessor receives data from car and GPS
02. Microprocessor logs data to SD card
03. When user connects to device, microprocessor sends data over BlueTooth
04. User retrieves data on their phone

The Android application has features varying from data displaying, location displaying, virtual car upgrading, virtual racing, acheivements, a social hub, and a car club. These features are integrated from the data we retrieve from the OBD-II port and the GPS.

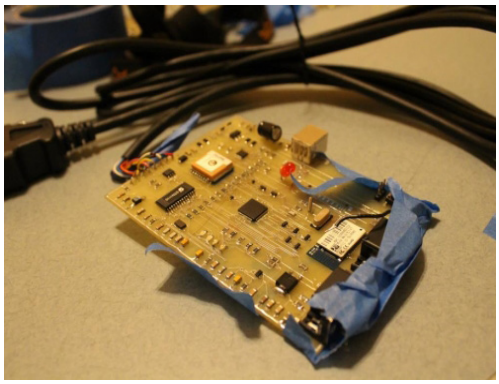
The next section provides a walkthrough of the above block diagram and the interconnecting components of technology that our project is composed of.

// COMPONENTS

WALKTHROUGH OF TECHNOLOGY

This section details a walkthrough of each functional block of technology that is included in our project. The first major component of our project is our user's car. Pictured below is a car -- a Mclaren MP4-12C (lovingly dubbed by our team as the Mclaren LPC2148).

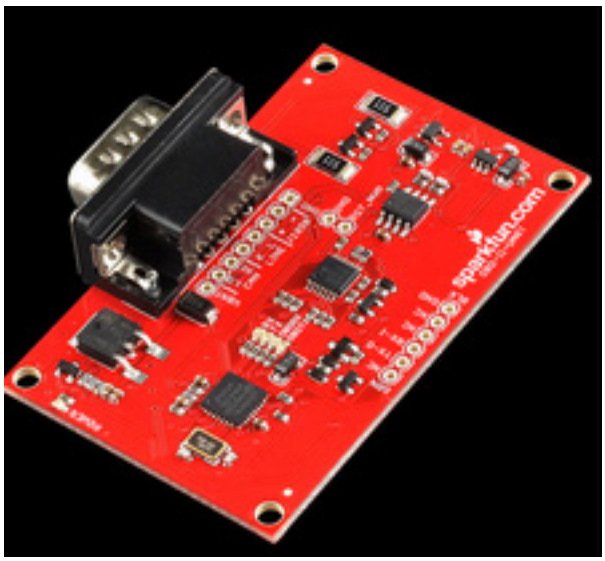
Aside from the 0-60 in 3.3 seconds the car user owns (or a more regular, pedestrian car than the one shown above) and the user's phone, only one component of Project Induzione is visible to the user. All of the following components will be placed within a case housing a PCB with the finished design. The user will only see an OBD-II connector to plug in and a block. Our goal is to be able to fit the PCB in a palm-fitting case/connector. We will be **creating our own custom PCB** for this design. For our final project we were unable to acheive the housing. Below is the PCB.



For protoyping purposes, the first component obfuscated to the user is an OBD-II to DB9 module that acts as the interface to the vehicle. The module will transcribe the OBD-II protocol into a workable UART form.

// COMPONENTS

WALKTHROUGH OF TECHNOLOGY CONTINUED



This UART module connects to an LPC2148 microprocessor via serial communication.

The LPC2148 interfaces with an MTK3339 GPS chip, which logs location data every ~2 minutes when the device is powered.

// COMPONENTS

WALKTHROUGH OF TECHNOLOGY CONTINUED

The logged data from both the GPS and OBD-II is stored into memory.

A BlueTooth module transfers this data wirelessly to a smartphone whenever the user chooses to the sync to the application.

(This page intentionally left blank.)

ROLES AND RESPONSIBILITIES

Our team and who was responsible for which aspects of the project.

(This page intentionally left blank.)

// ROLES AND RESPONSIBILITIES

Our team collaborated through all parts of the project. Every member has made a huge contribution to the project and has been a huge part of the team.

Garvin **Ling (Partner)**
Grandmaster at Arms
Garvin did most of the firmware and syncing logic for the Android application. This guy did a lot of the technical stuff associated with our project, and successfully tackled the challenge of creating a working OBD-II interface using the LPC2148 (what a boss!)

Anthony **Roncal (Partner)**
The Swift Scout
Anthony did a lot of the Android application and handling Social Hub aspects connected to our Parse DB backend. Dat app? Well, that's Anthony's doing.

// ROLES AND RESPONSIBILITIES

Paul **Camantigue (Partner)**
The Prodigal Explorer
Paul is our resident hardware master. He created both of our PCBs and is our master solderer. Making our PCB wasn't easy: the amount of components and the small pins of the LPC2148 make soldering a challenge. Paul was able to do it with ease.

Bo **Adepoju**
The Madman of Zaun
Bo did a lot of the UI/UX of the application and contributed to the firmware. He worked towards making sure that everything app-wise was boo-yoo-tiful.

FUNCTIONAL REQUIREMENTS

(This page intentionally left blank.)

// FUNCTIONAL REQUIREMENTS

The following are the functional requirements of our project.

POWER SOURCE FROM OBD-II

Our peripheral needs to be powered from the OBD-II port of the car and not require any other form of power.

BUILT IN GPS

THE GPS is a necessary part of our finished PCB design.

BUILT IN BLUETOOTH

The BlueTooth is a necessary component of our finished PCB design.

APPLICATION: REQUIRED TO COMMUNICATE WITH PERIPHERAL

Our Android application needs to be able to successfully receive and send data to our PCB.

APPLICATION: TABULAR VIEW

Our application's tabular view must contain the following:

- Home (shows info on your car / weekly challenges)
- Social Hub (car club and competitive, connect with friends and family
- Live data-logging / synced data displayed
- Diagnostics (show diagnostic trouble codes, reminders for oil changes, check tire pressure)

(This page intentionally left blank.)

SOFTWARE PROGRAM FLOW

A brief overview of the people our project would appeal to.

(This page intentionally left blank.)

// SOFTWARE PROGRAM FLOW

The application software powers our mobile device and it's phone-side interactivity with our PCB.

APPLICATION SOFTWARE DESCRIPTION

The application software is implemented as an Android application on an Android device. From here the user is able to communicate with the OBD-II interface via our embedded firmware. This is where the diagnostic checking and "gameplay" takes place. The application uses the data acquired from the OBD-II interface and applies it to the game. The user has the ability to acheive certain set goals for the week in order to gain points and level up. For example, the user will be given a goal such as one of the following:

01. Have a level coolant temperature for the entire week
02. Drive within a radius of a specified landmark or group of landmarks (i.e. drive past an In-N-Out) X amount of times in a week
03. Log onto the Project Induzione application X times in a week

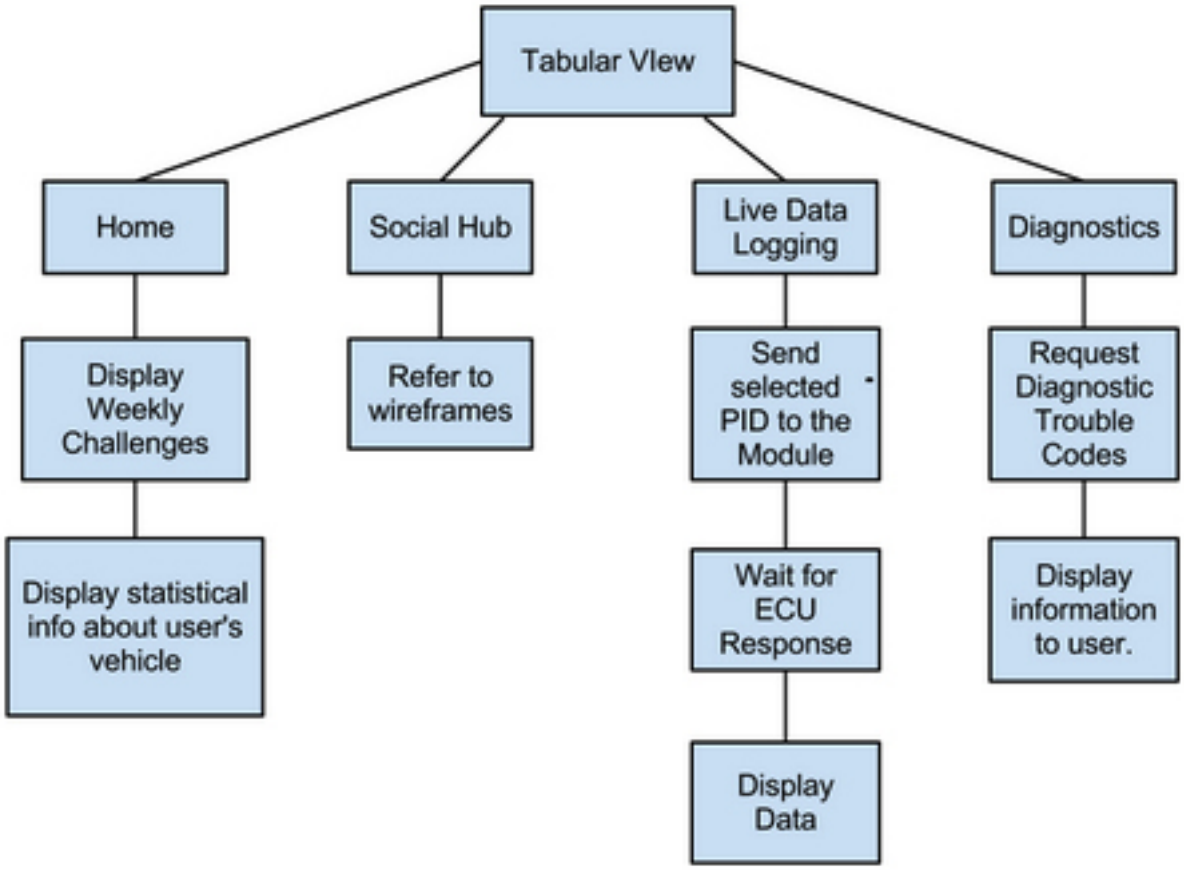
The product then accumulates statistics over the lifespan of this goal and reward the user accordingly.

Our product has passive logging capabilities and records data onto the SD card interface on our PCB. If the user chooses to do so, he/she can then use the application and "sync" up the SD card data to their phone. This way, the user does not need to have the application running in order to achieve certain gameplay goals. The user is also able to read live data from the OBD-II interface by sending out a byte to the BlueTooth module of the hardware board. The firmware then handles it and send it out through the OBD-II interface. This allows the user to display data such as engine rpm, air/fuel mixture ratio, and other parameters, in real time.

The flowchart shown below is a summarizzed version of our software flow in the Android application. It is a simple, tabbed, and organized set of screens that is easy to navigate through.

// SOFTWARE PROGRAM FLOW

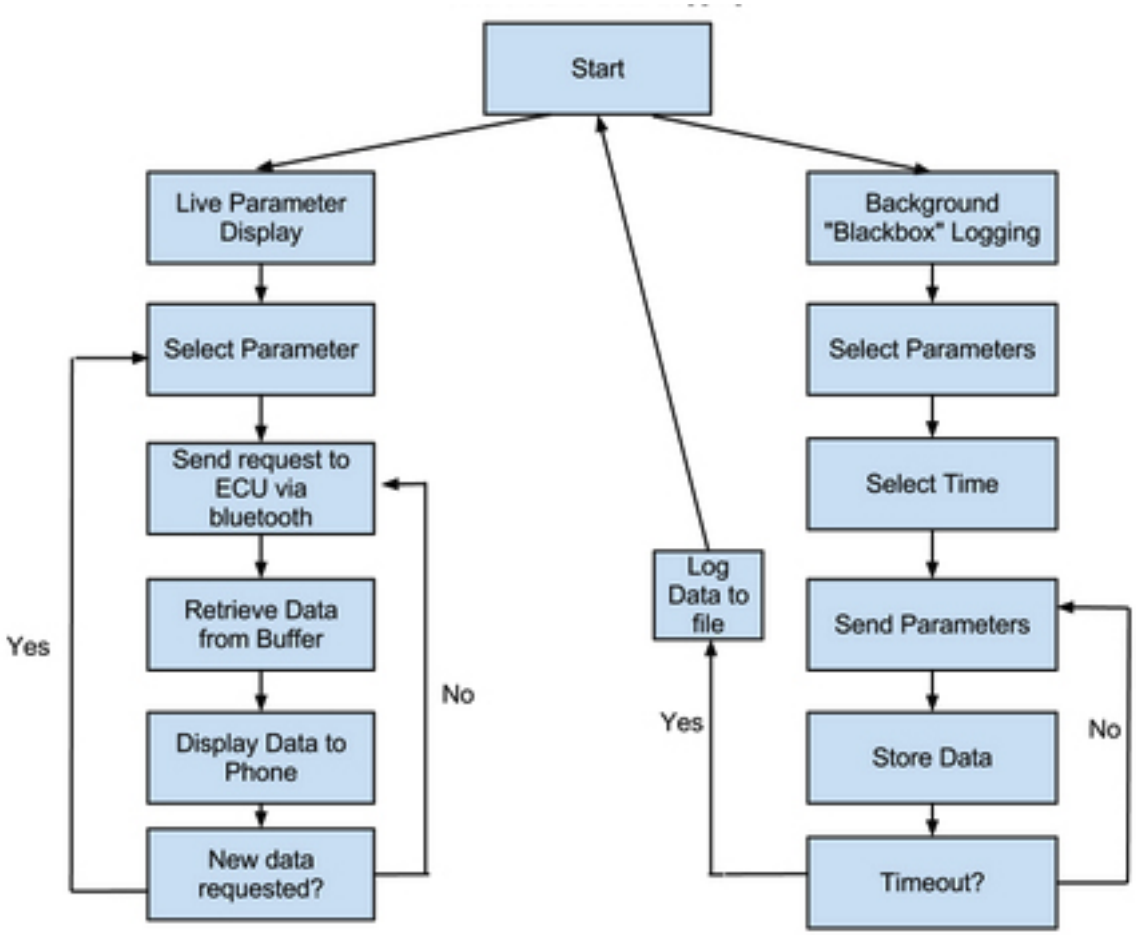
APPLICATION SOFTWARE DESCRIPTION CONTINUED



When one chooses to view the live data logging screen of the application, they will go through either one of two processes: live parameter displaying or background “blackbox logging”.

// SOFTWARE PROGRAM FLOW

APPLICATION SOFTWARE DESCRIPTION CONTINUED



LIVE PARAMETER DISPLAY

The Project Induzione peripheral constantly requests data corresponding to the user's choice through the smartphone application. This data is shown in real time, shown as a gauge for that specific parameter.

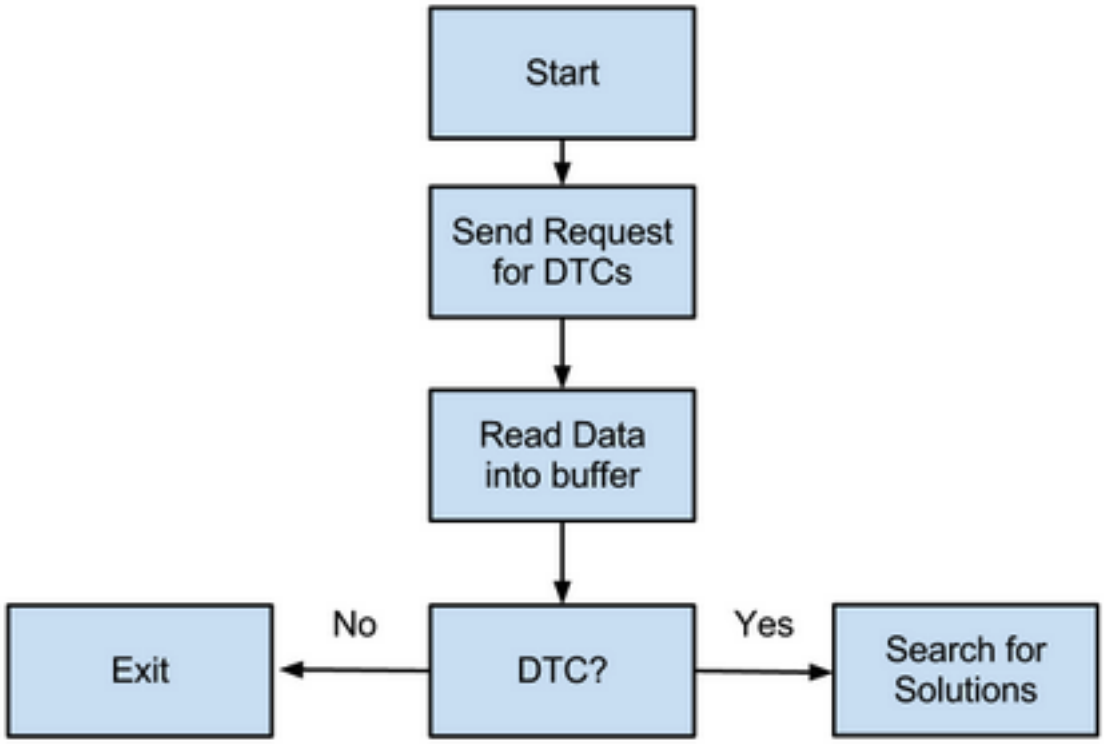
BACKGROUND "BLACKBOX" LOGGING

If something is wrong with a user's car, we want to give the user a choise to "log" and store the data to an SD card. This allows the user to send their logs to an automotive mechanic who understands engine management. This way, if the check engine light does not turn on, there can still be a way to find what is wrong with a user's vehicle. Our blackbox logging feature is also a useful tool to help the user keep track of either their own driving habits or the driving habits of other people who use their vehicle. For example, a parent might want to enable the blackbox logging mode to see whether or not their child was speeding or driving recklessly. This way, our logging functionality can prove to be

// SOFTWARE PROGRAM FLOW

BACKGROUND "BLACKBOX" LOGGING CONTINUED

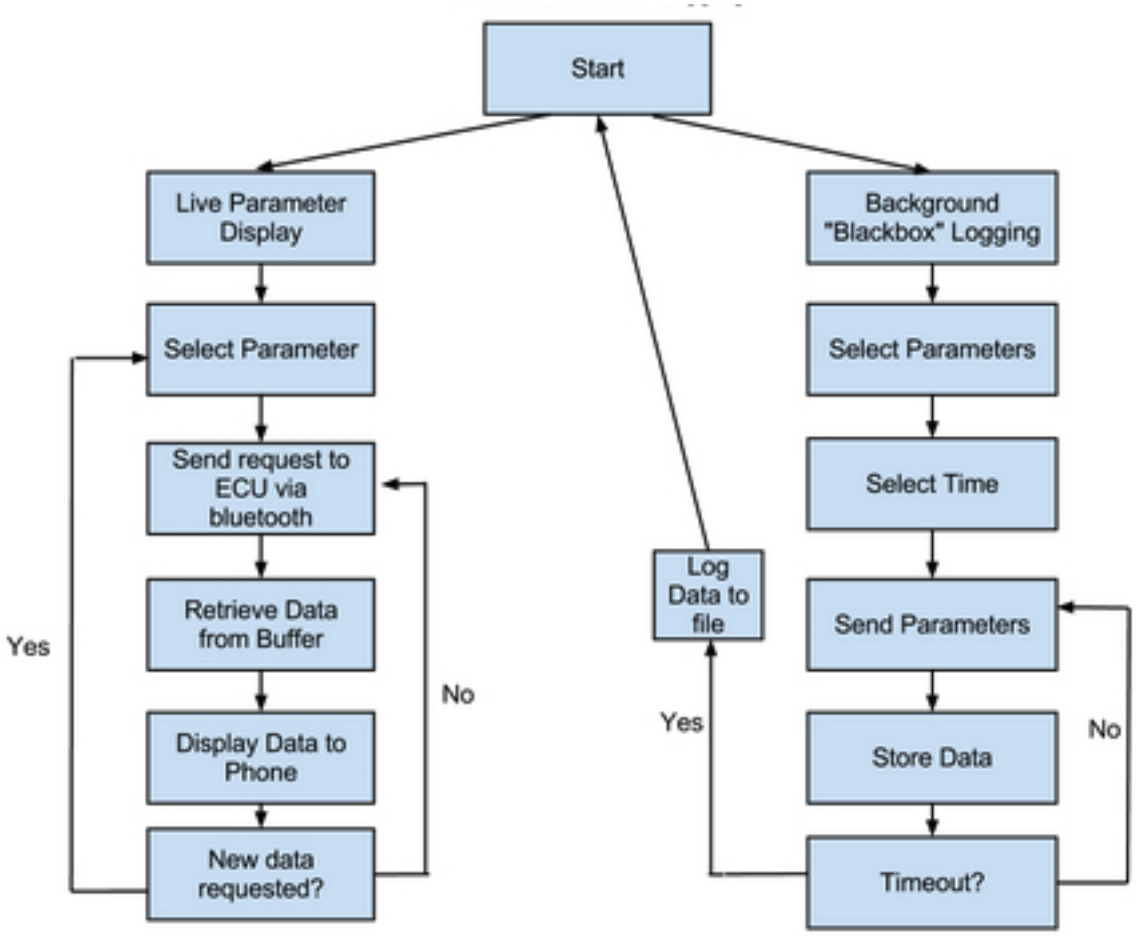
useful to both the enthusiast and the everyday commuter.



Whenever the check engine light goes on, the user is able to read what is called a "Diagnostic Trouble Code" from the ECU. These DTCs are universal identifiers for various problems that arise in an automobile. If a user's car triggers the check engine light, we can request the data via OBD-II and retrieve a code back respective to their current error. Once our application detects an error code in the vehicle's ECU, the application will launch the phone's browser and directly run a search for causes and solutions pertaining to that specific error code.

// SOFTWARE PROGRAM FLOW

APPLICATION SOFTWARE DESCRIPTION CONTINUED



LIVE PARAMETER DISPLAY

The Project Induzione peripheral constantly requests data corresponding to the user's choice through the smartphone application. This data is shown in real time.

BACKGROUND "BLACKBOX" LOGGING

If something is wrong with a user's car, we want to give the user a choise to "log" and store the data to an SD card. This allows the user to send their logs to an automotive mechanic who understands engine management. This way, if the check engine light does not turn on, there can still be a way to find what is wrong with a user's vehicle. Our blackbox logging feature is also a useful tool to help the user keep track of either their own driving habits or the driving habits of other people who use their vehicle. For example, a parent might want to enable the blackbox logging mode to see whether or not their child was speeding or driving recklessly. This way, our logging functionality can prove to be

// SOFTWARE PROGRAM FLOW

FACEBOOK LOGIN CODE

```
/**
 *
 * @param onClickShowDialog() method
 *
 */
private void onClickShowDialog(){
    // starts Facebook Login
    ParseFacebookUtils.logIn(Arrays.asList("email", Permissions.Friends.ABOUT_ME), this, new LogInCallback(){
        @Override
        public void done(ParseUser user, ParseException err){
            if (user == null){
                Log.d("Project Induzione", "The user cancelled the Facebook login.");
            } else if (user.isNew()){
                getFacebookIdInBackground();
                Log.d("Project Induzione", "User signed up and logged in through Facebook!");
                showAlert(user);
            } else {
                getFacebookIdInBackground();
                Log.d("Project Induzione", "User logged in through Facebook");
                showAlert(user);
            }
        }
    });
}
```

Shown above is a code snippet demonstrating our Facebook login implementation. Our application has a hook to Facebook that checks whether the user already has an existing session in the application via Facebook, if they have not yet registered, or if they are logging in for the first time.

// SOFTWARE PROGRAM FLOW

The user interface / user experience (UI/UX) for our product is unique in the fact that the user is receiving two parts: both hardware peripheral and mobile application. The main user interface is provided via the application.

USER INTERFACE COMPONENTS

The main components of the UI for our product are listed below:

UI COMPONENTS

- LEDs on hardware peripheral to indicate connectivity status
- Slick mobile application with:
    - Enhanced dashboard (for data displaying)
    - Stat tracking / rewarding game interface

The user will not be observing any moving hardwar,e as all calculations and communication is done in the PCB and wirelessly over BlueTooth. The user interacts with our product via the Android application.

The final product features a slick and user friendly application interface. Each user will be asked to create an account upon initial startup to associate their respective device. If a user already has an account, they will be greeted with a login screen asking for their username and password.

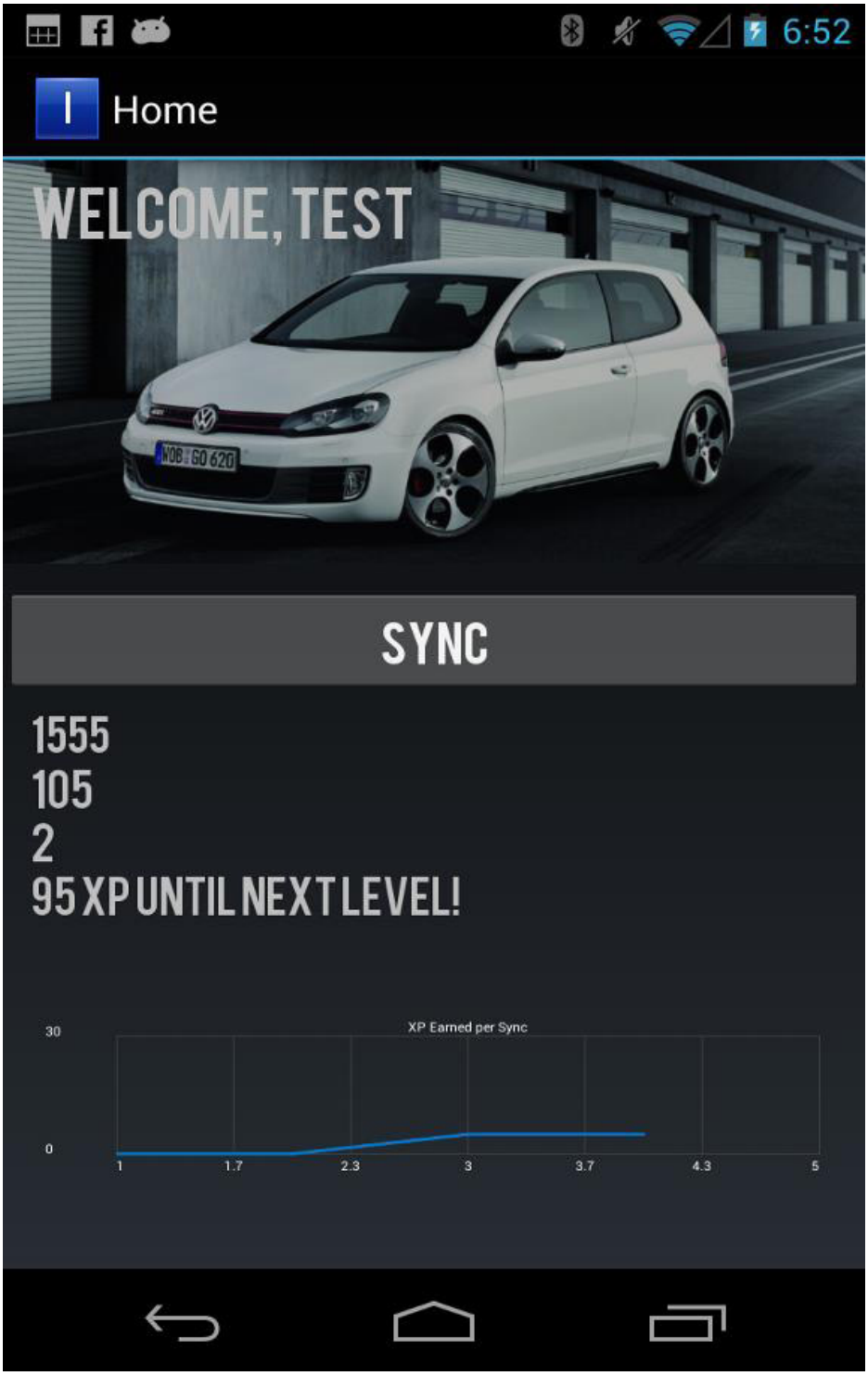
Upon successful login, it will display multiple tabs consisting of a home screen, “My Car”, social hub, car club, and diagnostics. Each tab will contain information ranging from details about a user’s car, driving habits, and friend information (including stats on a friend’s car).

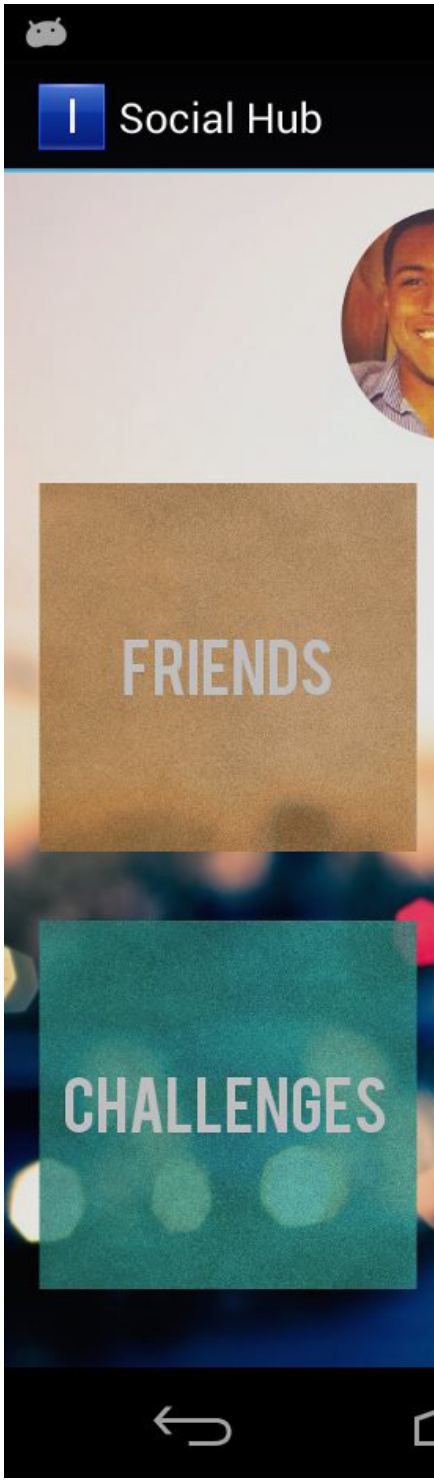
APPLICATION WIREFRAMES

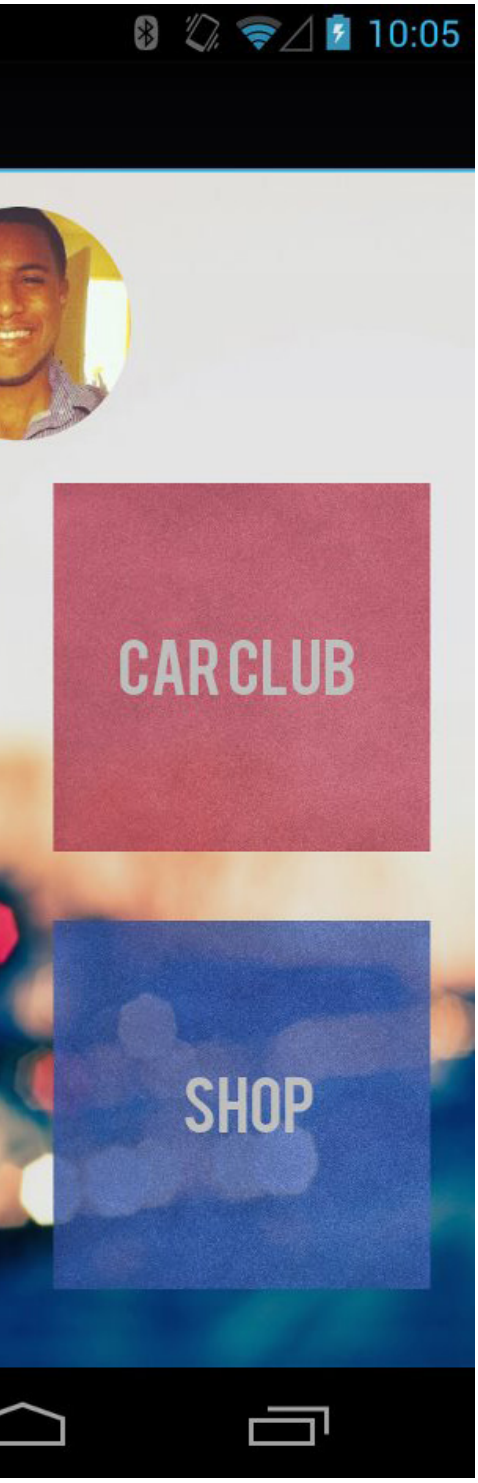
Provided in this section are early mockups that our team had created for the application we are developing.

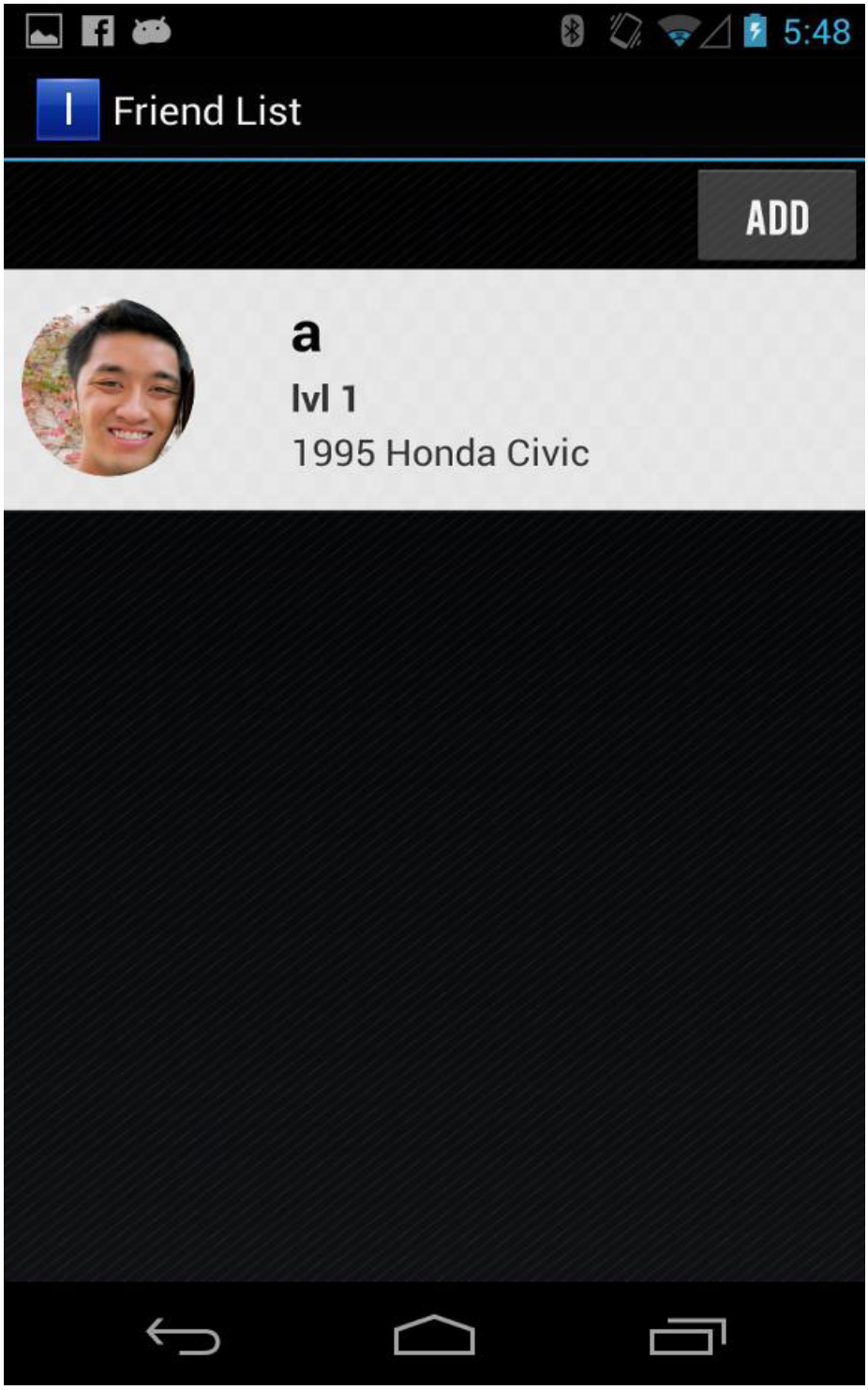
OUR APPLICATION

Provided below are some screenshots of our actual Android application!






10:05
CAR CLUB
SHOP


5:48
Friend List
ADD
a
lvl 1
1995 Honda Civic

HARDWARE CONFIGURATION

A technical description of our design and configuration solution including flowcharts, block diagrams, schematics, and explanations of each.

(This page intentionally left blank.)

// HARDWARE

Project Induzione's hardware needed to be smart, succinct, and well packaged. An ARM7 chip interfacing to BlueTooth and GPS chips is the core of our design, components which allow for us to create a small yet capable PCB.

HARDWARE METRICS

An ARM7 microprocessor is the core of our custom PCB design, interfacing to both BlueTooth and GPS modules. An ELM/STN1110 chip is used to translate ASCII to OBD-II protocols, and a CAN transceiver to transfer data over the OBD-II lines. We will be using a scheduling technique on our LPC2148 to smartly choose priority between UART data lines.

Provided below are some important metrics of the hardware required for our product.

OPERATING VOLTAGES
- LPC2148: 6.5 V to 3.3 V / 5 V DC (via linear regulators)
- Roving Networks BlueTooth Module: 3.3 V - 3.6 V DC
- Adafruit GPS chip: 5 V - 5.5 V
OPERATING CURRENTS
- LPC2148: 500 mA
- Roving Networks BlueTooth Module: 8-30 mA connected, 2mA idle
- AdaFruit GPS chip: 20 mA
BLUETOOTH OPERATION SPECIFICATIONS
- 15 dBm Transmitter
- 80 dBm Receive Sensitivity
- Operation Range: 100m Range
- Operation Frequency: 2402 MHz - 2480 MHz

Chart continued on next page.

// HARDWARE

- UART Interface (SPP, HCI)
  - SPP data rates: 240 Kbps (slave), 300 Kbps (Master)
  - HCI data rates: 1.5 Mbps sustained, 3.0 Mbps burst in HCI mode.

- Low Power Mode: 30 mA connected, < 10 mA sniff mode

GPS OPERATION SPECIFICATIONS

- 10 Hz Update Rate
- Output: NMEA 0183, 9600 baud default
- Antenna Size: 15mm x 15mm x 4mm
- Satellites: 22 tracking, 66 searching
- Position Accuracy: 1.8m

PACKAGED PRODUCT SPECIFICATIONS

- Operating Voltage: 5 V DC
- Operating Current: 500 mA
- Case Material: Flame Retardant Polycarbonate

// HARDWARE

OBD-II PINOUTS

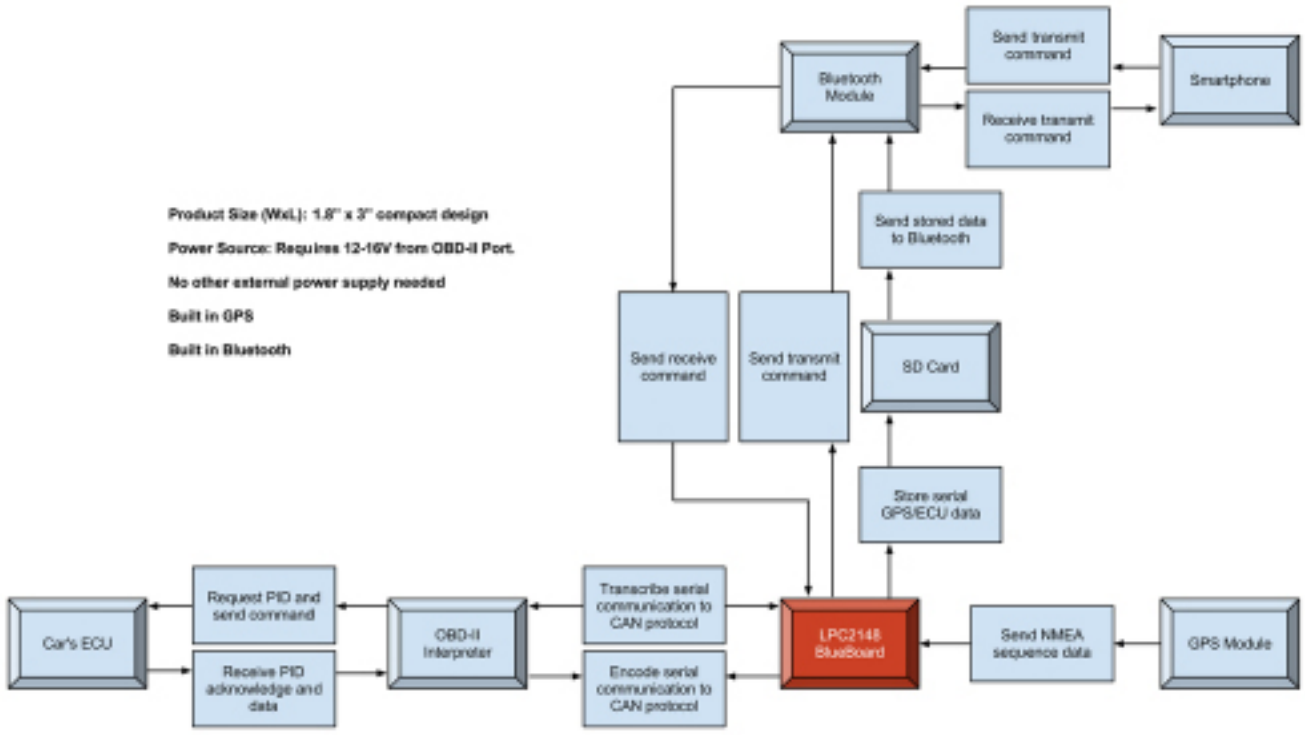
The pinouts of the OBD-II car interface are provided below.

PIN DESCRIPTION	OBD-II PIN	DB9 PIN	
J1850 BUS+.	2	7	
Chassis Ground	4	2	
Signal Ground	5	1	
CAN High J-2284	6	3	
ISO 9141-2 K Line	7	4	
J1850 BUS-	10	6	
CAN Low J-2284	14	5	
ISO 9141-2 L Line	15	8	
Battery Power	16	9	

// HARDWARE

DATA FLOW CHART

Provided below is a flow chart showing the data flow in between our several hardware components.



// FIRMWARE

The firmware of our project drives our embedded system to complete tasks effectively and efficiently.

FIRMWARE DESCRIPTION

The firmware portion of the code is written in C and embedded in our microprocessor. The responsibility of the firmware is to manage the communication between the client device (Android Device) and the host device (OBD-II Interface). The OBD-II interface has five different communication protocols and varies from brand to brand. For example, most European and Asian import vehicles use ISo 9141 / KWP2000 circuitry while most GM cars and trucks use SAE J1850 VPW (Variable Pulse Width Modulation). The protocol can be identified by observing which pins are connected on the OBD-II interface. However, since our product will be used across a wide array of vehicle makes and models, our firmware will dynamically select the protocol for the OBD-II interface that it is connected to.

Once the firmware establishes the communication protocol, it can then communicate with the OBD-II interface via serial communication. The OBD-II serial peripheral serves as the communication medium between the LPC2148 and the engine control unit of the vehicle. The GPS module actively collects latitude/longitude data and continuously sends it out via the tx line, even if it has not found a fix yet.

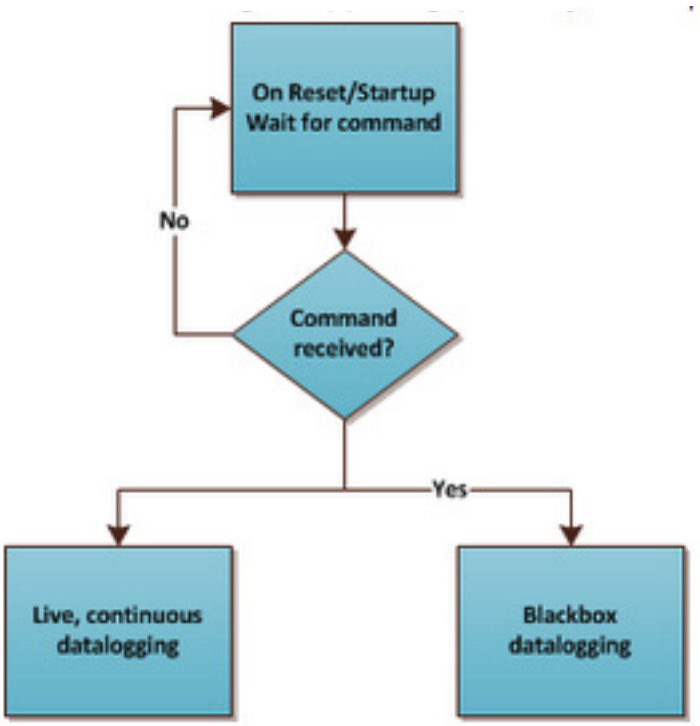
Since the LPC2148 has two serial peripheral wired to one UART, it is up to the firmware to control which peripheral talks to the board at any given moment in time. We are managing this by setting up two transistors, wired from the transmit of both the OBD-II serial peripheral and the GPS. We will use the on-chip GPIOs to switch between which peripheral data is to be received.

Below is a flowchart showing our main loop for our embedded firmware. It will consist of two cases, depending on if a command is received. The command is received from the phone when the user wants to log data parameters from their car. If a command is not received it will stay in an idle state where no intensive activity is happening, i.e. low power mode.

PARAMETER IDs

To get the data from the OBD-II interface, the firmware sends out a byte called the Parameter ID (PID) through the serial interface. Parameter IDs are OBD-II protocol communication codes that are sent to the ECU in order to request/retrieve data from the vehicle. OBD-II diagnostics are standard in vehicles manufactured since 1996. While most of the common parameter IDs are available in all cars, some vary depending on sensors present in the vehicle.

// FIRMWARE



PARAMETER IDs

To get the data from the OBD-II interface, the firmware sends out a byte called the Parameter ID (PID) through the serial interface. Parameter IDs are OBD-II protocol communication codes that are sent to the ECU in order to request/retrieve data from the vehicle. OBD-II diagnostics are standard in vehicles manufactured since 1996. While most of the common parameter IDs are available in all cars, some vary depending on sensors present in the vehicle.

// FIRMWARE

PARAMETER IDs CONTINUED

Parameter IDs consist of two bytes. The first byte is the mode, ranging in value from 00-09. Below is a table of the modes of operation when the OBD-II port is utilized.

OBD-II HEXADECIMAL RADIX	MODE OF OPERATION
01	Show current data
02	Show freeze frame data
03	Show stored Diagnostic Trouble Codes
04	Clear Diagnostic Trouble Codes
05	Test results, oxygen sensor monitoring
06	Component/System Monitoring (CAN only)
07	Show pending Diagnostic Trouble Codes
08	Control operation of on-board components
09	Request vehicle information

We primarily use mode 01 and mode 03 since these are the modes to retrieve live engine diagnostics and statistics. The second byte is the actual ID command. For example, if the firmware sends out a byte 0x0C (the PID for engine RPM), the OBD-II interface will respond with the vehicle of the current engine RPM of the vehicle. The response from the ECU will be in the form of 4 bytes.

1st byte. Mode + 0x40 (ex. for this case it would be 0x41)
2nd byte. Echo Mode (ex. for this case it would be 0x0C)
3rd byte. Upper byte of data
4th byte. Lower byte of data

(For RPM, we take the hex value and divide it by 4 to get the actual RPM value.)

CONVERTING VALUES

When we get the response back (for example, 1A 3F), we will receive the bytes as ASCII values (0x31, 0x41, 0x33, 0x46). We however want the actual decimal values of each character. Therefore, we will have a method in our firmware that converts the received ASCII values into their respective decimal values.

We then have a method that calculates/converts that full hex value to its respective decimal value. This value is then put through the appropriate conversions. However, before we send this data

// FIRMWARE

CONVERTING VALUES CONTINUED

to the Android device, we must convert those integer values back to their ASCII forms. The integer values will be split up into an integer array with each digit having its own offset. Each digit is then converted back into its ASCII form by adding 0x30 to the number.

This data is sent out through the BlueTooth module to the Android device. This cycle will keep happening until the user chooses to stop reading. Therefore, the data on the phone will be updated dynamically and the user will be able to continuously see what is going on in their car. We will be utilizing a scheduler to prioritize tasks in the firmware.

Below is a table of the many PIDs that we will be logging/displaying, their PID codes, and their conversion formulas.

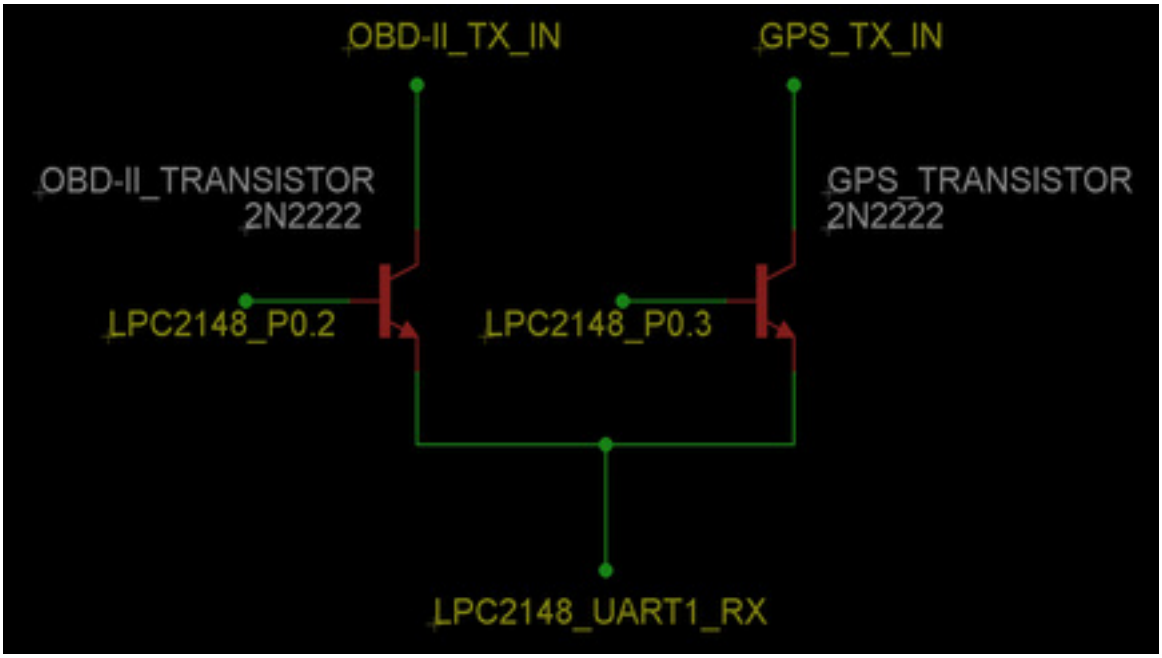
DESCRIPTION	MODE	PID	UNITS	FORMULA
Calculated engine load value	01	04	%	A*100 / 255
Engine coolant temperature	01	05	Celsius	A - 40
Fuel pressure	01	0A	kPa	A*3
Intake manifold absolute pressure	01	0B	kPa	A
Engine RPM	01	0C	RPM	((A*256)+B)/4
Vehicle speed	01	0E	km/h	A
Timing advance	01	0F	degrees	A/2 - 64
Intake air temperature	01	10	Celsius	A - 40
MAF air flow rate	01	11	Celsius	((A*256)+B)/100
Throttle Position	01	22	grams/sec	100*A/255
Fuel Rail Pressure	01	23	%	A
Fuel Level Input	01	2F	kPa	A-40
Barometric pressure	01	33	kPa	A-40
Ambient air tempeature	01	46	%	(((A*256)+B)-26
Engine oil temperatue	01	5C	kPa	A-125
Fuel injection timing	01	5D	Celsius	A-125
Driver's demand engine	01	61	Celsius	A*256 + B
Actual engine - percent torque	01	62	degrees	A
Engine reference torque	01	63	percent	A*256 + B

// FIRMWARE

SCHEDULER LOGIC

Our scheduler utilizes the on-board general purpose input/output ports (GPIO) to control which peripheral gets the attention of the LPC2148's UART. The firmware's scheduler will not be interrupt nor timer based. It will utilize counters based on whether or not we finished logging a specific amount of OBD-II data from the car. When it has reached a preset number of data parameters it will turn one GPIO "off" and another one "on" to drive our transistor circuit. This way, both peripherals are constantly outputting data so there will be no delay besides the timing in the switching of the transistors.

The schematic below shows the circuit dicussed above. It utilizes only two transistors: one for OBD-II data and the other for GPS data.



// FIRMWARE

CONVERTING VALUES

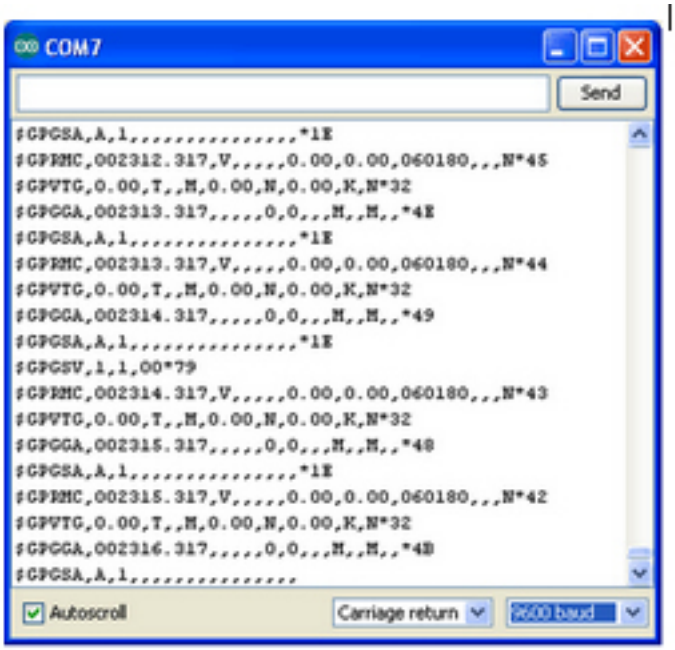
When we get the response from the Car's OBD-II, we receive bytes as ASCII values (i.e., 0x31, 0x41, 0x33, 0x46). We howeve want the actual decimal values of each char. Therefore, we have a function in our firmware that converts the ASCII values into their respective decimal values.

```
int convertDecimal(char charVal){
      int decimalValue;
          if(charVal >= 0x30 && charVal<=0x39) { //Char is 0-9
                decimalValue = charVal - 0x30;

                //remainder will be the decimal value.

                return decimalValue;  //change this.
          }//end 0-9

          else if(charVal>=0x41 && charVal<=0x46){   //Char is A-F
                decimalValue =  charVal - 0x31;
                decimalValue = decimalValue - 6; //Adjust value
                return decimalValue;
          }
          else{
                 return 0;
          }
}// end convertDecimal
```

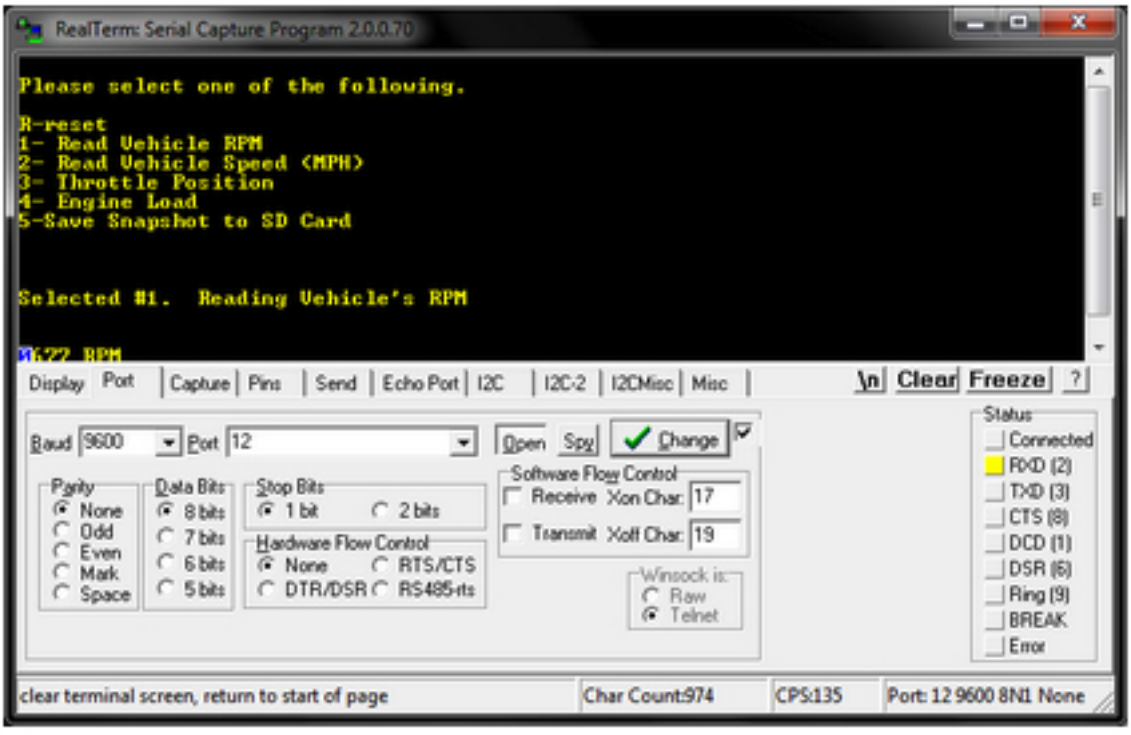
We then have a function that calculates/converts that full hex value to its respective decimal value. This value is then put through the appropriate conversions. The data is now ready to be transmitted back out to display to the user. However, before we do this we must convert those integer values back to its ASCII form. We split up the integer value into an integer array with each digit having its own offset. Each digit is then converted back into its ASCII form by adding 0x30 to the number.

The GPS outputs constantly outputs data, even if it has not found a fix yet. This data looks like this:

// FIRMWARE

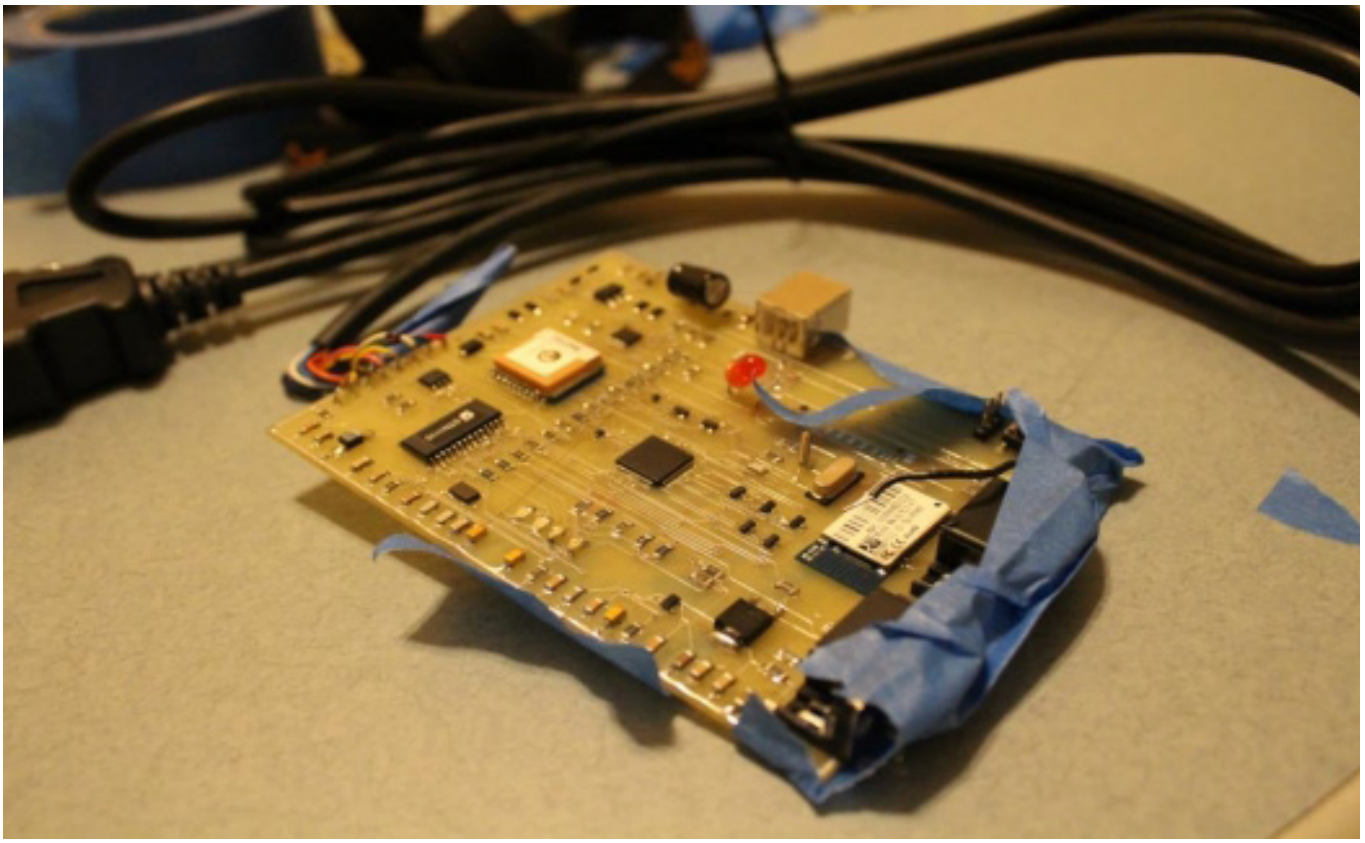


An example of us testing our firmware is provided below. Shown is our firmware logging vehicle RPM through OBD-II connection.



// OUR HARDWARE

Pictured below!

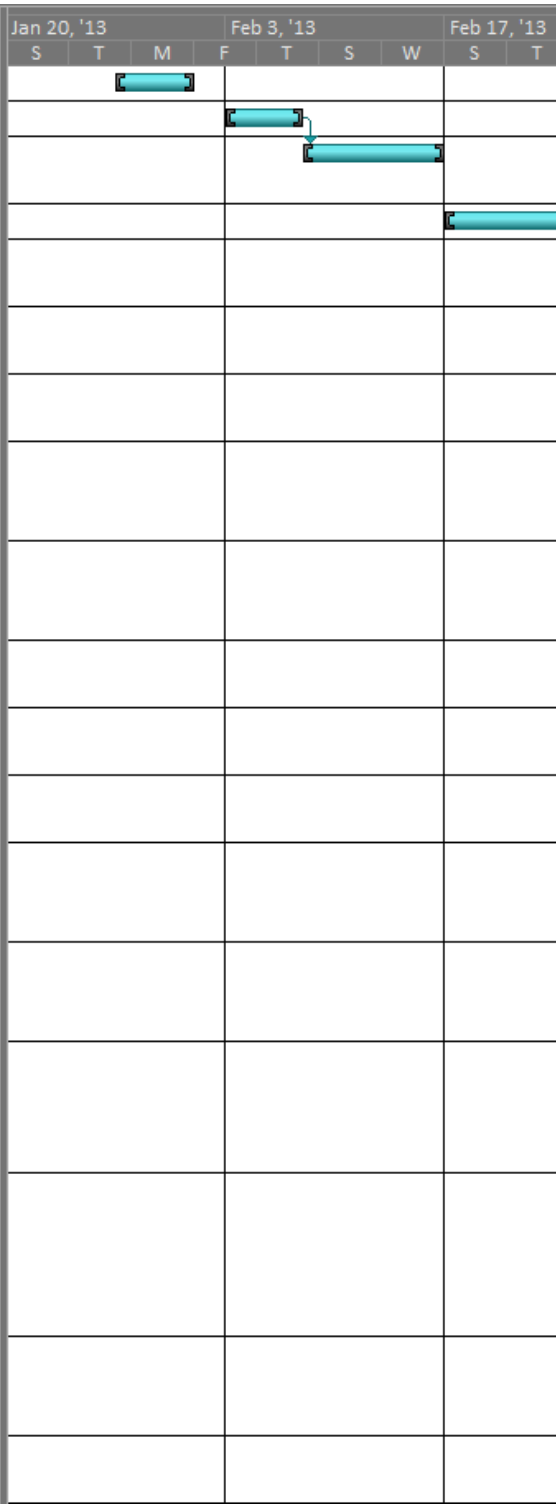


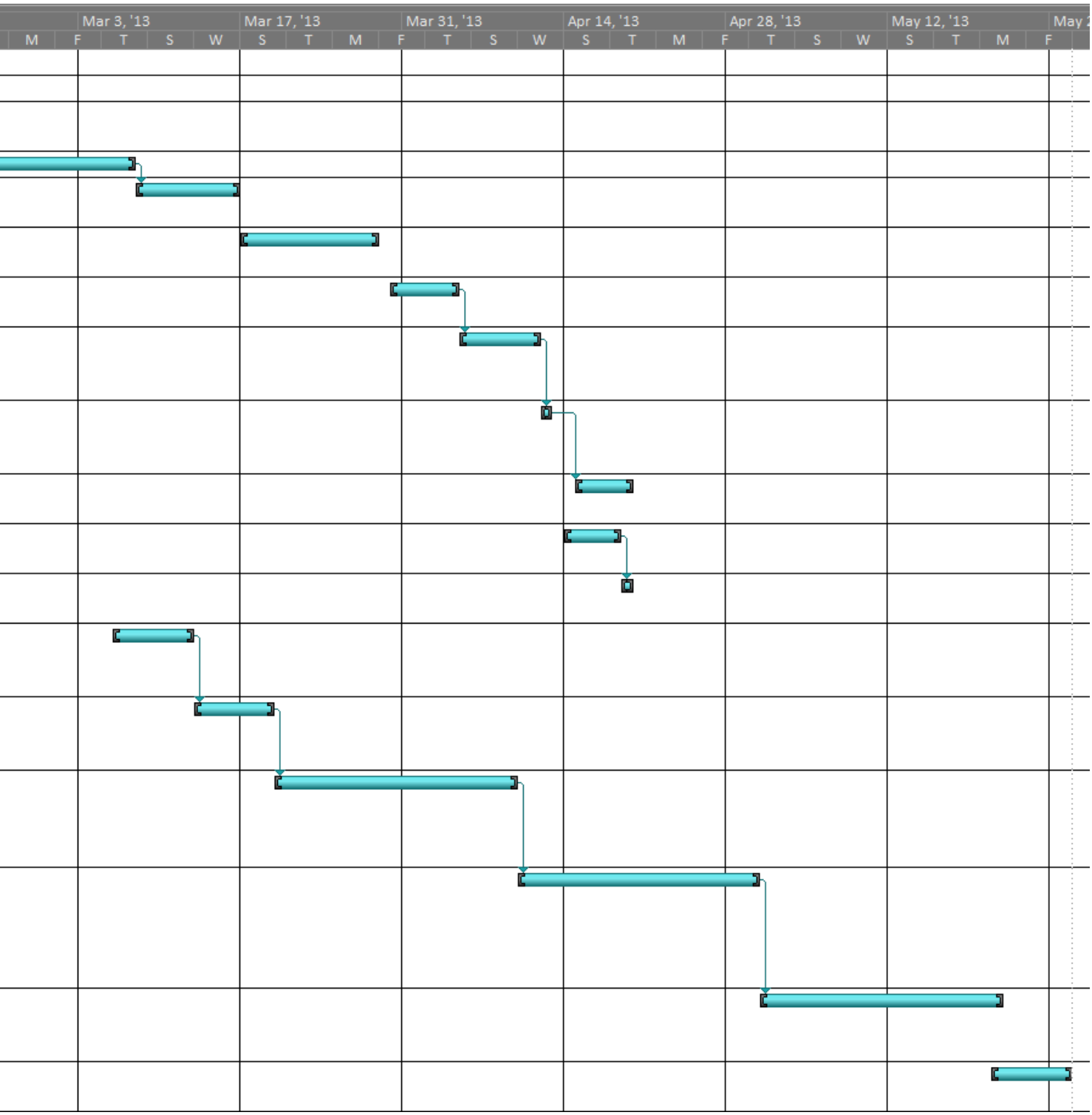
SCHEDULE - GANTT CHART

(This page intentionally left blank.)

// GANTT CHART

	Task Name	Duration	Start	Finish	Predecessors
1	Purchase all modules	5 days	Sun 1/27/13	Thu 1/31/13	
2	GPS data logging	1 wk	Sun 2/3/13	Thu 2/7/13	
3	GPS data parsing and storing to SD card	1.4 wks	Fri 2/8/13	Sat 2/16/13	2
4	OBD-II data logging	3 wks	Sun 2/17/13	Thu 3/7/13	
5	OBD-II data parsing and storing to SD card	1.4 wks	Fri 3/8/13	Sat 3/16/13	4
6	Bluetooth UART communication	2 wks	Sun 3/17/13	Thu 3/28/13	
7	Purchase surface mount version of the modules	5 days	Sat 3/30/13	Thu 4/4/13	
8	Create board and schematic using Eagle CADsoft	1 wk	Fri 4/5/13	Thu 4/11/13	7
9	Create gerber file and send it off for rapid prototyping	1 day	Fri 4/12/13	Fri 4/12/13	8
10	Solder board together and test	5 days	Mon 4/15/13	Fri 4/19/13	9
11	Plastic case enclosure research	1 wk	Sun 4/14/13	Thu 4/18/13	
12	Order plastic case enclosure	1 day	Fri 4/19/13	Fri 4/19/13	11
13	Application phase: determine development environment	1 wk	Wed 3/6/13	Tue 3/12/13	
14	Application phase: wireframing/screens/UX,I Flow	1 wk	Wed 3/13/13	Tue 3/19/13	13
15	Application phase: begin prototyping application, divide application into segments	3 wks	Wed 3/20/13	Tue 4/9/13	14
16	Application phase: bring and store data locally into an Android phone via Bluetooth in the application	3 wks	Wed 4/10/13	Tue 4/30/13	15
17	Application phase: server-side implementation	3 wks	Wed 5/1/13	Tue 5/21/13	16
18	Bring entire application together	1 wk	Tue 5/21/13	Mon 5/27/13	




Mar 3, '13
Mar 17, '13
Mar 31, '13
Apr 14, '13
Apr 28, '13
May 12, '13
M F T S W S T M F T S W S T M F T S W S T M F

SOFTWARE - TASK DESCRIPTIONS

(This page intentionally left blank.)

// SOFTWARE TASK DESCRIPTIONS

Software tasks for our project were the following:

GPS DATA LOGGING

For the GPS, we had to retrieve the sequence and parse it for the desired latitude, longitude, and timestamp. In order to do so, we had to set up a loop within our passive firmware logging to capture this data at any point and gather (parse) only the data that is needed.

GPS DATA PARSING AND STORING TO SD CARD

Storing the aforementioned GPS data to the SD card required placing the data needed in a buffer and storing it at the end of our passive logging loop.

OBD-II DATA LOGGING

The meat of our project -- storing the OBD-II data to the SD card. This involved communicating with the car and retrieving the necessary parameters, and storing it back to the SD card.

BLUETOOTH UART COMMUNICATION

This task involved sending the required data to the phone when a sync is requested, i.e. how a user retrieves all of the parameters for our Android application.

APPLICATION PHASE: DETERMINING DEVELOPMENT ENVIRONMENT

This task involved sending the required data to the phone when a sync is requested, i.e. how a user retrieves all of the parameters for our Android application.

APPLICATION PHASE: WIREFRAMING/SCREENS/UI-UX FLOW

This involved creating the detailed mockups of how we would like the application to look and feel.

APPLICATION PHASE: BRING AND STORE DATA LOCALLY INTO ANDROID PHONE

This was a huge step for us -- syncing data from our firmware and custom design to the application for us to play with. This involved sending data via BlueTooth and retrieving it appside.

APPLICATION PHASE: SERVER-SIDE IMPLEMENTATION

Our application relies on a lot of data. We used Parse as a server-side implementation of our Android application.

APPLICATION PHASE: COMPLETED APPLICATION

The last step -- the completed application! This is where the software is wrapped up with a bow on top, signifying our finished work of the project.

HARDWARE - TASK DESCRIPTIONS

(This page intentionally left blank.)

// HARDWARE TASK DESCRIPTIONS

Hardware tasks for our project were the following:

FUNCTIONAL DESIGN PROTOTYPE (BLUEBOARD)

Putting everything together! Having our working project on the LPC2148 BlueBoard was a major acheivement for our project.

PURCHASE SURFACE MOUNT VERSION OF THE HARDWARE COMPONENTS

This step involved buying the parts to be used for our custom PCB design.

CREATE BOARD AND SCHEMATIC USING EAGLE CADSOFT

Creating the board! Putting everything together for the PCB.

SOLDER BOARD AND TEST

Soldering every component ordered, and testing. We created two PCBs for this step.

(This page intentionally left blank.)

CONCLUSION

A summary of our project and the problems we have experienced.

(This page intentionally left blank.)

// CONCLUSION

Looking forward.

PROBLEMS ENCOUNTERED AND CONCLUSION

As stated in the previous section, we had some minor setbacks when ordering the PCB because the footprint we used in EAGLE for our DB9 Serial port was the wrong spec. We then corrected the schematic and board and have reordered the PCB.

We also had setbacks in being able to program the LPC2148 on the PCB (!). Once again, these are issues that could have been resolved with more time.

Developing a product that is threefold: (I) hardware based, (II) firmware based, and (III) software based requires a lot of hard work, team coordination and cooperation.

By eliminating variables within the development of our project, our team has worked hard to make sure our design is as straightforward as possible.

Our team has done a great job in creating something that we love, and something that can be built upon further so that the world can also love it.

We look forward to continuing to build the product and the dream.

LOVE,
Team Always Block

(This page intentionally left blank.)

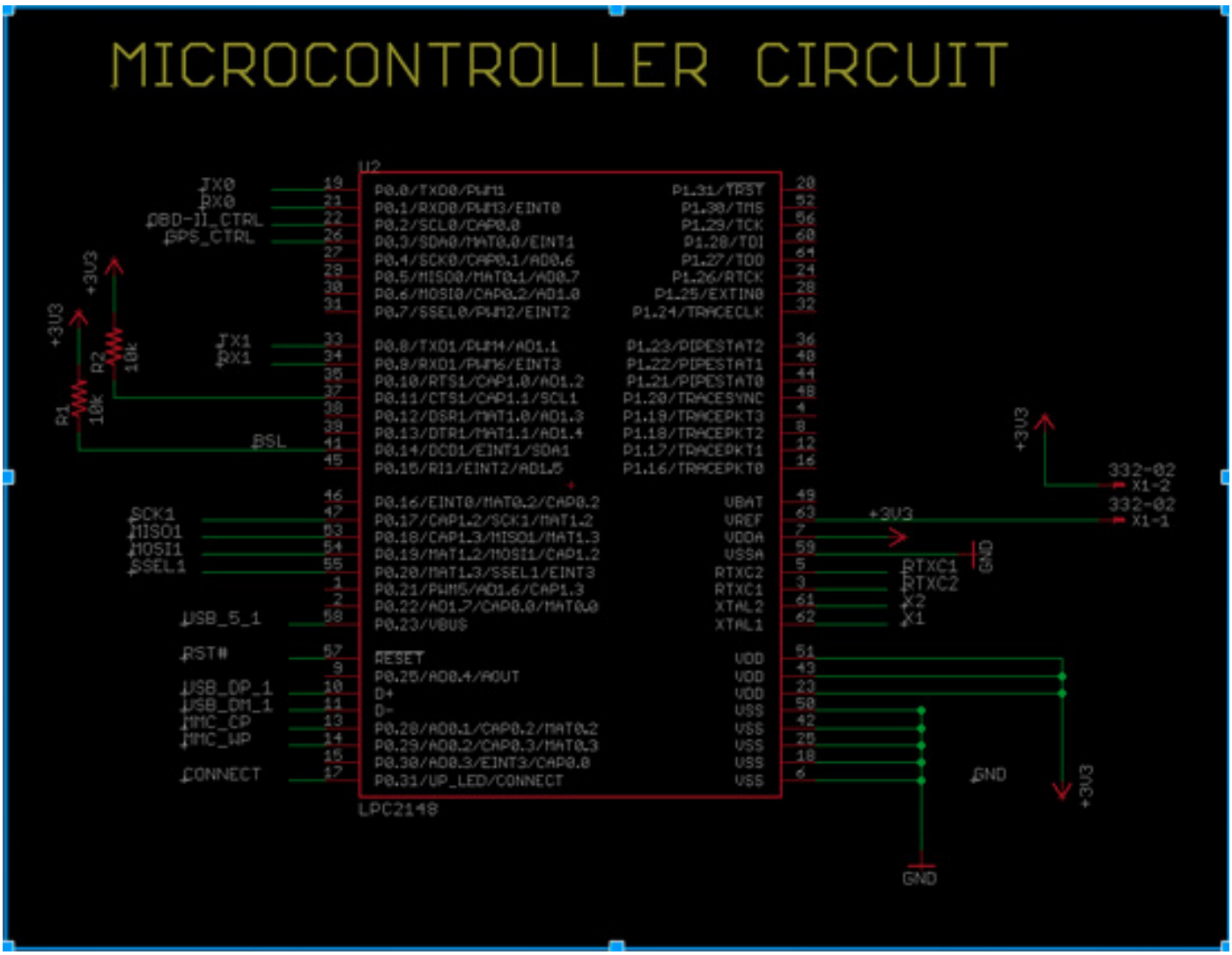
APPENDIX - SOFTWARE LISTINGS

(This page intentionally left blank.)

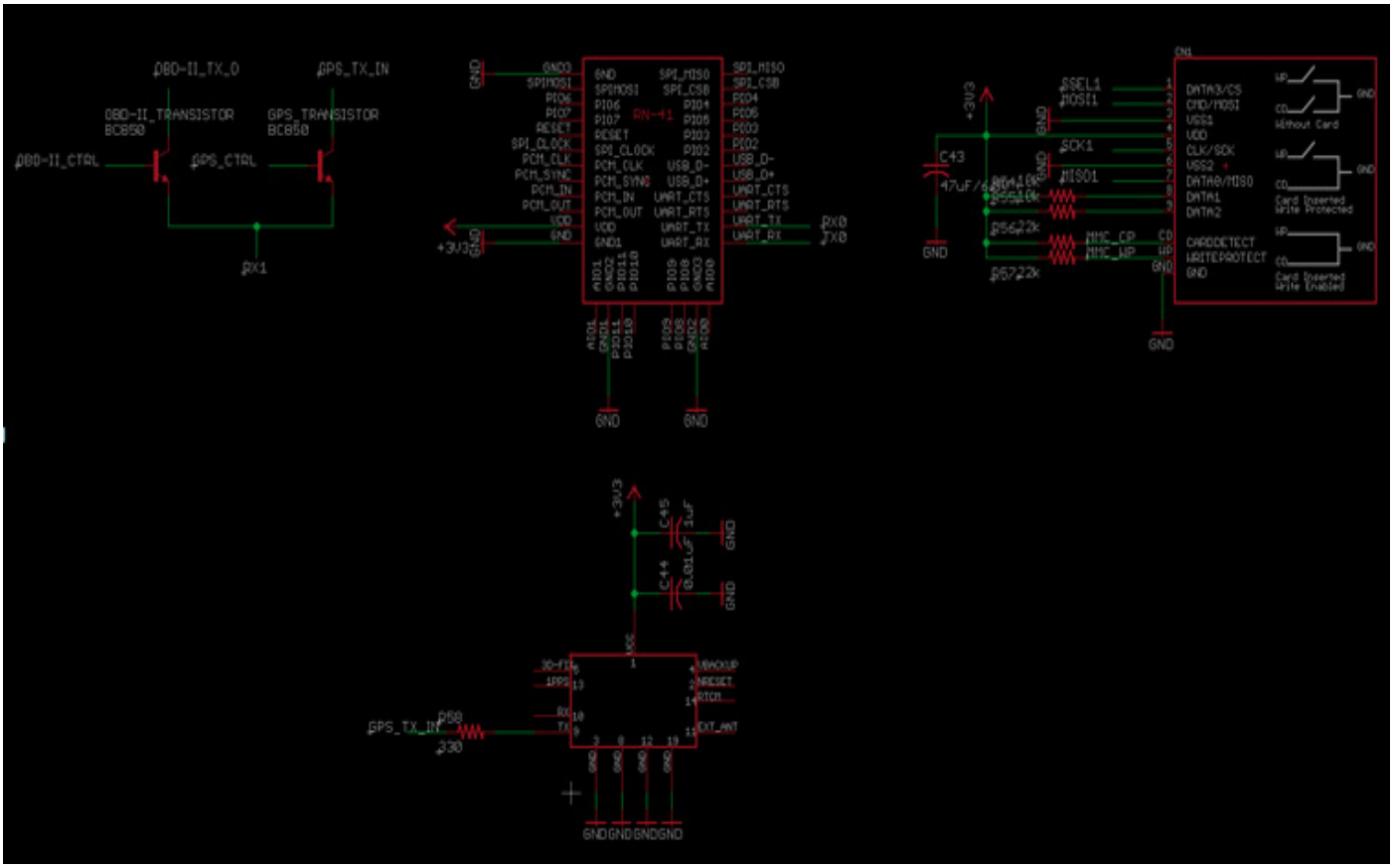
APPENDIX - SCHEMATICS

(This page intentionally left blank.)

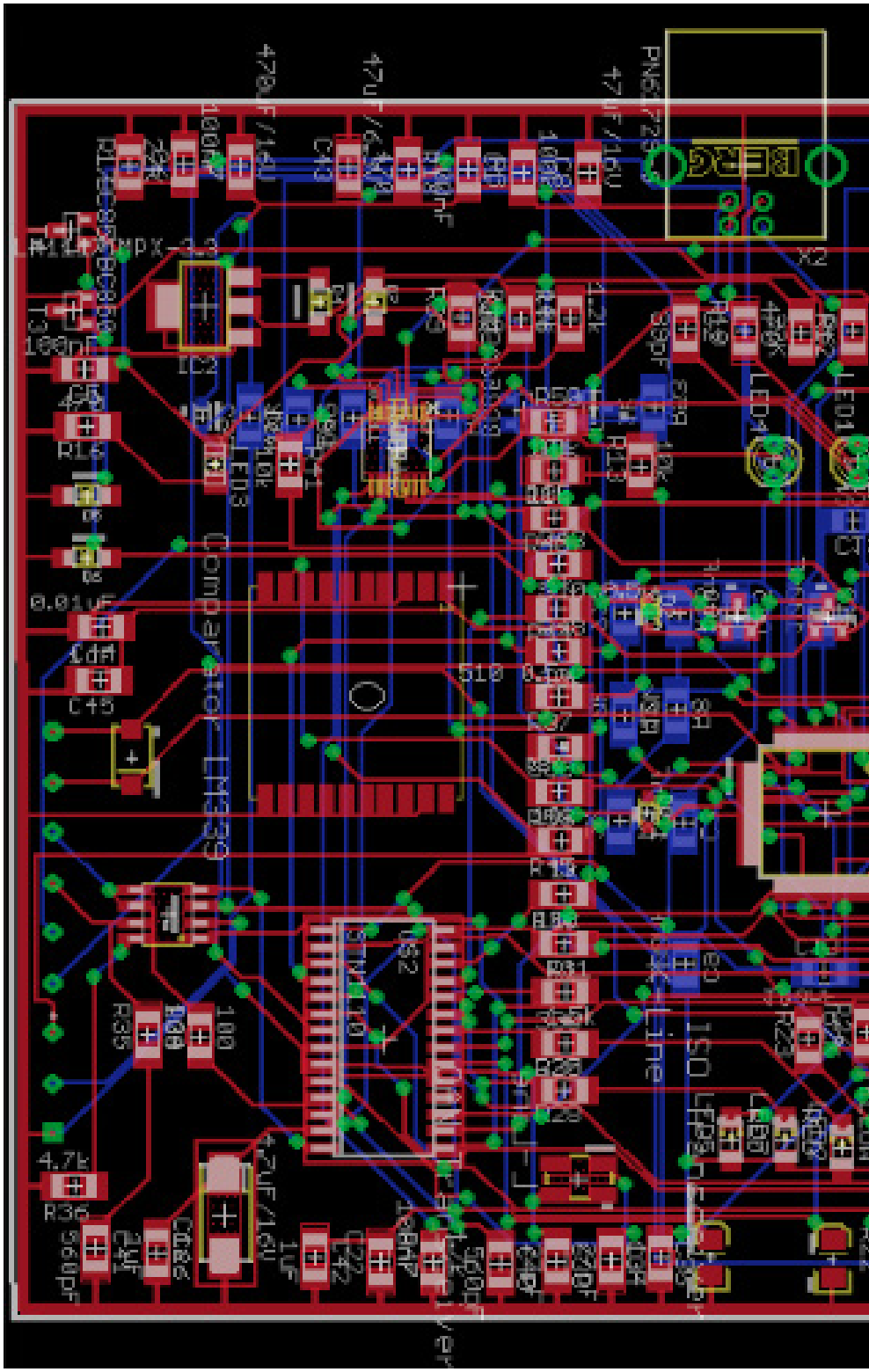
LPC2148

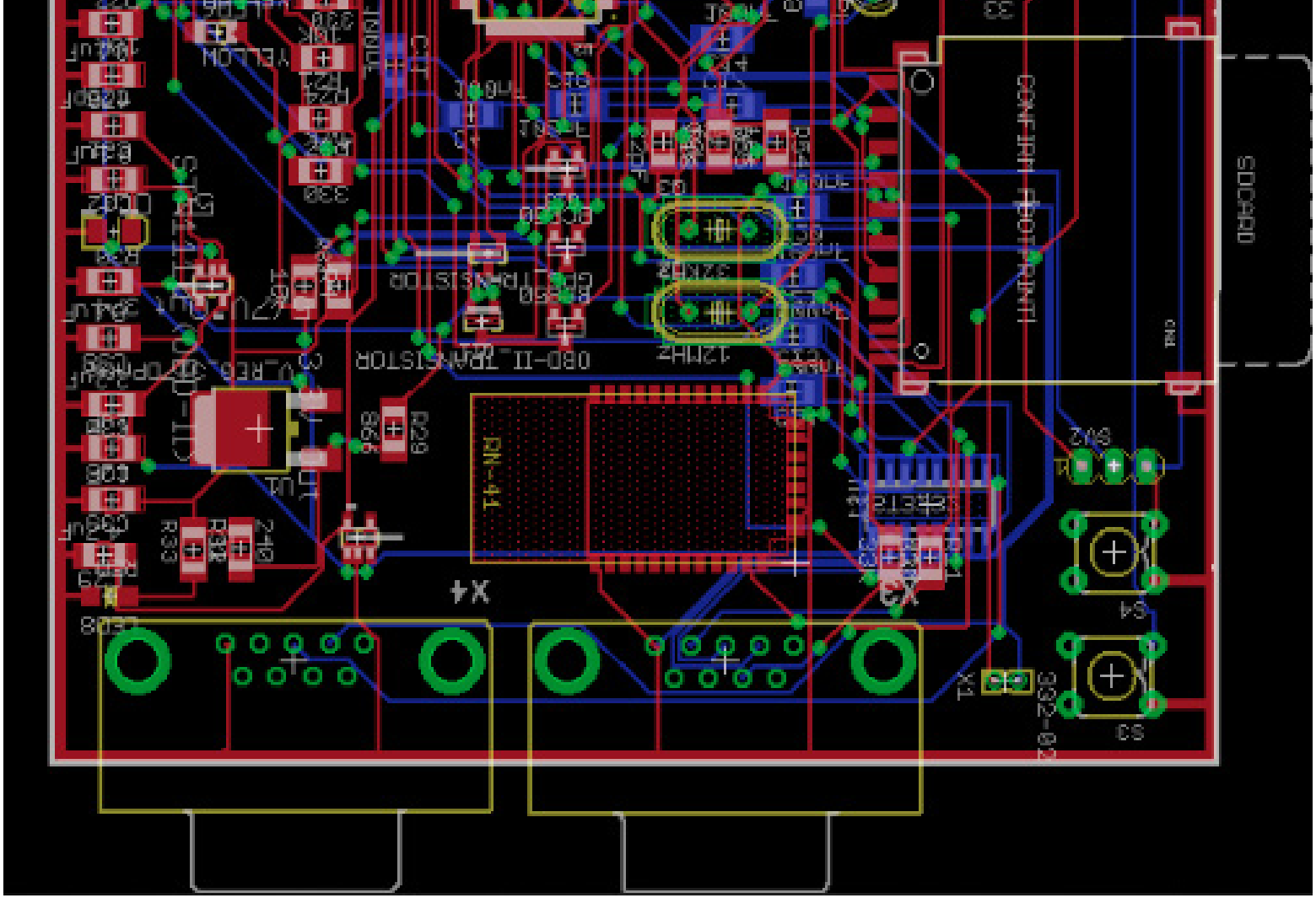

MICROCONTROLLER CIRCUIT
U2
P0.0/TXD0/PWM1
P0.1/RXD0/PWM3/EINT0
P0.2/SCL0/CAP0.0
P0.3/SDA0/MAT0.0/EINT1
P0.4/SCK0/CAP0.1/AD0.6
P0.5/MISO0/MAT0.1/AD0.7
P0.6/MOSI0/CAP0.2/AD1.0
P0.7/SSEL0/PWM2/EINT2
P0.8/TXD1/PWM4/AD1.1
P0.9/RXD1/PWM6/EINT3
P0.10/RTS1/CAP1.0/AD1.2
P0.11/CTS1/CAP1.1/SCL1
P0.12/DSR1/MAT1.0/AD1.3
P0.13/DTR1/MAT1.1/AD1.4
P0.14/DCD1/EINT1/SDA1
P0.15/RI1/EINT2/AD1.5
P0.16/EINT0/MAT0.2/CAP0.2
P0.17/CAP1.2/SCK1/MAT1.2
P0.18/CAP1.3/MISO1/MAT1.3
P0.19/MAT1.2/MOSI1/CAP1.2
P0.20/MAT1.3/SSEL1/EINT3
P0.21/PWM5/AD1.6/CAP1.3
P0.22/AD1.7/CAP0.0/MAT0.0
P0.23/VBUS
RESET
P0.25/AD0.4/AOUT
D+
D-
P0.28/AD0.1/CAP0.2/MAT0.2
P0.29/AD0.2/CAP0.3/MAT0.3
P0.30/AD0.3/EINT3/CAP0.0
P0.31/UP_LED/CONNECT
P1.31/TRST
P1.30/TMS
P1.29/TCK
P1.28/TDI
P1.27/TDO
P1.26/RTCK
P1.25/EXTIN0
P1.24/TRACECLK
P1.23/PIPESTAT2
P1.22/PIPESTAT1
P1.21/PIPESTAT0
P1.20/TRACESYNC
P1.19/TRACEPKT3
P1.18/TRACEPKT2
P1.17/TRACEPKT1
P1.16/TRACEPKT0
VBAT
VREF
VDDA
VSSA
RTXC2
RTXC1
XTAL2
XTAL1
VDD
VDD
VDD
VSS
VSS
VSS
VSS
VSS
LPC2148
TX0
RX0
OBD-II_CTRL
GPS_CTRL
TX1
RX1
BSL
SCK1
MISO1
MOSI1
SSEL1
USB_5_1
RST#
USB_DP_1
USB_DM_1
MMC_CP
MMC_WP
CONNECT
+3V3
R1
10k
R2
10k
RTXC1
RTXC2
X2
X1
GND
332-02
X1-2
X1-1

GPS/BlueTooth/SD



FINAL PCB


Comparator LM339
ISO Line
0.01uF
4.7k
R36
100
R35
470uF/16V
47uF/10V
100nF
R16
R17
R18
R19
R20
R21
R22
R23
R24
R25
R26
R27
R28
R29
R30
R31
R32
R33
R34
R35
R36
C43
C44
C45
C46
C47
C48
C49
C50
C51
C52
C53
C54
C55
C56
C57
C58
C59
C60
C61
C62
C63
C64
C65
C66
C67
C68
C69
C70
C71
C72
C73
C74
C75
C76
C77
C78
C79
C80
C81
C82
C83
C84
C85
C86
C87
C88
C89
C90
C91
C92
C93
C94
C95
C96
C97
C98
C99
C100
C101
C102
C103
C104
C105
C106
C107
C108
C109
C110
C111
C112
C113
C114
C115
C116
C117
C118
C119
C120
C121
C122
C123
C124
C125
C126
C127
C128
C129
C130
C131
C132
C133
C134
C135
C136
C137
C138
C139
C140
C141
C142
C143
C144
C145
C146
C147
C148
C149
C150
C151
C152
C153
C154
C155
C156
C157
C158
C159
C160
C161
C162
C163
C164
C165
C166
C167
C168
C169
C170
C171
C172
C173
C174
C175
C176
C177
C178
C179
C180
C181
C182
C183
C184
C185
C186
C187
C188
C189
C190
C191
C192
C193
C194
C195
C196
C197
C198
C199
C200
C201
C202
C203
C204
C205
C206
C207
C208
C209
C210
C211
C212
C213
C214
C215
C216
C217
C218
C219
C220
C221
C222
C223
C224
C225
C226
C227
C228
C229
C230
C231
C232
C233
C234
C235
C236
C237
C238
C239
C240
C241
C242
C243
C244
C245
C246
C247
C248
C249
C250
C251
C252
C253
C254
C255
C256
C257
C258
C259
C260
C261
C262
C263
C264
C265
C266
C267
C268
C269
C270
C271
C272
C273
C274
C275
C276
C277
C278
C279
C280
C281
C282
C283
C284
C285
C286
C287
C288
C289
C290
C291
C292
C293
C294
C295
C296
C297
C298
C299
C300
C301
C302
C303
C304
C305
C306
C307
C308
C309
C310
C311
C312
C313
C314
C315
C316
C317
C318
C319
C320
C321
C322
C323
C324
C325
C326
C327
C328
C329
C330
C331
C332
C333
C334
C335
C336
C337
C338
C339
C340
C341
C342
C343
C344
C345
C346
C347
C348
C349
C350
C351
C352
C353
C354
C355
C356
C357
C358
C359
C360
C361
C362
C363
C364
C365
C366
C367
C368
C369
C370
C371
C372
C373
C374
C375
C376
C377
C378
C379
C380
C381
C382
C383
C384
C385
C386
C387
C388
C389
C390
C391
C392
C393
C394
C395
C396
C397
C398
C399
C400
C401
C402
C403
C404
C405
C406
C407
C408
C409
C410
C411
C412
C413
C414
C415
C416
C417
C418
C419
C420
C421
C422
C423
C424
C425
C426
C427
C428
C429
C430
C431
C432
C433
C434
C435
C436
C437
C438
C439
C440
C441
C442
C443
C444
C445
C446
C447
C448
C449
C450
C451
C452
C453
C454
C455
C456
C457
C458
C459
C460
C461
C462
C463
C464
C465
C466
C467
C468
C469
C470
C471
C472
C473
C474
C475
C476
C477
C478
C479
C480
C481
C482
C483
C484
C485
C486
C487
C488
C489
C490
C491
C492
C493
C494
C495
C496
C497
C498
C499
C500
C501
C502
C503
C504
C505
C506
C507
C508
C509
C510
C511
C512
C513
C514
C515
C516
C517
C518
C519
C520
C521
C522
C523
C524
C525
C526
C527
C528
C529
C530
C531
C532
C533
C534
C535
C536
C537
C538
C539
C540
C541
C542
C543
C544
C545
C546
C547
C548
C549
C550
C551
C552
C553
C554
C555
C556
C557
C558
C559
C560
C561
C562
C563
C564
C565
C566
C567
C568
C569
C570
C571
C572
C573
C574
C575
C576
C577
C578
C579
C580
C581
C582
C583
C584
C585
C586
C587
C588
C589
C590
C591
C592
C593
C594
C595
C596
C597
C598
C599
C600
C601
C602
C603
C604
C605
C606
C607
C608
C609
C610
C611
C612
C613
C614
C615
C616
C617
C618
C619
C620
C621
C622
C623
C624
C625
C626
C627
C628
C629
C630
C631
C632
C633
C634
C635
C636
C637
C638
C639
C640
C641
C642
C643
C644
C645
C646
C647
C648
C649
C650
C651
C652
C653
C654
C655
C656
C657
C658
C659
C660
C661
C662
C663
C664
C665
C666
C667
C668
C669
C670
C671
C672
C673
C674
C675
C676
C677
C678
C679
C680
C681
C682
C683
C684
C685
C686
C687
C688
C689
C690
C691
C692
C693
C694
C695
C696
C697
C698
C699
C700
C701
C702
C703
C704
C705
C706
C707
C708
C709
C710
C711
C712
C713
C714
C715
C716
C717
C718
C719
C720
C721
C722
C723
C724
C725
C726
C727
C728
C729
C730
C731
C732
C733
C734
C735
C736
C737
C738
C739
C740
C741
C742
C743
C744
C745
C746
C747
C748
C749
C750
C751
C752
C753
C754
C755
C756
C757
C758
C759
C760
C761
C762
C763
C764
C765
C766
C767
C768
C769
C770
C771
C772
C773
C774
C775
C776
C777
C778
C779
C780
C781
C782
C783
C784
C785
C786
C787
C788
C789
C790
C791
C792
C793
C794
C795
C796
C797
C798
C799
C800
C801
C802
C803
C804
C805
C806
C807
C808
C809
C810
C811
C812
C813
C814
C815
C816
C817
C818
C819
C820
C821
C822
C823
C824
C825
C826
C827
C828
C829
C830
C831
C832
C833
C834
C835
C836
C837
C838
C839
C840
C841
C842
C843
C844
C845
C846
C847
C848
C849
C850
C851
C852
C853
C854
C855
C856
C857
C858
C859
C860
C861
C862
C863
C864
C865
C866
C867
C868
C869
C870
C871
C872
C873
C874
C875
C876
C877
C878
C879
C880
C881
C882
C883
C884
C885
C886
C887
C888
C889
C890
C891
C892
C893
C894
C895
C896
C897
C898
C899
C900
C901
C902
C903
C904
C905
C906
C907
C908
C909
C910
C911
C912
C913
C914
C915
C916
C917
C918
C919
C920
C921
C922
C923
C924
C925
C926
C927
C928
C929
C930
C931
C932
C933
C934
C935
C936
C937
C938
C939
C940
C941
C942
C943
C944
C945
C946
C947
C948
C949
C950
C951
C952
C953
C954
C955
C956
C957
C958
C959
C960
C961
C962
C963
C964
C965
C966
C967
C968
C969
C970
C971
C972
C973
C974
C975
C976
C977
C978
C979
C980
C981
C982
C983
C984
C985
C986
C987
C988
C989
C990
C991
C992
C993
C994
C995
C996
C997
C998
C999
C1000


SDCARD
12MHZ
OBD-II_TRANSISTOR
TRANSISTOR
CAN-44
SW2-03
SW2
S4
S3
X4
X3

TEAM ALWAYS BLOCK

www.teamalwaysblock.com