2011

Team Cool Beans

Poor Man’s LoJack

Our solution to stolen cars and short range vehicle security systems

Bryon Wheeler	konkop@gmail.com
Philip Krebs	u0321860@gmail.com
Gregory Beck	saigon.delta@gmail.com
Simon Chulin	simon.chulin@gmail.com

Project Website:
www.GodImGood.com

Table of Contents

Team Cool Beans Poor Man’s LoJack

1.1 Introduction

Our senior project will be the design and implementation of a low cost, LoJack type car communication system. Our goal is to put together a two part system that will implement the vehicle tracking ability of the original LoJack system, along with some additional features including: a remote start, a remote door lock/unlock feature, and remote vehicle monitoring system. All of these features will be controlled or displayed on an Android phone used by the owner of the car. An Arduino microcontroller will provide the vehicle based computing necessary to achieve this functionality.

1.2 Motivation

The motivation for re-designing a LoJack type system is three fold. While LoJack requires a monthly service fee, our implementation will allow the use of any GSM based wireless provider. In the United States this would mean being able to use one of T-Mobile’s or AT&T’s prepaid or contract plans for a much lesser value. Android phones are also becoming increasingly popular and while most new cars have remote features they have a limited range usually within only 100 feet or less. Because the Poor Man’s LoJack System is GSM based, range is only limited by the coverage of the provider on both the vehicle and Android phone. Additionally we will be implementing extra features that LoJack does not currently have.

1.3 Technical Description

In order for this system to work we are going to build an embedded system for the car using an Arduino board and an Arduino shield card that privides a cellular interface, meaning that you would simply need to purchase a sim card and install it in the device to begin communication between the Android platform and the car. The Arduino will use NMEA 2.0 GPS location strings from an rs232 GPS receiver to determine the car’s location, it will then package the GPS data, OBDII data, and door pin status and send all that to the Android device. The Android device will receive the data packets and display them to the GUI to be displayed in an understandable format to any user. The Android will package the command headed for the car and send them to the car based platform where it will be parsed and used to trigger different pins on the Arduino board to change the boards logical state. The pins attached to these command lines will be connected to a circuit card where solid state relays will then be used to lock or unlock the doors and start the car.

Power adapters and converters will need to be devised to power each of the components off of 12VDC.

1.4 Required Functionality for Demo Day

Remote GPS vehicle tracking using Google Maps

Remote engine start
Remote lock and unlock
Basic security door open status
OBDII remote data monitoring (rpm, speed, coolant temp)

1.5 Optional Functionality Time Allowing

GPS Finding the car in the parking lot using Google Maps
Security between the Android and Car platforms

2 Hardware Design

This section describes the hardware implementation of our LoJack system.

2.1 Hardware Description

The bulk of our analog hardware design will be in the OBDII interface and relay driver board. Our digital hardware will mostly be COTS items.

2.3 Hardware Pieces

Our hardware implementation will contain the following pieces where work will need to be done by the group.

Relay Bank - automotive grade relays wired to drive the starter, and door locks.

Relay Driver - circuit that will process TTL signals from the Arduino as well as door pin signals from the car.

Arduino/Cellular Shield - will need to be wired to send and receive from other hardware pieces.

OBDII interface - will provide the voltage level and logical conversion between the car and serial port on the Arduino.

GPS Antenna – will provide NMEA 2.0 GPS strings to the Arduino serial port at a given baud rate.

Hardware Power Requirements – will need to be addressed such that the incorporated hardware will run off 12 Volts.

2.3.1 Relay Bank

A bank of 3 SPDT 30A relays will be used to start the car, lock the car doors, and unlock the car doors. Wiring should be fairly straight forward here.

2.3.2 Relay Driver

The purpose of this circuit will be to amplify the signals from the Arduino output's to drive the relays. As such the driver board will need to be diode protected so that back EMF does not ruin the driver circuitry. This circuit will also be used to condition any inputs to the Arduino to TTL levels.

2.3.3 Arduino/Cellular Shield

The Arduino, Cellular Shield, and GSM module will be bought. An antenna of the correct band will need to be bought and attached to the GSM unit. Integrating the Arduino into the vehicle will be fairly straightforward. Voltage regulation will be key as we are told the GSM module is sensitive to voltage variation. Voltage regulation is planned to be handled by the cellular shield this however may not be enough.

2.3.4 OBDII interface

The vehicle OBDII port operates at +12VDC logic levels. A voltage level conversion will have to be done to interface the Arduino to the OBDII interface. The plan is to use an OBDII interpreter chip to handle the voltage level translation. Also since our vehicle uses pulse width modulation interfacing to the OBDII port with nothing more than a serial port is not possible. Therefore the OBDII interpreter will provide level conversion and a UART interface to talk to the Arduino serial port.

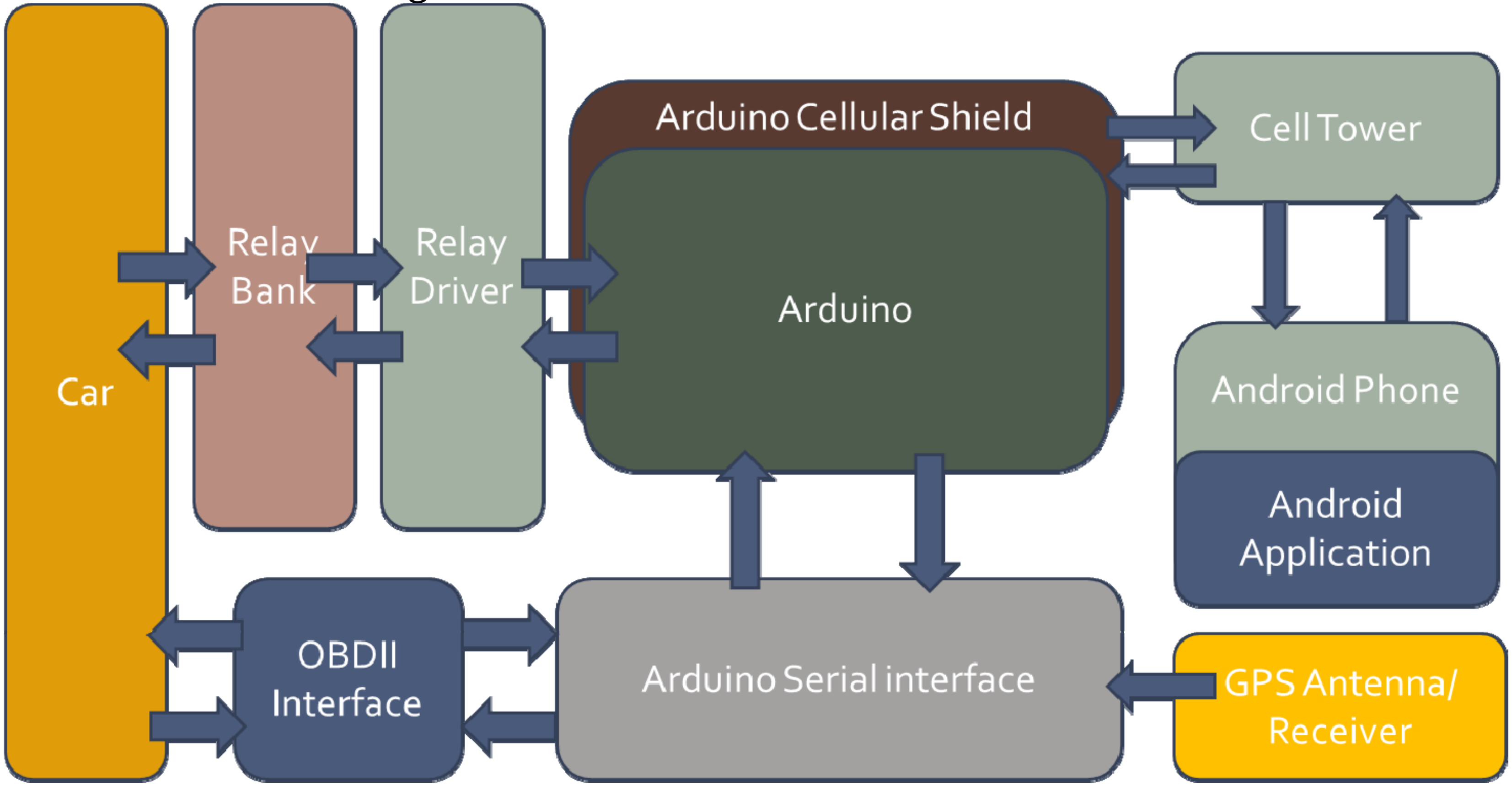
2.3.5 GPS Receiver

An rs232 serial GPS receiver will be used to output NMEA 2.0 location strings to an Arduino serial port. The GPS will need to be externally powered and will most likely be the Garmin 16xHVS.

2.3.6 Hardware Power Requirements

Hardware Element	Current Draw	Native Voltage	Power
Relay Bank	N/A(only when fired)	12V	-
Relay Driver	200mA	12V	2.4W
Arduino/Cellular Shield	600mA	5V	3W
OBDII interface	N/A(OBDII powered)	N/A(OBDII powered)	-
GPS Antenna	40mA	12V	0.48W
Totals	-	-	~6W

2.4 Hardware Block Diagram



3 Software Design

This section describes the software implementation of our LoJack system.

3.1 Description

A lot of our software design work will be designing a message interchange format between the Arduino and Android platforms. Our software will also need to take apart a few different message types from the OBDII port and GPS.

3.2 Software Pieces

Our software implementation will contain the following pieces where work will need to be done by the group.

Arduino LoJack MSG Based Controller – will be the main controller piece for the car based platform.

GPS Interface – will intercept messages coming from the serial ports and feed messages to the MSG controller.

OBDII Interface – will act as the go between the OBDII Interface hardware and the MSG controller.

Arduino TCP/IP Interface – responsible building and disassembling packets of data between Arduino TCP/IP Library and the Arduino Based MSG Controller.

Android TCP/IP Interface – responsible for building and disassembling packets of data between Android TCP/IP Library and the Android based MSG Controller.

Android Lo Jack MSG Based Controller – will be the main controller piece for the android based platform.

Android GUI – will be the piece that the user sees and interacts with to provide the LoJack services.

3.2.1 Arduino LoJack MSG Based Controller

The Arduino MSG based controller will accept and send messages between different send and receive threads. The controller will need to time the starting of the car and send signals to the control pins.

3.2.2 GPS Interface

This module will need to take apart the NMEA 2.0 strings and send them to the Android device.

3.2.3 OBDII Interface

This module will need to transact between the PID’s in the ECU of the vehicle and requests from the controller.

3.2.4 Arduino TCP/IP Interface

This piece will be responsible for building and disassembling packets of data between Arduino TCP/IP Library and the Arduino Based MSG Controller.

3.2.5 Android TCP/IP Interface

Responsible for building and disassembling packets of data between Android TCP/IP Library and the Android based MSG Controller.

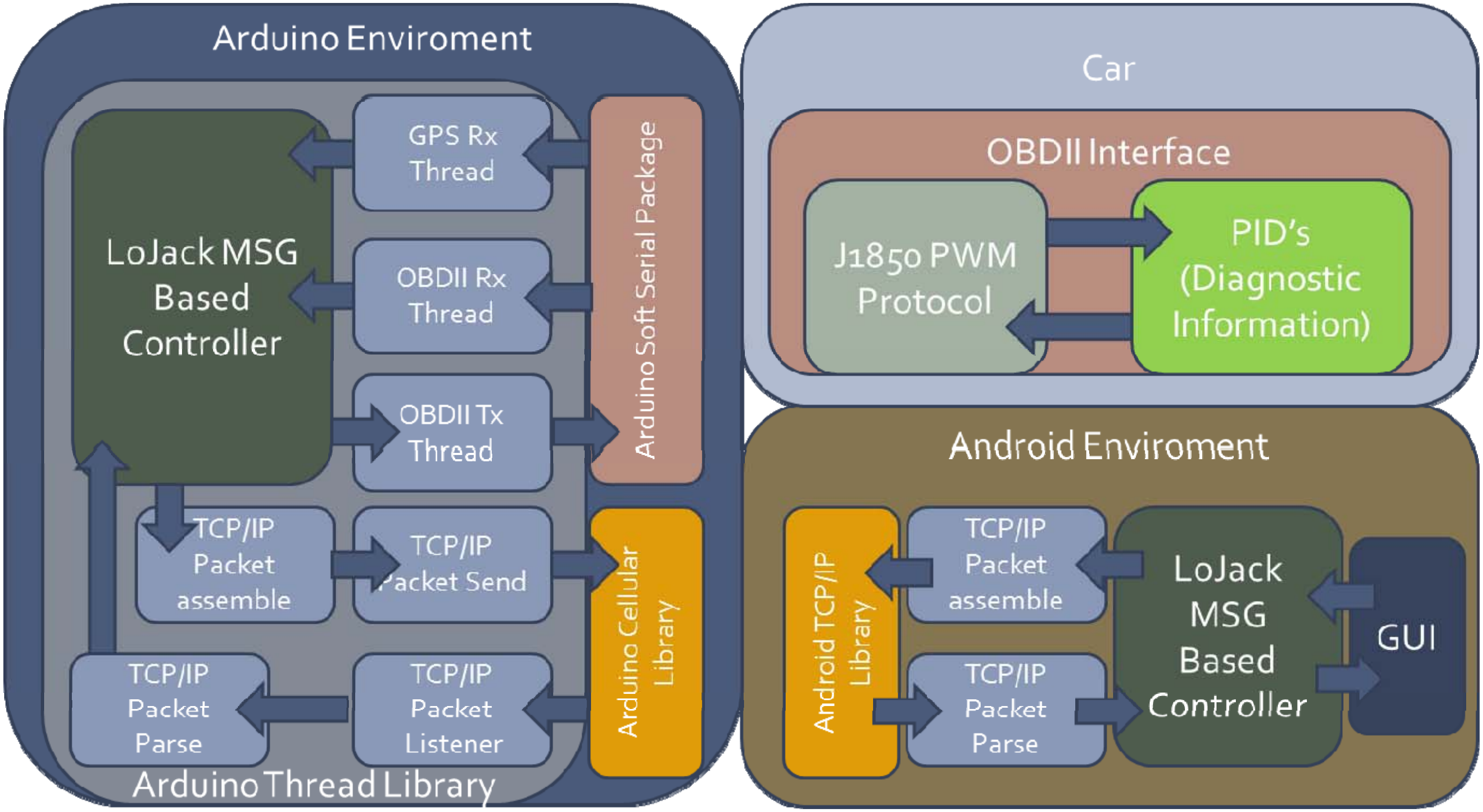
3.2.6 Android Lo Jack MSG Based Controller

This module will be the main controller piece for the android based platform.

3.2.7 Android GUI

Will be the piece that the user sees and interacts with to provide the LoJack services

3.3 Hardware Block Diagram



4 Schedule

Team Cool Beans

Today's Date: 4/4/2011 Monday

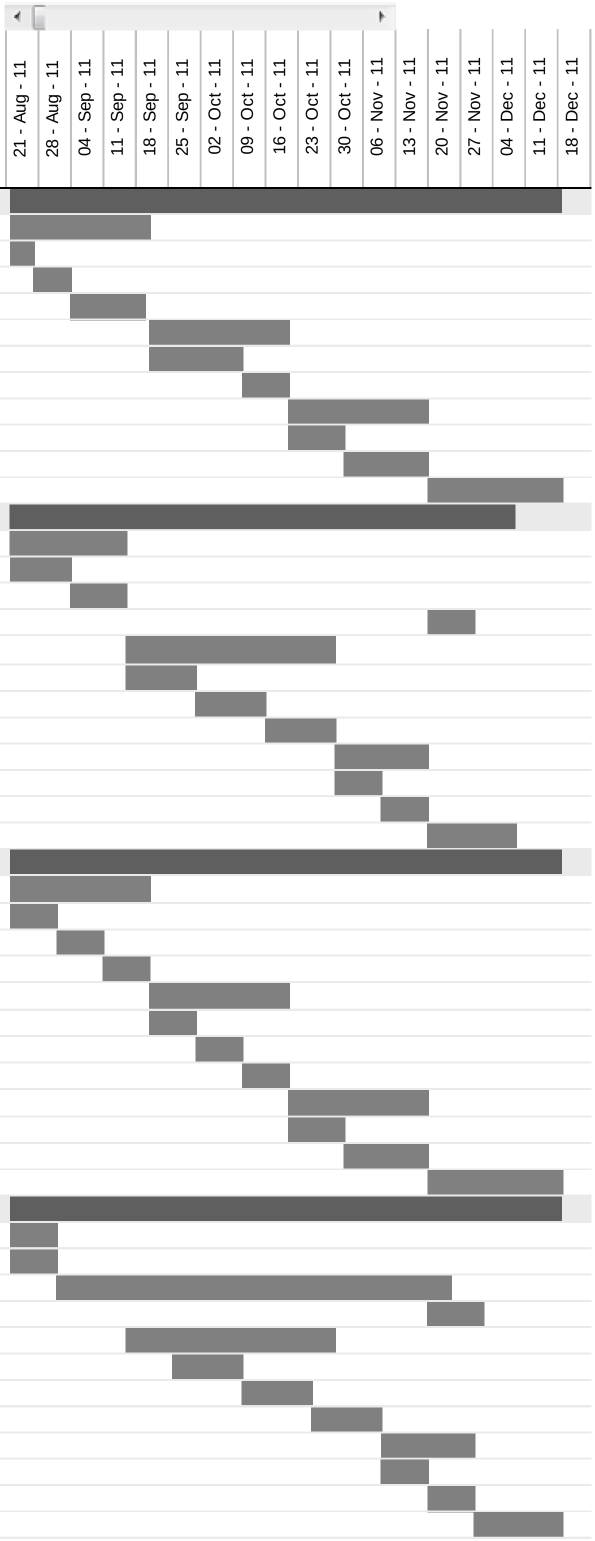
(vertical red line)

Project Lead: Team Cool Beans

Start Date: 8/22/2011 Monday

First Day of Week (Mon=2): 1

WBS	Tasks	Task Lead	Start	End	Duration (Days)	% Complete	Working Days	Days Complete	Days Remaining
1	Car Interface	Philip	8/22/11	12/18/11	119	0%	85	0	119
1.1	Relay Bank for Car		8/22/11	9/20/11	30	0%	22	0	30
1.1.1	Purchase Relays		8/22/11	8/26/11	5	0%	5	0	5
1.1.2	Install Relay's In Car		8/27/11	9/03/11	8	0%	5	0	8
1.1.3	Build/Test Relay Driver		9/04/11	9/19/11	16	0%	11	0	16
1.2	Complete Power and Interface		9/21/11	10/20/11	30	0%	22	0	30
1.2.1	Stable 5V Arduino Power.		9/21/11	10/10/11	20	0%	14	0	20
1.2.2	Stable GPS Power		10/11/11	10/20/11	10	0%	8	0	10
1.3	Arduino OBDII Interfacing		10/21/11	11/19/11	30	0%	21	0	30
1.3.1	Build/Test OBDII Interface		10/21/11	11/01/11	12	0%	8	0	12
1.3.2	Inst Arduio and get to work		11/02/11	11/19/11	18	0%	13	0	18
1.4	Finalize		11/20/11	12/18/11	29	0%	20	0	29
2	Android	Greg	8/22/11	12/08/11	109	0%	79	0	109
2.1	Basic android App		8/22/11	9/15/11	25	0%	19	0	25
2.1.1	Android Research		8/22/11	9/03/11	13	0%	10	0	13
2.1.2	GUI Functionality		9/04/11	9/15/11	12	0%	9	0	12
2.1.3	Polish GUI w/ Simon		11/20/11	11/29/11	10	0%	7	0	10
2.2	Connect Android & Arduino		9/16/11	10/30/11	45	0%	31	0	45
2.2.1	Research Network Protocol		9/16/11	9/30/11	15	0%	11	0	15
2.2.2	IP Cell & Comp /w Simon		10/01/11	10/15/11	15	0%	10	0	15
2.2.3	IP cell & Arduino /w Simon		10/16/11	10/30/11	15	0%	10	0	15
2.3	Full Testing of Arduino & Car		10/31/11	11/19/11	20	0%	15	0	20
2.3.1	Arduio Talking /w Bryon		10/31/11	11/09/11	10	0%	8	0	10
2.3.2	Finalize Documentation		11/10/11	11/19/11	10	0%	7	0	10
2.4	Finalize		11/20/11	12/08/11	19	0%	14	0	19
3	Arduino	Bryon	8/22/11	12/18/11	119	0%	85	0	119
3.1	Basic Arduino Code		8/22/11	9/20/11	30	0%	22	0	30
3.1.1	Research Arduino		8/22/11	8/31/11	10	0%	8	0	10
3.1.2	Build Threads		9/01/11	9/10/11	10	0%	7	0	10
3.1.3	Setup Serial Interface		9/11/11	9/20/11	10	0%	7	0	10
3.2	Connect Cell & Arduino		9/21/11	10/20/11	30	0%	22	0	30
3.2.1	Communcate with Chip		9/21/11	9/30/11	10	0%	8	0	10
3.2.2	Setup IP to Computer		10/01/11	10/10/11	10	0%	6	0	10
3.2.3	Connect Android & Arduino		10/11/11	10/20/11	10	0%	8	0	10
3.3	Full testing of arduino & car		10/21/11	11/19/11	30	0%	21	0	30
3.3.1	Write code for relays		10/21/11	11/01/11	12	0%	8	0	12
3.3.2	Test on relay with phil		11/02/11	11/19/11	18	0%	13	0	18
3.4	Finalize		11/20/11	12/18/11	29	0%	20	0	29
4	Communications	Simon	8/22/11	12/18/11	119	0%	85	0	119
4.1	Purchase Parts & Interface		8/22/11	8/31/11	10	0%	8	0	10
4.1.1	Purchase All parts		8/22/11	8/31/11	10	0%	8	0	10
4.1.2	Design Cell Images		9/01/11	11/24/11	85	0%	61	0	85
4.1.3	Polish GUI		11/20/11	12/01/11	12	0%	9	0	12
4.2	Network Communications		9/16/11	10/30/11	45	0%	31	0	45
4.2.1	Research Network Protocol		9/26/11	10/10/11	15	0%	11	0	15
4.2.2	Cell to Comp. over IP		10/11/11	10/25/11	15	0%	11	0	15
4.2.3	Cell to Arduino over IP		10/26/11	11/09/11	15	0%	11	0	15
4.3	Documentation		11/10/11	11/29/11	20	0%	14	0	20
4.3.1	Collect Documentation		11/10/11	11/19/11	10	0%	7	0	10
4.3.2	Help Everyone Finish		11/20/11	11/29/11	10	0%	7	0	10
4.4	Finalize		11/30/11	12/18/11	19	0%	13	0	19



5 Testing and Integration (Milestones from schedule detailed)

5.1 Car

5.1.1 Car Interface

The completion of the car interface will follow the schedule. Components will be designed/built/tested in the following order. First relay wiring, next relay driver circuit, and finally OBDII interface. It is not our opinion that leaving the OBDII interface for last introduces any additional risk (work hard stuff first) as the first two components should be easy to build.

5.1.1.1 Purchase Relays

Buy (2) SPDT 30 amp automotive style relays. These are available at any automotive parts house or junk yard.

5.1.1.2 Install Relays in Car

The relays purchased will be installed in the car and tested manually to trigger ignition on and starter function.

5.1.1.3 Build/Test Relay Driver Circuit

The relay driver circuit will be designed built and tested using an Arduino to test trigger a relay not necessarily attached to the car. A bench test should be sufficient until integration can be performed.

5.1.2 Complete Power and Interface

The power requirements for the devices have been assessed and stable power supplies will need to be acquired.

5.1.2.1 Stable 5V Arduino Power

The GSM module attached to our cellular shield for the Arduino is very sensitive to power fluctuations a stable 12V to 5v power supply will need to be sourced.

5.1.2.2 GPS Power

The Garmin GPS we intend to use is capable of accepting 12VDC. It will need to be wired into the 12V of the car and may need additional voltage stabilization.

5.1.3 Arduino OBDII Interfacing

The Arduino will need to know how to signal the OBDII port and receive the response. Software to do this will need to be written. The circuit to perform the electrical translation between the Arduino serial port and OBDII port will need to be designed and built.

5.1.3.1 Build/Test OBDII interface

The OBDII circuit will be first tested with a serial port on a computer to ensure that the messages are being sent and received properly by the interface. Then the circuit can be tested with the Arduino serial port with some assurance that the circuit is working.

5.1.3.2 Install Arduino and get to work

Once the other members of the teams get their part’s working and we are able to send messages over the cellular tower we should be able to install the Arduino in the car and test for functionality.

5.1.4 Finalize

Polish up any interfaces that the user will see and work any bugs that will crop up.

5.2 Android

5.2.1 Basic android App

The Basic android application will feature a polished user interface and correctly interpret commands from the user to be sent across the network. To accomplish this we will need to research Android programming methods, create control software, and interface the control software with a polished GUI.

5.2.1.1 Figure out Android

Research into Android programming to implement GUI, controls and TCP/IP communication.

5.2.1.2 Application Prototype

A basic application with mock-up interface that will read button presses, correctly interpret the commands and prepare the commands for transmission over the network.

5.2.1.3 Finished Application Interface

Replace mock-up interface with polished design and insure commands are correctly read from user.

5.2.2 Android to Arduino Communication

In this stage we will demonstrate two-way communication between the Arduino and the Android Phone across the cellular network.

5.2.2.1 Implement Protocol on Android

Demonstrate correct communication output from Android Phone.

5.2.2.2 Communication Over Network

Demonstrate communication between Android and Computer across Cell Network.

5.2.2.3 Communication With Android

Demonstrate communication between Android and Arduino across Cell Network.

5.2.3 Finalize Interface between Android and Arduino

Fully functional implementation of user input being properly translated to the Arduino output signals.

5.2.3.1 Functional Interface between Arduino and Android

Demonstrate correct interpretation of commands between Android and Arduino across Cell Network

5.2.3.2 Help with documentation

Finalize documentation of Android Application

5.2.4 Fully Functional Integration

Demonstrate a fully functional system that meets all specifications.

5.3 Arduino

5.3.1 Basic Arduino Code

The basic Arduino setup will go as follows. First since Bryon has never worked on an Arduino before he should build a basic Arduino circuit that can turn on leds and other fun functionality. Next a simple threading system should be build that will do something through threads. Lastly, a serial interface should be setup that will allow the Arduino to communicate with a serial device.

5.3.1.1 Figure out the Arduino

An Arduino board should be borrowed from Eric Brunvand. From there a system should be setup that will allow for the turning on and off of LEDs. The code should be formed that will allow for the testing of these LEDs.

5.3.1.2 Setup the Threads

Threads should be setup with a basic scheduler. The scheduler will need to be played with later to find the one that will function the most efficiently. This scheduler should be setup to turn on and off the LEDs to identify the switching of threads

5.3.1.3 Setup the Serial Interface

The multi-serial communication should be setup. This should set-up 3 different ports to communicate with another device. This can be tested using a computer and hyper terminal.

5.3.2 Connect the cell phone and the Arduino

The connection of the cell phone and Arduino should happen in the following fashion. First a serial communication that was setup in the last task should be setup to communicate with the cell chip as per the specifications. Set it up to where the cell chip will communicate with a computer. Lastly set up the cell chip to communicate with the android phone.

5.3.2.1 Communicate with chip

Following the specifications, from the cell chips design document, setup the cell chip communication to a serial port. This communication can be tested by sending text messages to a cell phone.

5.3.2.2 Setup IP to computer

Next setup a connection that can be transmitted to the IP address of a computer. This means that the packets will need to be setup so that a specific IP address can read them.

5.3.2.3 Connect Android & Arduino

Now the Android should be setup to accept data packets from the Arduino. Work with Simon and Greg to get the packets transferring both from and to the Android correctly.

5.3.3 Full testing of Arduino & car

The Arduino will now need to be hooked up to the car. It should be able to accept simple messages from the Android and turn on the relays. It should also be able to send messages back to the Android on current status of the car.

5.3.3.1 Write code for relays

The code should now be able to take simple messages and make changes to the relays. The code should also be able to collect data serially from the car and pass it back to the Android. A test function should be built that can be activated to test all of the relays and the receiving of information from the ODBII system. This is so that the device can be tested without the Android present.

5.3.3.2 Test on relay and ODBII with Phil

All bugs in any of the relay or ODBII communications should be fixed at this point and time. The device at this point should allow for the car and Arduino to do full communications. Also the test code from the last module can be used for testing when an Android is not available.

5.3.4 Getting things to work

Polish up any bugs or issues that may pop-up. If needed help any other class mates with efforts they have remaining.

5.4 Communications

We will need to find out how to communicate via a cellular network using an Arduino board and an android cell phone. Hopefully the network traffic will allow for push notifications to the car, if not we may have to use a text to wake strategy.

5.4.1 Purchase Parts & construct interface

In order to be user friendly our project will need to have a clean and intuitive interface to run the custom

Components as described below.

5.4.1.1 Purchase All parts

Use the shopping list on the back of this paper to purchase all the necessary parts and components, check the Arduino board out from the University, and get everything ready for assembly.

5.4.1.2 Design Cell Images

Design a user friendly and intuitive graphical interface for the cell phone, that would ultimately allow users to click on a graphic and have their car respond accordingly.

5.4.1.3 Work with Greg on interface

The last part would be to make the graphical user interface connect with the back end software to function **a**s planned.

5.4.2 Design code of cell to car

Creating the back end software to run on the android phone will definitely be a big part of this project

And we expect this to be the biggest hurdle, but we plan on using push notifications to alert the arduino when a function is requested.

5.4.2.1 Figure out network

As it stands, interfacing through a cellular network is currently somewhat of a mystery since no one in our group has done so before, we plan to search the web and AT&T’s developer site for answers to these questions.

5.4.2.2 IP cell and computer

Communications between the cellular network and android cell phone shouldn’t be too difficult, as the phone is already connected to the network by default, we hope to just use the built in network interface libraries to accomplish the communications required.

5.4.2.3 IP cell and arduino

Communicating between the Cellular network and the arduino board will be a challenge. First off, getting the board to be accepted by the network may be a little tricky on its own, once that is done though, the use of push notifications will be a hurdle as well. Luckily there is a good amount of documentation available on the internet describing how to perform these tasks, so we plan to use that as a reference.

5.4.3 Documentation

Since documentation is an important part of every project, we plan to document every big achievement as it is made, and uploading that progress to our website, to be assembled and printed at the end of the
semester.

5.4.3.1 Manage documentation

We hope to use the team website as a repository for us to upload documents, so they can be viewed by the whole team, and function as a marker of progress for everyone else.

5.4.3.2 Help everyone complete it

Hopefully group meetings will function as a good way of motivating people to write up the documentation as they go, if anyone needs any help with something they're not sure about, we will decide on the desired answer as a group to keep the project moving along.

5.4.4 Getting things to work

Perhaps the most important part will be to get everything working the way it should, we plan on working as a team to get this project finished, allowing all of us to come to each other for help if there is something we are absolutely stuck on.

6 Risk Mitigation

The risks that Bryon may encounter are limitations found within the cell Arduino. While coding the device cellular specific code and other code should be made modular. In the case that the memory or shortage of pins becomes an issue the following tasks need to take action. If this occurs we will need to purchase the Arduino mega 2560 which will act as an expansion port. The moving of modules over to the mega 2560 should then be done. At this point Greg will need to help Bryon in catching up to the tasks found.

There will be issues with the Arduino trying to get so many things running on a single board. To fix this we will be using threads to control each of the different reading and writing parts of the board. We will also need to incorporate some thread controls to make sure that certain more important threads finish. The thread control also needs to make certain that it doesn't "choke" any other thread from being able to do its job.

Another risk we have is the Arduino only has a single serial communications port. To fix this we will need to use serial software and other pins to make software based serial communications to the other devices.

There may be issues getting the IP communication to work between the Arduino and the cell phone. If that become the case Simon will need to get text based communication working on the cell phone and Bryon will need to decrypt those messages on the Arduino board.

We will select Phil's Contour as our vehicle. There is no security to bypass and the ODB2 and other connections on the car are easily available. If the case happens that we fry circuitry on the car we will need to find an alternative backup or demo our project on something less expensive.

The CAN bus communication will only happen once all other components are working well. The CAN bus will need to be interfaced using a special interfaced designed by Phil. If at any point we run out of time or cannot get it to work we will revert back to the functioning version without CAN bus support.

There have been reported issues regarding the voltage regulator on the Arduino Cell Shield causing chip malfunction. As we understand it the GSM chip overheats from over voltage. This defect requires some more investigation but Phil will fix this by getting a good heat sink on the chip and also making sure that the voltage regulator is always within a safe range for the board.

There is also a worry that security may be an issue. To fix this we will need to work on a basic encryption method for the cell and Arduino that will encrypt the message based on a number that we will place on the Arduino. Also in this case Simon will need to make for the ability to enter both the phone number of the receiving Android in the interface as well as the android's secret encrypting number.

7 Bill of Materials:

Arduino Processor Card:

Free

Check out from the U of U

Arduino Cell Shield:

$99.95

http://www.sparkfun.com/products/9607

Cell Shield Antenna:

$6.99

http://www.cutedigi.com/product_info.php?cPath=242_263&products_id=4180

Android phone:

Free

Gregory Beck

Sim card:

$5.95

AT&T Store

Wires and cabling:

$10.00 ?

Warnock Engineering Building Supply Shop

Relays:

$8.99 each bank

http://www.radioshack.com/product/index.jsp?productId=2049722&CAWELAID=107596643

Lab Power Supply for testing:

Free

Engineering Lab at U of U

ODBII Connector:

$5.00

http://www.carplugs.com/products.html

Project Boxes:

TBD

To Be determined based on size of final project

GPS Antenna (if applicable):

$35.90

http://www.cutedigi.com/product_info.php?cPath=248&products_id=4289