Car Jackers - Project Proposal

CS 3992 (Spring 2012)

Team members:
Jeremy Bonnell
Tong Wu

I. Functional Description

The Car Jacker has 3 basic functions controlled via a wireless phone: the remote start/ stop of the vehicle, remote lock/unlock of the vehicle's doors, and the control of the vehicle's heating/cooling settings. The heating/cooling settings it controls are the fan settings, temperature settings, vent settings, AC settings, and rear defroster settings. Control over these settings is enabled only after the vehicle has been started and is returned to dashboard controls when a person sits in the driver's seat. If time allows, additional work will be done towards other functions such as the implementation of a location detector that shows the position of the vehicle.

II. Motivation

The motivation behind the Car Jackers project is a personal one. The vehicle Jeremy's wife currently owns is a 2005 Toyota Camry. It has power-locks but did not come equipped with keyless entry. So for her birthday, last year, the vehicle was taken to a local car alarm and stereo company where a keyless entry system was installed. The system ran about $200 and had remote-start capabilities. Her previous vehicle had a remote-start option and, though she would have liked that as well, cost an additional $200 for the component. When it was discovered that any idea can be implemented for senior project, we felt this was the perfect opportunity to save a buck and discover if a remote-start component is really worth $200.

While contemplating the idea of the starter implementation, other considerations resurfaced. Jeremy had remembered on numerous occasions, while using the remote-start option in her previous vehicle, his wife had often forgot to adjust the heater settings prior to exiting the vehicle. Having started it several minutes before departure, she would still have to go out to the vehicle and adjust the heater settings or scrape the windows and drive away cold. This is when the decision to take the project a step further and also implement remote heating/cooling controls was made.

III. Implementation

A-1. Hardware Components

Arduino Mega 2560

The microcontroller used for this project is the Arduino Mega which was purchased from

Sparkfun.com [1]. It has 256KB of flash memory and is more than enough memory for our purposes. It has four serial ports but only two were needed. One port is connected to the cellular shield so the phone can communicate with the microcontroller. The other port is connected to the onboard diagnostic system (OBD-II) which shows the engine status of the car. The Arduino also has plenty of digital pins (highlighted blue in Figure 1) for the PCB (printed circuit board) relay driver. The relay driver connects the microcontroller signals to the car and amplifies their output voltages. The cellular shield, OBD-II, and PCB relay driver will be explained in greater detail in later sections.

Some controls, such as the fan control, require a range of input voltages to drive each setting. In these cases, pins 51 - 53 (MOSI, CLK, SS), which is the SPI interface (highlighted green in Figure 1), are used. The SPI signals are connected to the PCB relay driver and ultimately control various voltages with digital potentiometers. An A-B USB cable is used for programming the board.. The A end connects to the computer and is a standard USB connection while the B end connects to the Arduino board and is a trapezoid shaped connection. Specifications for the Arduino Mega recommend using a power source of 7 to 12 volts-DC. Therefore, the 12V car battery is used to power the board. However, since the alternator can potentially boost the voltage level up to 13.8V once the car is started, a 12V voltage regulator is placed between the battery and board to prevent the board from exceeding voltage specifications. The Arduino Mega also has a 3.3 voltage supply, administered by an onboard voltage regulator, which is used to power the cellular shield [2].

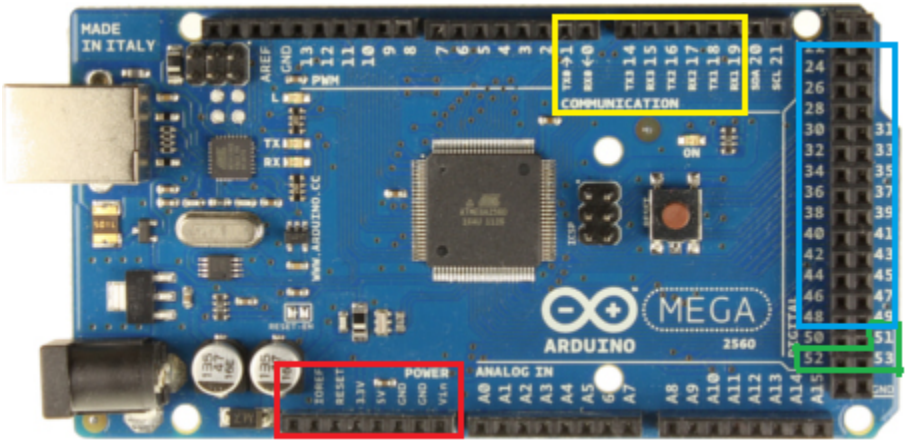


Figure 1. Arduino Mega 2560

Cellular Shield SM5100B

The SM5100B cellular shield was purchased from Sparkfun [1]. This particular cellular shield was chosen because it is able to communicate with the Android smartphone via SMS (short message service) technology. SMS is also known as “text messaging”. The shield is powered by the 3.3 voltage supply [1] from the Arduino (highlighted red in figure 1). The Arduino and cellular shield are able to transfer data by connecting one of the transmit/receive serial ports from the board (Serial-1 highlighted yellow in Figure

1) to one of the transmit/receive serial ports of the shield (highlighted blue in Figure 2). For receiving, pin 2 of the cellular shield is connected to pin 19 of the board. For transmitting, pin 3 of the cellular shield is connected to pin 18 of the board. The serial ports on the cellular shield are able to transfer up to 460 kbps. The frequency used is 1900 MHz since it is the frequency used by cellular phones in the US. In order for the phone and cellular shield to communicate, a SIM card and an antenna are required. The SIM cards were purchased from AT&T for $25 each and came with a 30 day unlimited text messaging plan.

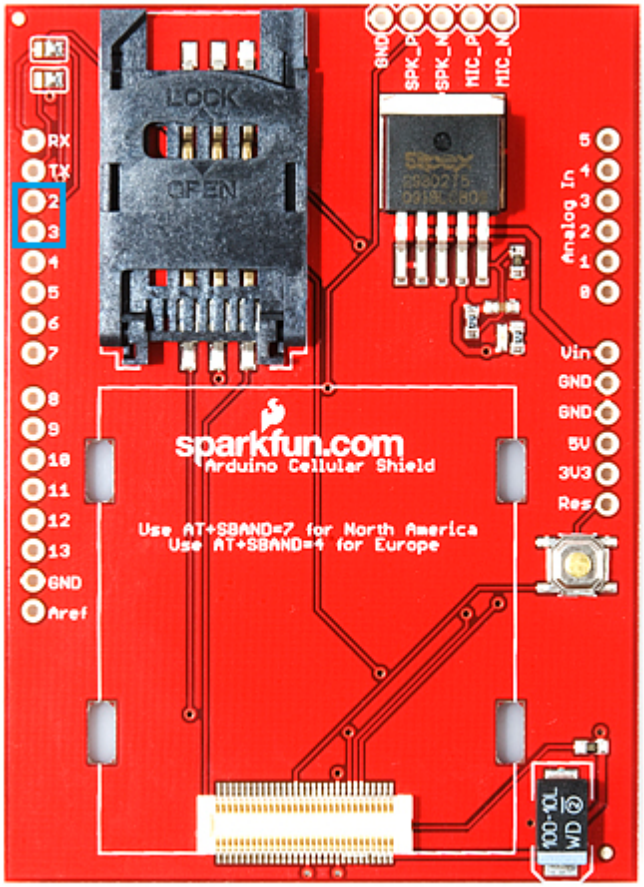


Figure 2. Cellular Shield SM5100B

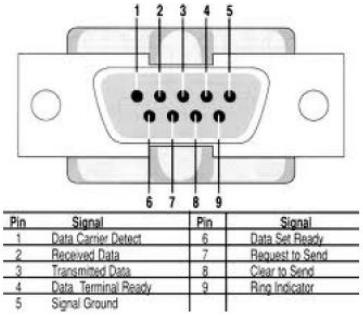
Quad-band Wired Cellular Antenna SMA

The quad-band wired cellular antenna sma (Figure 3) was also purchased from Sparkfun [1]. This particular antenna was chosen because it has a quad-band of 1900 MHz with a gain of 3.5 dBi [1] which can transmit and receive data from the cellular tower. It also performs well with receiving SMS messages inside a started vehicle, where noise factors have the potential to block incoming messages.

Figure 3. Quad-band Wired Cellular Antenna SMA

On Board Diagnostic System (OBD-II)

The OBD-II interface is used to report the status of the car to the Arduino board which is then used to send data to the Android. The larger end of the OBD-II cable is connected to the OBD-II port under the steering wheel and the smaller end is connected to the communication interface (part C Figure 4). The communication interface is then attached to Arduino Serial-2 port. The OBD-II interface is only used to report the start/ stop status of the car.



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

A. OBD-II cable **B. Pin reference** **C. OBD-II Communication Interface**

Figure 4. OBD-II Interface

PCB Relay Driver

Each option controllable by the Car Jack system has 2 signals parsed through the PCB relay driver. One signal is the vehicle's default driven control and the other is the Arduino driven control (these options can be observed by referencing the "vehicle controls" in Figure 5)

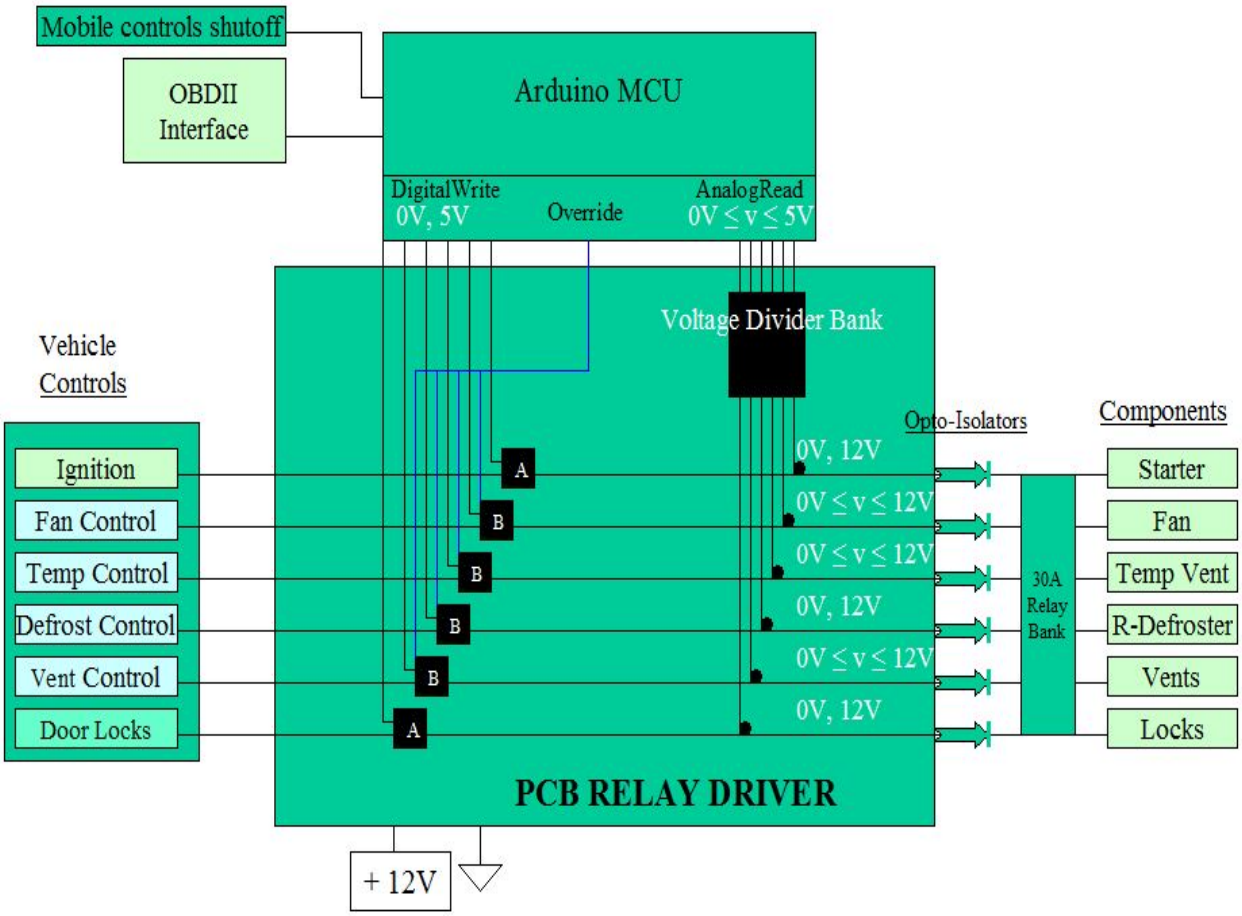


Figure 5. PCB Relay Driver Overview

There are three primary tasks the relay driver is responsible for and can be denoted as boxes 'A', 'B', and "Voltage Divider Bank" in Figure 5.

Amplifier Circuit (Box 'A')

The amplifier circuit 'A' is only used for the ignition and door lock systems. This is because the car must typically be able to be stopped, started, locked, or unlocked at any point in time. Also, since the ignition and locks are only powered for a short period of time (50ms - 1500ms) it was not necessary to have a switch select between vehicle

and Arduino controls. The circuit is a simple MOSFET amplifier circuit and is driven by (0V,5V) digital pins from the Arduino. The transistor used is the SQ2310ES, N-Channel, 0V - 20V, 6A MOSFET by Vishay (Figure 6), with a threshold voltage of 1.5V [3].

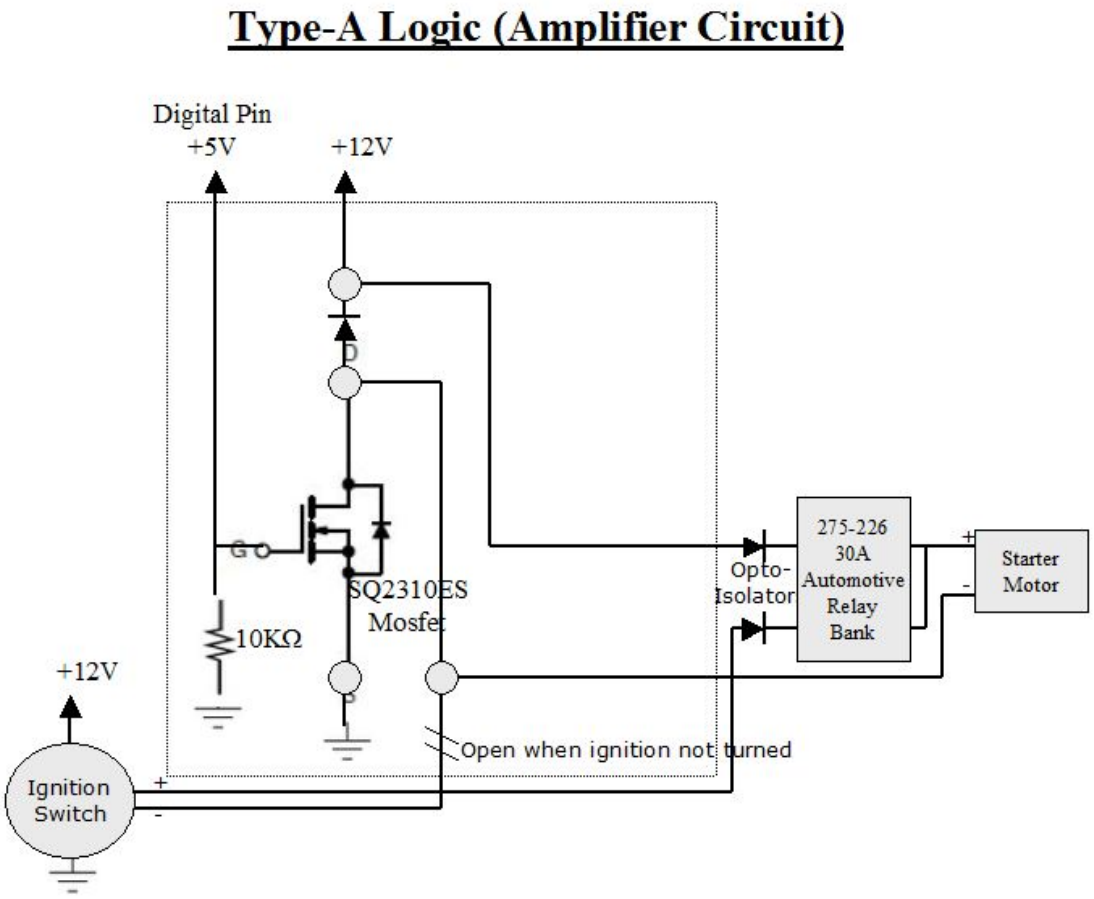


Figure 6. PCB Relay Driver - Amplifier Circuit

The boxed area of Figure 6 signifies logic within the PCB relay driver. When the vehicle's default ignition and lock systems are operated and the digital pin is 0V, the MOSFET behaves as an open circuit. This allows the car to be started in the normal fashion. If the digital pin is set high to 5V and the ignition is not being turned, the transistor becomes saturated, allowing current to flow, and the ignition switch acts as an open circuit. This allows the Arduino to control the ignition and door lock systems.

Switch/Amplifier Circuit (Box 'B')

The task of the switch/amplifier circuit is to select heating/cooling settings between the vehicle driven controls and the Arduino driven controls as well as amplify the signals between 0V and 12V. This circuit uses the same MOSFET as the type-A circuit to amplify the signal but uses a different configuration. Since the heating/cooling controls of the vehicle are essentially potentiometers that control a variable amount of output

voltage through resistance, the type-B circuit had to be configured to behave in the same manner. However, the Arduino can only directly output either 0V or 5V. With the aid of the Arduino's SPI interface and digital potentiometers, it was found that a variable amount of voltage can be achieved. The digital potentiometer (Figure 7) used is the AD5206 from Analog Devices [4], with 50K terminal resistance and 256 positions .

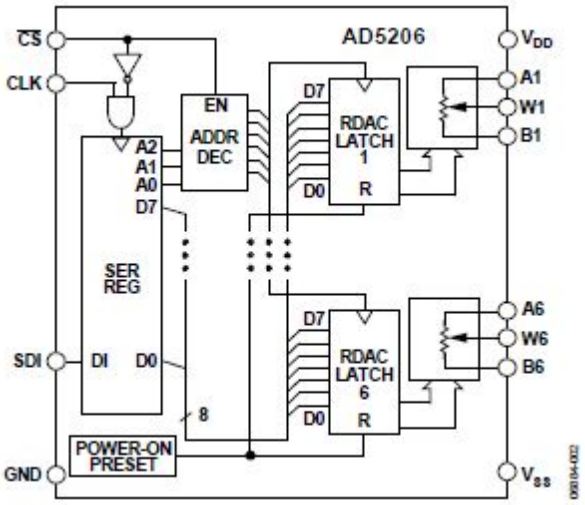


Figure 7. AD5206 Digital Potentiometer

The output voltage of the amplifier circuit is controlled variably by the voltage at the gate terminal (+5V terminal, Figure 8). The drain output is driven by the 12V car battery with a voltage regulator preventing output voltages above 12V. When the digital potentiometer is set to high resistance nearest the ground, the full 5V is applied to the gate. This actually causes the voltage at the drain terminal (node below 2.5K R, Figure 8) to be 0V. This is because the current is high enough for the full +12V source to be consumed through the 2.5K resistor. It was found that as the voltage at the gate is lowered, the voltage at the drain goes up to the full 12V. That is until the gate voltage becomes approximately 1.8V. Once the threshold voltage of 1.5V is approached, the transistor is no longer saturated and the drain voltage is 0V. This process allows the system to control the voltage output sent to the switch.

Since the heating/cooling signals are driven an indefinite amount of time, the Car Jacker must select between dashboard controls and Arduino driven controls. This is done by means of switches in the relay driver type-B circuit. The switch used for this circuit is the ADG1436 SPDT (2:1) switch, with 0V to 12V input and a 5V select line by Analog Devices [5] (Figure 8).

Type-B Logic (Switch/Amplifier Circuit)

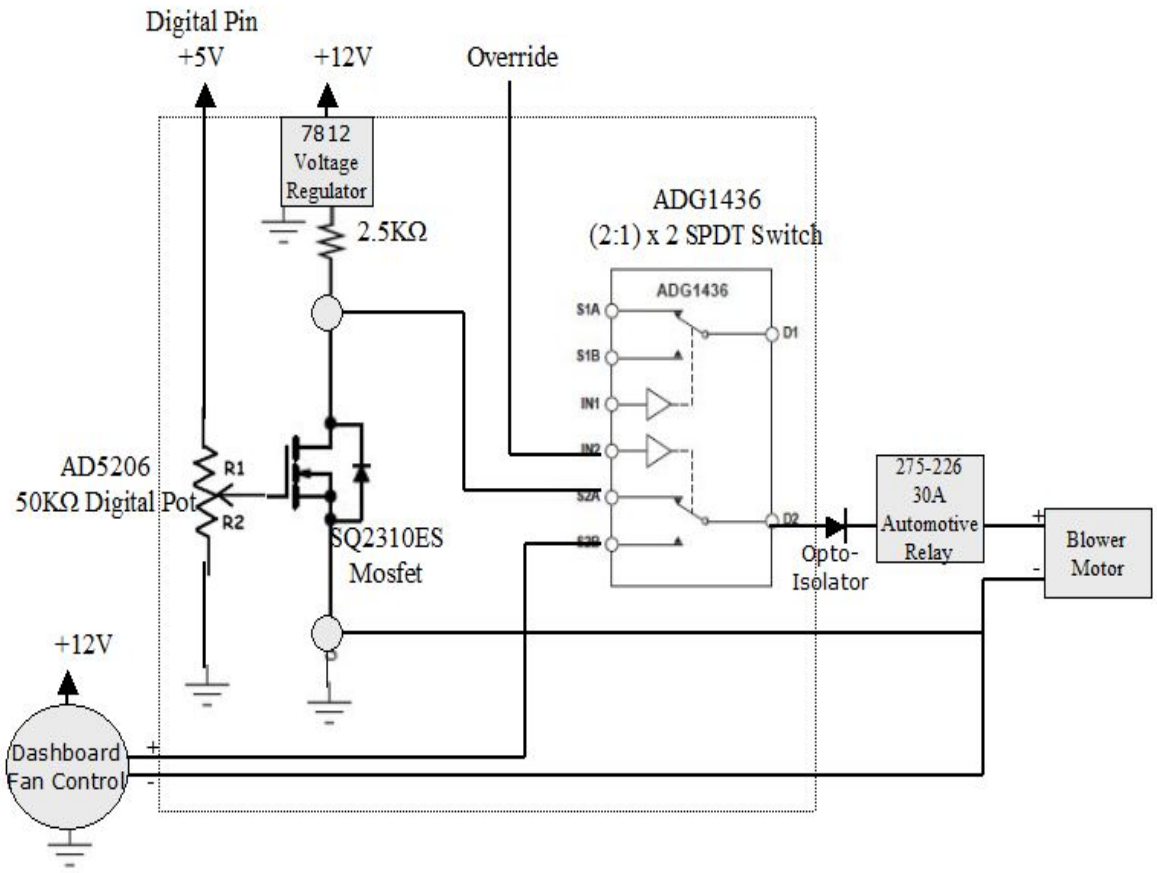


Figure 8. PCB Relay Driver - Switch/Amplifier Circuit

The switches can essentially be thought of as multiplexers, with an override signal from the Arduino serving as a select line between the two options. When the override signal is low, the vehicle's default heating/cooling settings are selected. When the override signal is high, the Arduino driven heating/cooling settings are selected.

Voltage Divider Bank

This portion of the circuit is self-descriptive. It is a set of voltage dividers that lower switch output voltages to the Arduino's 5V input specifications. Each voltage divider simply consists of a 5K resistor leading to an analog in port of the microcontroller and a 7K resistor leading to ground. This causes the maximum switch output of 12V to be reduced to 5V and safely be used as Arduino input. The Arduino typically does not poll these ports unless a call is made to read the voltage from a particular pin in software. Once an analogRead() call is made in software, the analog port allows voltage to pass long enough to take a "snapshot" of the line voltage, the line is set back to an open circuit, and an integer of 0-1023 is returned in software. The integer can then be deciphered and returned to the Android to display the vehicle's current settings.

Also within the voltage divider circuit is another ADG1436 switch positioned between the 7K resistor and ground. One input of the switch is connected to nothing, while the other input is connected to the 7K resistor. The output is connected to ground. Using a digital pin from the Arduino, the select line is set to default on the nothing input. Right before the call to analogRead() the select line is set to the 7K input then set back to nothing afterwards. This keeps the 7K branch from being a continuous drain on the switch output.

PCB Relay Bank

The vehicle's starter motor, blower motor, locks, etc.. require up to 30A of current to operate. Therefore a PCB relay bank is set in place to ramp up the current potential with 275-266 SPDT 30A Automotive Grade Relays from Radio Shack [6].

A-2. Hardware Procedure

Using the SMS (Short Message Service) capabilities installed on the Android smartphone, a signal is transmitted to the cellular shield and the data is parsed to the Arduino microcontroller. OBD-II data is also sent to the microcontroller to determine if the engine is on or off. If the engine is started, the option to override the default settings is enabled. The Car Jack system is installed in the Camry by splicing into the vehicle's heating/cooling, door lock, and ignition systems. This allows the microcontroller to capture and decipher the vehicle's current settings and transfer them to the phone where a GUI displays them to the user. A general overview of the design can be seen in Figure 7.

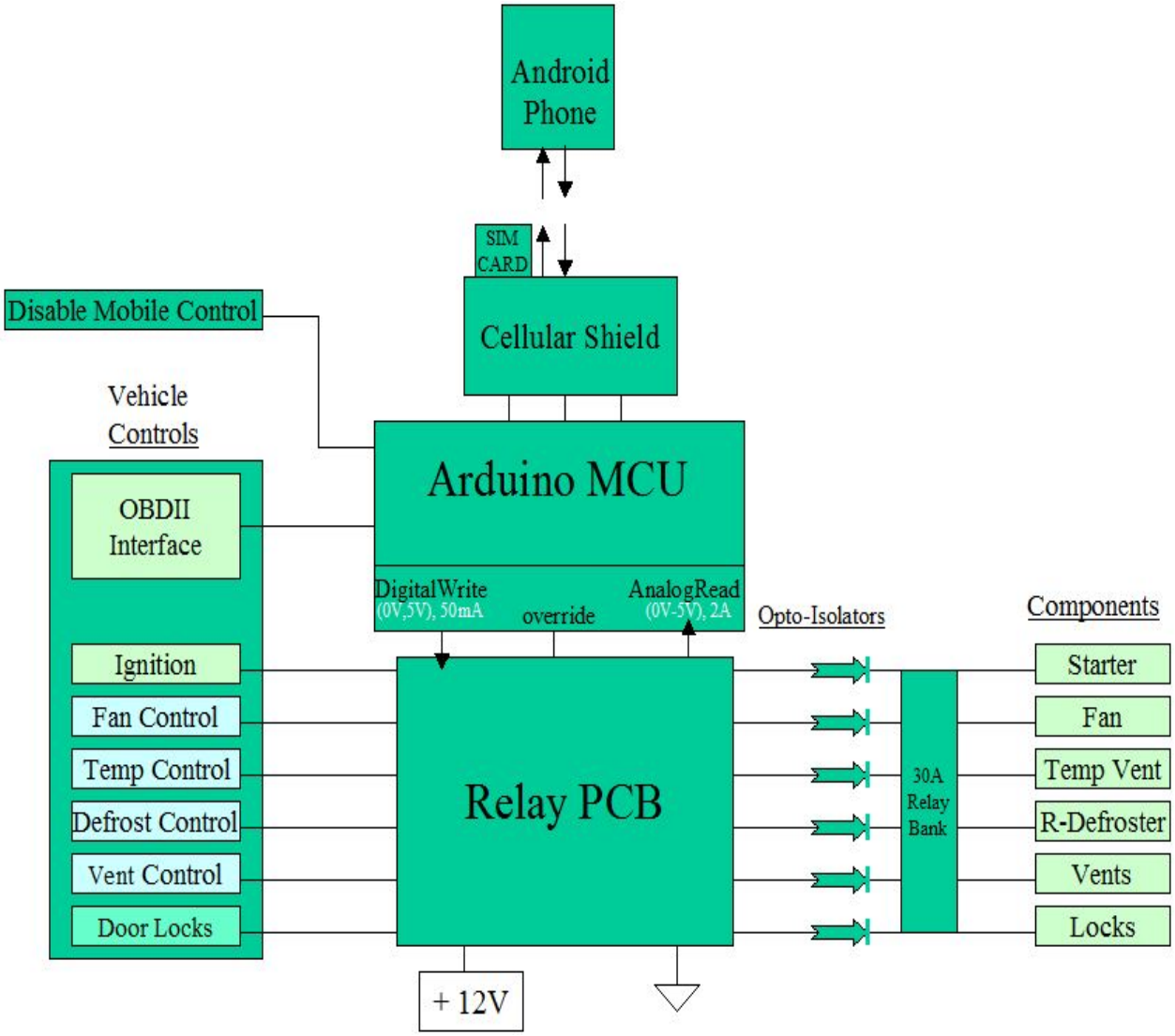


Figure 9. Design Overview

Some components within the vehicle's wiring scheme, such as vent and fan settings, are controlled by servo motors or blower motors deep under the dash. Therefore in order to parse the signals through the microcontroller, the splicing of the heating/cooling and ignition wires are done at the back of the dashboard controls. This required a fair amount of reverse engineering since the setting of every controller's voltage, current, and resistor values had to be recorded in order for the microcontroller to decipher the which settings they are currently on. This information is also used to determine which settings were needed for the digital potentiometers. Within the microcontroller, the software implementation is responsible for allowing direct or Arduino control of the settings. For Arduino control, this is done by setting the override signal to high and adjusting the digital potentiometer values to control voltage levels within the PCB relay driver. In addition, each signal from the relay has an opto-isolator attached to prevent reverse EMF (electromagnetic flux) which could potentially destroy the microcontroller circuit.

The phone first initializes the system by requesting the vehicle start/stop status. The OBD-II interface checks the ignition status of the vehicle and sends it to the Arduino to return to the phone. If the car has not been started, the option will be enabled. Otherwise the heating/cooling, door lock, and start status of the car will be fetched and returned to phone. At this point the option to override the heating/cooling systems will be given to the user. If an override signal is sent by the phone, the Arduino enables the override signal to the PCB relay driver. The dashboard heating/cooling settings then short circuit while the Arduino takes control of the signals and begin sending digital signals to power the PCB relay driver circuits an SPI output is used to set the digital potentiometers. The Arduino settings are initially set the same as the dashboard settings, but can now be adjusted via the Android app. As long as the Android application is running and a person is not in the driver's seat, the door lock function is always enabled. Heating/cooling functions are available if the car is started and no one is in the driver's seat. Start/stop functions can always be invoked unless someone is driving the vehicle. These detections are done by the driver's seat sensor. Once a person has entered the vehicle, all CarJack functions are disabled, save the ignition. The driver needs to put the key in the ignition in order to unlock the steering wheel. So 10 seconds are given to do so before the ignition times out and is disabled as well.

B-1. Software Implementation

The software implementation of this project will be a GUI interface written on an Android smartphone and will display control options and vehicle's start and heating status to the phone. It will generate encoded signals to be sent to the microcontroller via text messaging. This will require the use of a SIM card in order to transmit and receive the SMS signals. Once the signal has been received and decoded by the microcontroller, a confirmation signal will be sent back to the phone and the phone will display the current status to the screen. Both the phone and microcontroller will need to know the encoding scheme used in order to communicate effectively.

Arduino

The board will parse the messages sent from the phone and perform the appropriate tasks based on the message through the relay by setting the digital pins to HIGH or LOW. The fan and temperature controls need different voltages from the arduino so we decided to use digital potentiometers. The digital potentiometers are operated with the arduino SPI interface, pins 51-53. A tutorial is available on arduino's website [2]. The list of tasks are listed in figure 10. It will also receive signals from the OBD-II and send the appropriate message of the engine status to the phone through the cellular shield by sending AT commands along with the android phone's number and messages. All the programming for the board are done in arduino 1.0 IDE.

Android Application

The phone will send messages to the cellular shield with the SIM card's number hard coded in the software. The messages are sent with SMS (short message service) based that is linked with buttons in the android application. If a button is pressed, it will send a message to the arduino. The list of messages sent to the arduino is in figure 10. The messages are sent It will also receive messages from the cellular shield for the status of the car. All the android application's code will be done in eclipse. An image of what the GUI look like shown in figure 11.

Button Name / Task	Message
Start Engine	a0
Stop Engine	a1
Lock Door	a2
Unlock Door	a3
Defrost	b0
Temp Up	a4
Temp Down	a5
Fan Up	a6
Fan Down	a7
Override On	a8
Override Off	a9

Figure 10. List of button names and tasks with messages

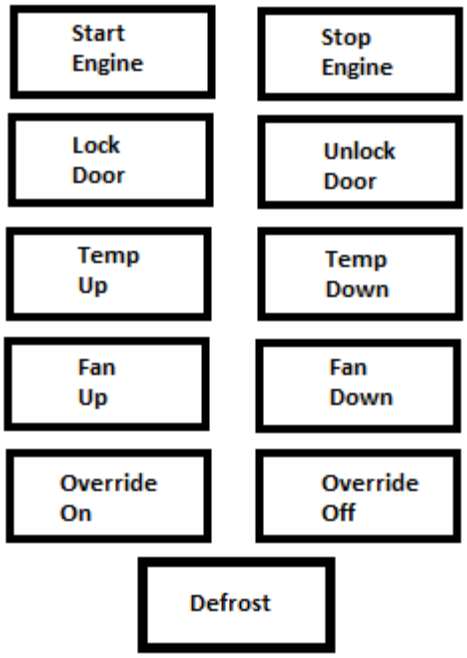


Figure 11. Android GUI

IV. Initial Tasking

The first task accomplished was the gathering of materials and components for the project. A breakdown of all items necessary for the project are as follows:

- Arduino Microcontroller Board
- Cellular Shield
- * Vehicle
- Android Smartphone
- Multimeter
- SIM card
- Antenna
- OBD-II Plug
- usb (type-A) to usb (type-B) adapter
- 2005 Toyota Camry wiring diagrams and schematics

This step is crucial since the initialization of the project could not begin until some of key components were acquired. All orders were placed and University components (such as the Arduino microcontroller) arranged for checkout by mid-April. The goal was to have everything acquired by the end of Spring semester 2012.

The next task to work towards is the sending and receiving of the SMS signal. This will began by writing a simple Android application that allowed the phone to send a short SMS signal to the transmitter/receiver attached to the microcontroller. The signal performed trivial tasks, such as turning on an LED. Then the advancement of more complicated tasks began to evolve, such as - send a sequence of signals, light a sequence of LED's, return a signal to the phone after the lighting sequence is complete, and display something to the screen. When the group was satisfied that the signals are being sent and interpreted correctly, the introductory part of the project was considered to be done. However after the reverse engineering portion of the project was complete, a suitable Android application was written to interact with the hardware. This step required minimal wiring and was not a difficult portion of the project. The most difficult portion of the project was the PCB relay and wiring it up to the vehicle. The initial goal was to start and complete this task during Summer semester 2012.

The final task dealt with wiring and hardware components of the project. The dashboard of the vehicle had to be removed to access the wiring of the controls. Next, each setting had to be read and documented with the multimeter by splicing into the control wires. This was done for each controller and for every setting. The next steps were done incrementally:

- Wire controls into and out of the microcontroller
- Write Arduino microcontroller application and test outputs to see if they matched

the specifications needed for dashboard controls
- Build PCB relay and test against the Arduino application
- Plug OBD-II into vehicle and test readings
- Adjust Android application to interact with microcontroller
- Install Arduino, cellular shield, PCB relay driver, and PCB bank combination into the Toyota Camry.
- System testing

A simple implementation was first constructed to perform trivial tasks, such as turn on rear defroster. Then more functionality was added to the controls until the project was complete. The goal was to complete the project one month before the “demo day” exhibit in Fall semester 2012.

V. Risks and Interface Issues

- Relay Driver required a lot of testing before installing. Bought blower motor, vent servo motors, etc... to test in lab first
- All PCB components were not accessible in PCB design library. Had to design with slots to solder aftermarket chips in.
- Had to make sure all sectors were isolated so back elecro-magnetic flux did not fry components
- Danger of making car inoperable. Always made wiring diagrams before disassembling

The risks were subject to, like everyone else, running the risk of not receiving all materials in time. Also, the OBD-II and relay driver, and signal interpretation proved to be difficult. We also had to be careful about dismantling the vehicle and its wiring. If caution is not practiced the dashboard or one of us could have gotten hurt. We also needed to make diagrams so we did not detach a bunch of wires and not know how to rewire them. With only a two-man group, we also ran the risk of not having enough man-power to complete the project. This is why we needed to start as early as possible. We had to work very hard to complete this project in time! But in the end, the challenge and learning experience were well worth it.

VI. Tasking and Scheduling

1. Communication between Phone and Arduino	3 weeks
2. Android Application and Arduino code	2 weeks
3. PCB for Relay	2 week
3. Connections: Car with Relay	4 weeks
4. Connections: Arduino and Relay	1 week
5. Communication between Phone and Car	4 weeks

6. Finishing touches: Controls from Phone to Car 4 weeks

Testing/Debugging every step at a time

VI. Bill of Materials

Toyota Camry	$...
Android Phone	$...
Sim Card (3) from AT&T	$75
Arduino Mega 2560 (1) from Sparkfun	$70
Cellular Shield SM5100B (1) from Sparkfun	$100
PCB Relay (1) from automate (Resistors (7), Potentiometers (2), Mosfets (7), Diode (7), Opto Isolator (7), Multiplexer (2))	$100
OBD-II (1) from ebay	$20
Antenna (1) from Sparkfun	$20

VI. Bibliography

References

1. Sparkfun. [Online] April 21, 2012. [Cited: April 21, 2012.] http://www.sparkfun.com/
2. Arduino. [Online] April 21, 2012. [Cited: April 21, 2012.] http://arduino.cc/en/
3. Vishay. [Online] April 21, 2012. [Cited: April 21, 2012.] . N-Channel MOSFET http://www.vishay.com/docs/67036/sq2310es.pdf
4. Analog Devices. [Online] May 1, 2012. [Cited: May 1, 2012.] .AD5206 Digital Pot. http://www.analog.com/static/imported-files/data_sheets/AD5204_5206.pdf
5. Analog Devices. [Online] April 21, 2012. [Cited: April 21, 2012.] . SPDT Switch http://www.analog.com/static/imported-files/data_sheets/ADG1436.pdf
6. RadioShack. [Online] May 1, 2012. [Cited: May 1, 2012.] .SPDT 30A Automotive Grade Relay. http://www.radioshack.com/product/index.jsp?productID=2062477