COE 485: Senior Design Project

Smart Car Project Progress Report

1 Introduction

Remote starting the car is one of the great and important features that is missing from most of the cars sold globally. Although some cars support the remote starting functionality, dealers usually limit such feature to cars sold in certain regions such as the US or to high-end models only. Additionally, manufacturers use short-range remote controllers, which limits the benefits of using the system. This project will help bring this important feature to almost all cars, either old or new, and will add many important features such as GPS tracking in the system, all without compromising the safety and security aspects.

2 Problem Statement

The project should add the feature of remote starting the car to almost all cars that miss this feature, and it is important to use long-range connections in order not to limit the usage of the system.

The ability to start the engine remotely is very useful in two conditions:

1. In extreme hot weather conditions, it is very convenient for drivers to start the car along with the air conditioner prior to getting out of the house and driving the car. Especially that the temperature inside the car is expected to be hotter than the hot outside temperature while it is parked. According to weather.com, the temperature of a car cabin can reach 59 Celsius if the outside temperature is 32 Celsius, an extra 27 Celsius degress after parking the car for 90 minutes only.
2. Warming up the car in extreme cold weather conditions, which will save the time of drivers by letting the car warm before reaching it.

There are negative impacts that may result from the usage of the remote starting system:

1. Users may start the engine much earlier than what they need. Leaving the engine running in idle mode for long periods consumes a lot of gas and affect the engine health negatively. This negative effect can be limited by stopping the engine automatically if the driver does not reach the car within a certain time period.
2. Having the car running in public without people inside may attract thieves to hijack the car. Although the system can be designed to prevent thieves from stealing the car, losses can happen during hijacks attempts such as windows glasses breakage.

The other part of the system is GPS tracking, such feature will be very useful in several situations:

1. If the car is stolen, the user can easily locate its location accurately
2. Locating a driver who needs help by giving access to relatives or trusted people
3. Locating the car after parking it in a large parking area

However, car GPS tracking can have negative impacts as well, such as violating the privacy of other people by tracking their location in real time, especially if the driver is unaware of having such a system installed in the car if he is not the primary owner or if the car is rented.

3 Project Specifications

The system must meet the following user requirements:

1. Ability to start the engine using mobile phone from long ranges.
2. Preventing thieves from stealing a car after remotely starting it
3. Locating the car location accurately from anywhere in the world
4. The system must not interfere or disable the regular operation of starting the car using the key.

The above user requirements can be translated into technical requirements as follows:

1. GSM modem will be used to communicate with the system from virtually anywhere in the world for performing the control functions as well as location tracking. GSM connection must not replace the short-range communication channel as GSM may have large latencies sometimes and the GSM network may fail at any time. Additionally, using the GSM network may require additional user costs that can be avoided in short ranges.
2. Bluetooth connection will be used as a short communication channel between mobile phones and the system
3. GPS module will be used for locating the car position through the GSM communication channel, and external antenna to the GPS should be used because the system will be installed in hidden locations in the car most of the times; signals in such locations are not strong enough to get a 3D GPS location.
4. The system should tap into all the required ignition wires and should minimize cutting wires or performing modifications. If doing modifications is necessary, the car should operate normally after doing all the required modifications in the absence of the system or in case of failure.
5. After remotely starting the car, the user should use the car original key to drive the car, otherwise the system should turn the engine off to prevent stealing the car.
6. An Android mobile application should be developed as a user interface to the system.

4 Task Schedule

For designing and prototyping the system, the process was divided into several task.

1) **Determining the exact required components and purchasing them**

   This task is mainly about analysing the system technical specifications and determining all the required components by model that will be used in designing the system. It also includes purchasing the components either from the local market or online.

   Timespan: 2nd week of February
   Status: Completed

2) **Identifying all the required car wirings**

   The system will be prototyped and tested on a Honda Accord 2003. It's necessary to identify all the required wirings, their colors and locations in the car. Some of the wirings have been identified in an early stage, and the task is about identifying all the remaining wirings.

   Timespan: 2nd and 3rd weeks of February
   Status: Completed

3) **Main Circuit Design**

   Designing the circuit of the system for interfacing with several parts of the car, interfacing between different components within the system itself and driving the relays as well as the power part.

   Timespan: 3rd week of February – 2nd week of March
   Status: In progress
   Status Details: Working on the power part as its requirements changed with the addition of components such as the GSM module.

4) **Car relays box**

   This task is directly related to prototyping the remote starting part of the system, where several relays needs to be installed and connected to the ignitions wires of the Accord and the system.

   Timespan: 2nd week of February
   Status: Delayed
   Status Details: An initial prototype for testing was completed and installed successfully. By time, one of the relays have failed and it is not easy to replace it in the current design. Working on another professional design which will be fabricated in the FABLAB that supports easy plugging and replacing of relays.

5) **Bluetooth Communication**

   Working with the Bluetooth chip, identifying and soldering all the needed pins, configuring the chip's name and PIN, as well as testing communication between Arduino and an Android mobile.

   Timespan: 4th week of February – 1st week of March
   Status: Completed

Status Details: The tasks in the description is completed, but the major part of the Bluetooth communication lies in the software part of the Arduino and Android which is not completed yet except for testing.

6) GSM Communication

This task is about the GSM modem, getting the modem working and registered in the cellular network, testing it by making calls and sending text messages and integrating the modem fully with the system.

Timespan: 2nd – 4th week of March

Status: In progress

Status Details: Most of the testing is complete; the testing was between the modem and a PC using serial interface. After completing all the testing, the remaining part is integrating the modem with the microcontroller and performing all the required functions automatically by software.

7) GPS Module

This task is about getting the GPS module working and getting accurate locations by verifying the coordinates on Google maps. Additionally, integrating the module with the system.

Timespan: 2nd – 4th week of March

Status: In progress

Status Details: Testing the GPS module using PC and a serial interface is completed. The remaining part is integrating the module with the system to perform the required functions.

8) Arduino Software

Writing the code for the Arduino microcontroller for complete integration with all the system components and the car. It includes Bluetooth and GSM communication, as well as remote starting the car.

Timespan: 3rd week of February – 3rd week of April

Status: In progress

Status Details: This is one of the major tasks in the system as it includes a lot of details and subtasks. A huge progress is done in this task as the code for all the completed tasks such as starting the engine is already written. The remaining progress on this task depends on the progress on the other tasks such as GSM communication, GPS integration and the Android Application.

9) Android Application

The development of an Android application that will be the main user interface for users remotely controlling the engine or tracking the location of cars.

Status: Not started

5 Completed Tasks

1) Determining the exact required components and purchasing them

Performing this task started by analysing the system technical requirements. The following major components needs to be purchased:

- Microcontroller
- Bluetooth module
- GSM modem
- GPS module
- Relays

There are several models from several companies for almost each of the components listed above, therefore, it was necessary to make a reasonable comparison on which model to buy and use for prototyping the system.

A. Microcontroller

There are several candidates in the microcontroller section, they include PIC microcontrollers, Arduino boards or even an FPGA chip which is not considered a microcontroller.

The PIC microcontrollers are good for their cheap prices and the availability of huge models that vary according to the user needs. They seem like a right choice for final production with large quantities. However, for prototyping, their cost is not so cheap given the need for purchasing a development board or a programming board for programming the chips. Using PICs for prototyping has no justification over using an Arduino for example.

FPGAs are very powerful, however using a complete FPGA board for prototyping will be very costly, will take large space and doesn't seem the right choice for final production as well. Additionally, although FPGAs are powerful, they don't fit in this type of application where real time processing is not needed, and FPGA lack the availability of software libraries that will ease the process of development and integration with the system components.

Arduinos are the right choice for this type of applications; complete boards for prototyping are available with reasonable prices, also, the ATmega chips that are used with the Arduino are available with low prices for final production. Main advantages of the Arduino is its simple IDE that will ease the process of development, also the availability of huge range of software libraries that will accelerate developing the system and integrating its components.

The board that was chosen for prototyping is the MEGA2560 shown in Figure 1. It is based on the ATmega2560 chip and runs at 16 MHZ. It has four hardware UART interfaces that are needed in this application, where the other models have only one UART interface. It also has all the needed features such

as having 6 external interrupts among the 54 digital ports. It also has 16 analog inputs pins but only few of them will be used.

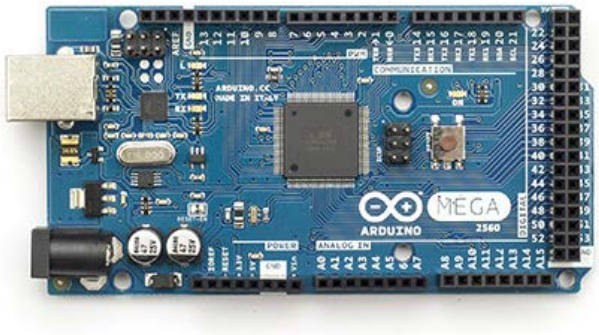

Figure 1 Arduino Mega 2560 board

The new Arduino Due board was an attractive candidate for the project. It is more powerful than the Mega board and runs at 84 MHZ, but it consumes more power during operation, which is a major disadvantage for our car application.

B. Bluetooth Module

A wide range of Bluetooth modules was found and their prices vary a lot. Many of the modules found that were part of ready development boards were limited in the pins offered. For this application, the module needs to support serial interface and must have pins available that indicate the connection status which will be used as an interrupt to wake up the microcontroller to save power, and a pin to access the command mode to configure the module should be available.

Figure 2 Bluetooth Module

Five pieces of a cheap and very small module for prototyping that have all the requirements were purchased for ~17SR/module. One module will be used in this application. The module is shown in Figure 2.

C. GSM Modem

The GSM modem is one of the important parts of the system. A good range of modems was found. They vary a lot in prices as well as in features supported such as supporting 3G networks, quad or dual band GSM networks. The modem for this application must support sending and receiving text messages.

- **The Arduino GSM shield**

  The Arduino GSM shield was a very attractive board for prototyping especially that the Arduino community directly supports it, also libraries for easy and direct controlling of this modem were available. However, it was not chosen as the GSM modem for this application for the following reasons:

  1. The version of this board that supports external antenna was not available at the time of purchasing.
  2. The board is overpriced, offered at ~$100
  3. It was not clear whether it is possible to put the modem into power saving modes to save power.

- **The SIM900 modem**

  The SIM900 modem seems like the right choice as it supports all the application requirements. A wide range of boards that uses this modem was found, and they vary in prices and number of pins available to the user. The board chosen for prototyping is based on the SIM908 chip as shown in the GPS module section.

D. GPS Module

GPS modules are one of the modules that are widely available. Similar to GSM modules, a lot of options were evaluated before choosing the right components to purchase. They differ in power consumption as well as the number of satellites used.

- **The SIM908 modem**

  The SIM908 modem packs all the GSM modem features of the previously mentioned SIM900 modem, but it is equipped with a GPS engine as well. What makes the SIM908 GPS special is that it uses A-GPS (Assisted GPS), a feature that is usually available in smartphones that uses the geographical location provided by the cellular network to accelerate the process of getting a 3D GPS fix. The modem also supports direct control of the GPS power system for power saving.

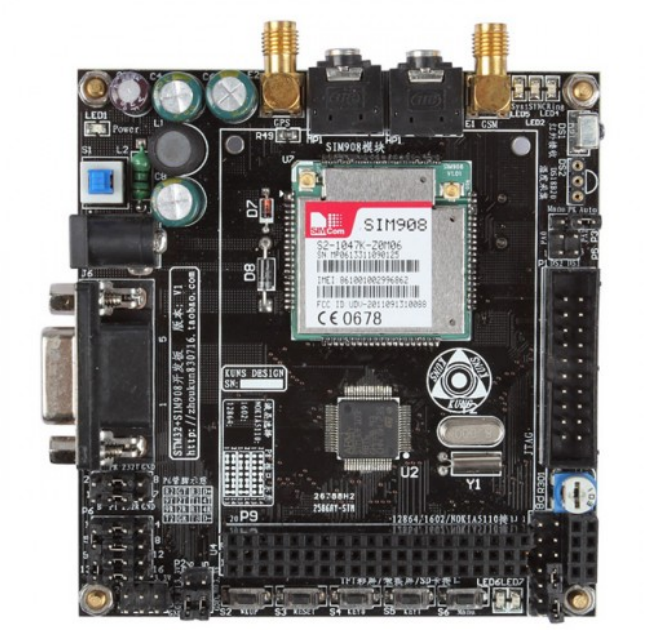

Figure 3 SIM908 development board

A board from a Chinese provider that uses the SIM908 chip was purchased for 96$. It has two antenna inputs for the GSM network and the GPS engine, and it is good for prototyping as it has an RS232 interface for PC debugging. The board is shown in Figure 3.

E. Relays

There are some ready to use relay boards in the market, however, they cannot be used in this application because all of them use relays that are rated for 10A DC or less. To control the ignitions wires of a car electronically, relays that can handle large current ~30-40A should be used. Relays that are marketed as automotive relays can handle such amounts of current. A pack of relays has been purchased for the purpose of controlling the ignition system of the car.

Figure 4 Automotive Relay rated at 40A 14V

2) Car wirings identification and analysis

The required wirings of the Accord car where the system will be prototyped needs to be identified and located. By analysing the system technical requirements, the following wirings have been identified per category:

• Ignition System Wirings

These are the main wirings that will eventually be controlled using the high current relays. Because they carry large current, they usually have large diameter compared to the other wirings which make them easier to locate.

Two methods helped in locating the ignition wires and verifying the identity of each wire. First, by reading the service manual of the car. This manual is intended to guide Honda dealers on how to identify the source of ignition problems, and the guide mentions the wiring colors of the ignition wires and their location. Second, by using a voltmeter and measuring the voltage of each wire relative to car ground in the different switch positions.

After tracking the wiring harness coming from the ignition switch under the steering column, the following ignition wires have been identified:

- **White Wire:**
  This wire is directly connected to 12V battery. It's used to provide battery power to the other wires when desired.

- **White/Red Wire:**
  This wire powers the accessory part of the car such as the radio and DVD player when connected to 12V battery.

- **Black/Yellow Wire:**
  This is the first main ignition wire that provides power to the several main components of the car when connected to 12V battery.

- **Black/Red Wire:**
  This is the second ignition wire, it provides power to several secondary components of the car such as the heater and the AC when connected to 12V battery.

- **Black/White Wire:**
  This is the starter wire that provides power to the starter motor while starting the car when connected to 12V battery.

• Security System Wirings

It's necessary to find the wires that will give control of enabling and disabling the security system of the car. This is needed to prevent the car and the original security system from starting the panic mode at the attempt of starting the engine remotely using our system.

After a deep search, the security system of the car is integrated in the door module of the driver side inside the door panel. Two wires were identified that can control the security system of the car. The white wire of the module disables the security system, while the white/red wire enables the system. Both are activated by sending a ground pulse. Unfortunately, the two wires control the doors locks along with the security system. For example, disabling the security

system unlocks all the doors of the car as well. No other wirings to disable or enable the security system without changing the doors lock position were found, which is not the case with several other car models. This could leave the doors of the car open after remotely starting it, but a workaround is to relock the doors after disabling the security system.

The two wires were tapped and extended to the under-dash panel of the car.

• **Doors Locks Wirings**

A way to control the doors locks without using the security system wirings is needed for two reasons. First, to be able of relocking the doors after disabling the security system while remotely starting the car. Second, because the security system wirings do not respond to any pulses while the engine is running, while a user may want to lock or unlock the doors while it's running from outside the car. Even the original car remote control do not work while the engine is running to lock or unlock the car, as most cars are not designed to have the engine running while the key and the driver are outside the car.

Again, it was not easy to find central wires that control all locks of the car except for two wires found in the module of the passenger side inside the door panel. The two wires control the locks of all doors without changing the security system status of the car.

These two wires were also tapped and extended to the under-dash panel of the car.

• **Immobilizer Wirings**

It is true that the car will not panic while remotely starting it after disabling the security system of the car, but it will never start while the immobilizer system is working, also it cannot be disabled as the car computer will not allow fuel to flow to the engine without a confirmation from the immobilizer system. This system is almost present in all cars sold within 12 years from writing this report.

The car key has a built in RFID tag that is activated and read by an RFID loop near the key lock of the 2003 Accord. Other cars may have different designs. The data read from the key is sent to the immobilizer for verification. The data is sent on a blue/red wire near the key lock of the car under the steering column. Also, a security key sign lamp will flash at the instrument panel if the car was started without a key inserted in the key lock, a blue/orange wire powers the lamp of that sign. Details on how to bypass the immobilizer will be on the remote starting design section of the report.

3) Engineering Design

a. Engine Starting Hardware Design

This section will cover the design and integration of several components that are directly related to starting the engine from a microcontroller.

i. Ignition Wirings Control

The ignitions wires that were shown in the wires identification section before will be controlled using high current rated relays, by measuring the voltage of each wire relative to ground in the different switch positions, the wires connect as shown in Table 1.

Table 1 Ignition wires connectivity in different switch modes

Wire / Mode	White/Red ACC	Black/Yellow IGN1	White 12V	Black/Red IGN2	Black/White Starter
OFF					
ACC	○	———	○		
ON	○	———	○	———	
Start		○	———	○	———

The microcontroller must control the relays to connect all the ignition wires according to the connections shown in Table 1. The Atemga2560 uses 5v digital output with limited maximum current of 40mA which cannot drive our relays. An external circuit driving the relays using the Arduino digital output can be designed by using a transistor allowing the 12v power source to pass through the relay coils when needed. A diode between the coils of the relay is needed as well to protect the transistor.

To calculate the needed current to drive the relay we need to measure the coils resistance of our relays. The multi-meter reads a value of 85ohms between the coils. Therefore, each relay needs a current of:

$$I_c = \frac{V}{R} = \frac{12v}{85ohms} = 141\ mA$$

The 2N2222A transistor will be used in the circuit; it is available in the local market in cheap prices. Its maximum V_{CEO} is 40v > 12v , and its maximum collector current is 600mA > 141mA.

From the datasheet of the 2N2222A, the transistor has an h_{FE} (DC current gain) value of 100 at the conditions that are the closest to our application (10v , 150mA). Therefore, the resistance between the Arduino and the transistor can be calculated as the following:

$$Arduino\ needed\ output\ current = I_a = \frac{I_c}{h_{FE}} = \frac{141mA}{100} = 1.41\ mA$$

$$Required\ resistance < \frac{V}{I} = \frac{5V}{1.41mA} = 3.5K$$

Because the h_{FE} value is not very accurate, and to ensure that the relay will get enough current to drive it safely, a lower resistance of 1K will be used. Such resistance will only draw 5mA < 40mA from the Arduino digital output and the transistor will allow up to 0.5A > 0.14A to flow.

Note that driving the relays will be only when the engine is running or about to start which will not affect the battery life of the car and the power consumption of the system negatively.

Figure 5 shows the final circuit for controlling one relay, the relay is represented by its coils resistance of 85 ohms.

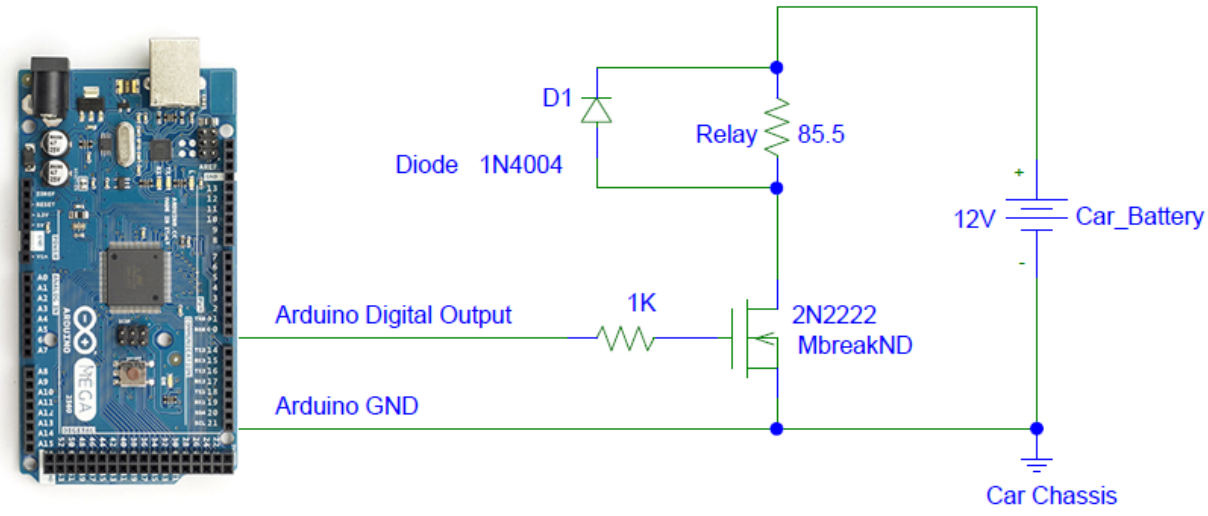

Figure 5 Circuit for controlling one automotive relay from a microcontroller

Software can activate the four relays according to the connections shown in Table 1 for each mode. The normally closed connection terminal of each relay will have no connection. The other two terminals will connect the white 12v wire to the other desired wire for each relay (ACC, IGN1, IGN2 and STARTER)

ii. Car Security System

The security system will panic during the attempt of connecting the ignition wires while it is active. The first thing to do before setting the ignition to the ON mode is to disarm the security system by sending a ground pulse to the disarm wire.

As noted before, sending such signal and disabling the security system will also unlock all the doors of the car in some car designs, which is not desired. In such cases, it is necessary to relock the wires of the car by sending a ground signal to the central lock wire.

To send ground pulses to any wire, a transistor connecting the wire to ground with its gate connected to the Arduino digital output is used as shown in Figure 6.

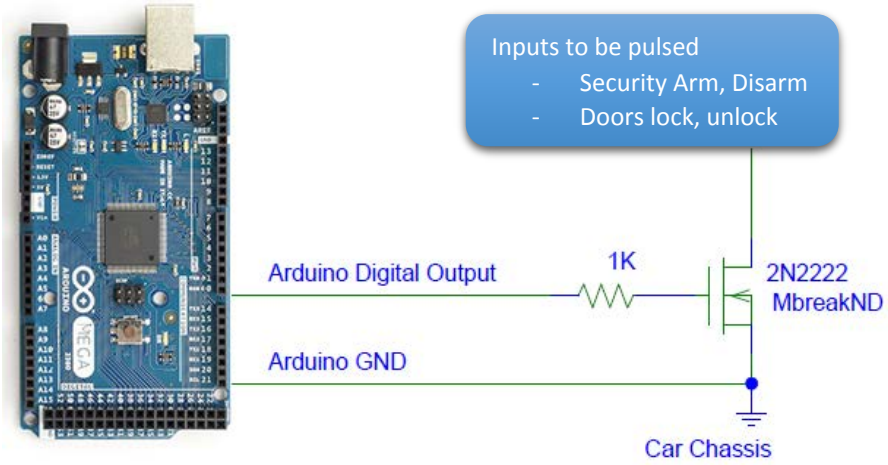

Figure 6 Circuit for sending ground pulses to external sources

The software part takes care of activating the transistor for only a short period to send a pulse to the desired wire. The following function sends the pulse to the desired digital pin for a given duration in milliseconds:

```
void sendPulse(int pin, int duration)
{
  digitalWrite(pin,HIGH);
  delay(duration);
  digitalWrite(pin,LOW);
}
```

iii. Immobilizer System

The immobilizer system is used to prevent thieves from hot wiring and starting the car, which is very close to what our system does. The Blue/Red wire mentioned in the wirings section of this report carries the data read from the key to the immobilizer system for verification. A method to bypass the system is needed in order to have a usable remote starting system. Several methods can be used to achieve this goal:

- **Placing a copy of the key near the key lock.**

  In this method, any user installing the remote starting system should make a copy of his key (The chip inside the key is enough) and place it near the key lock of the car. Whenever the system tries to remote start the car, the immobilizer will read the data from the copied key and will send its confirmation to the car ECU allowing fuel to flow to the engine.

  This method has one major drawback, which is disabling the immobilizer functionality. Hot wiring the car will become easy and the immobilizer will allow any attempt to start the car.

- **Designing the system to regenerate the signal**

  It is possible to design the system so it learns the data sent on the wire while starting the car using the key, and then regenerate the data on the same wire whenever the system tries to remote starting the car.

  The only problem with this method is that it is very hard to design the system to work with all car models. Such solution will make the system limited to certain car models. For example, designing the system while prototyping on the Honda Accord will make the system work on that car and similar Honda cars. This solution violates the goal of making the system universal for all car models.

- **Immobilizer bypassing modules**

  The other solution is to use the bypassing modules available in the market. Its concept is very close to the concept described above which is regenerating the signal on the data wire. The bypassing modules are provided by specialized companies such as 'Fortin Electronic Systems'. Bypassing modules for almost all car types have been designed and are available in the market.

The solution of using a bypasser module fits our application especially while prototyping. In terms of security, the bypassing module will be controlled from the microcontroller and it will be enabled only while remote starting the car. Additionally, no need to get involved with the different car immobilizer designs, the system will control the bypasser without the need of knowing which car model it is controlling.

For prototyping on the Accord. The 'HONDA-SL3' bypasser by Fortin Electronics is used.

The bypasser must be programmed to learn the code of the car key. Instructions on how to program each bypasser is available on the manual of each bypasser model per car.

Figure 7 'Honda-SL3' immobilizer bypasser by Fortin Electronics

Bypasser Connections

In addition to the power inputs and the ignition signal, the bypasser will tap into the data line to output the key signal while remotely starting, and will also act as a bridge cutting the signal from the security lamp if the car is started without the key. The security lamp has no effect on the remote starting procedure after enabling the bypasser, but it will show a flashing lamp in the car instrument panel, which can affect the user experience, therefore it is recommended to cut that signal whenever the car is started remotely until it is stopped.

Figure 8 shows the connections of the bypasser to the car and to the microcontroller. If a bypasser from a different company is used, the connections may slightly vary and the wiring colors coming from the bypasser may not be the same.

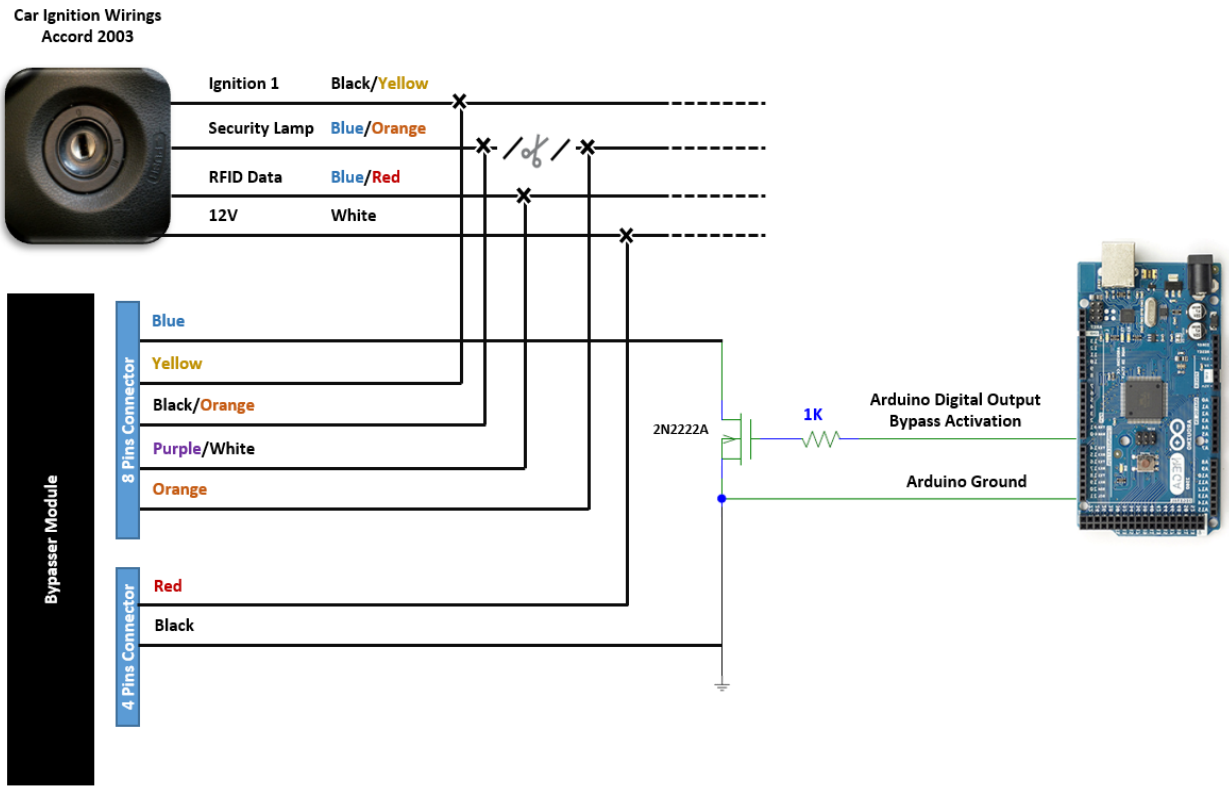

Figure 8 Immobilizer bypasser connections to the microcontroller and the Accord

b. Engine Starting Software Design

The software part plays a major role in automating the process of starting the engine remotely. The relays will be driven according to the connections shown in Table 1 for each mode. Time delay is used between each mode to simulate the process of starting the engine by a human.

The process of starting the engine while it is completely off is as the following:

- Disable the security system of the car
- Enable the bypasser module
- Set the ignition mode to ON by powering the ACC, IGN1 and IGN2 relays. No need to go to the ACC mode first.
- An optional action of relocking the doors can be done if the engine-starting request is remote.
- Most gas cars can be started immediately after setting the ignition to ON unlike diesel cars, but it is good to wait for the fuel pump to prime the fuel rails for a smoother engine cranking especially in the winter.
  The system will wait for 6.9 seconds. This time was estimated based on the sound of the fuel pump of the Accord.

- The system will set the ignition mode to the START mode by powering the IGN1 and START relays only. Other relays need to be powered off. The system should keep the car in the ignition mode for a period of ~960ms, which is enough to crank the engine and get the car started.
- After starting the car, the system should set the ignition mode of the car to the ON mode as long as the car is running without a key.

It is important not to attempt starting the engine while the engine is already running by the key switch; this could result in harming the starter motor of the car if not more components. In addition, the system should not attempt starting the engine if the key is present on the ON mode of the key switch even if the engine is not running, because the line IGN2 will get power from the key switch while the system is in the START mode which violates the original design modes of the car electrical system shown in the table before.

These two scenarios can be avoided by checking for the IGN1 line of the car before attempting to start the engine. Having a high value of 12V in the wire indicates that either the car is running or the key switch is on the ON mode, and the engine-starting request should be aborted.

Appendix A contains the Arduino code written to illustrate the automation process of starting the engine and controlling the locks of the car and the security system. The other part of the code related to parts not documented in this report such as Bluetooth communication still not finalized and under the progress of development.

Appendix A - Arduino Code

```
#include <avr/sleep.h>
#include <avr/power.h>

//pins
int ign[] = {2,3,4,10,8}; //acc,ign,ign2,st, bypass
int control[] = {6,7,9, 40, 50}; //arm,disarm, horn, lock, unlock
int input[] = {47,51, 53}; //brake, ignition, doors

//Bluetooth Communication
int blue[] = {20}; //Connection

//Serial
int buffer1[10];
int status1 = 0;
int pointer1 = 0;

//PWD
unsigned long mil = 0;

//Status
int ignStatus = 0;
boolean bypass = false;

//millis
int ignledm=0;

boolean BluetoothConnected()
{
  return (digitalRead(blue[0]) == HIGH);
}

void sendBluetooth(String msg)
{
  if (BluetoothConnected())
       Serial1.print(msg);
}

void setup()
{
  //Prepare ports
    setOutput(ign,5);
    setOutput(control,5);
    setOutput(interface,1);
  //Prepare Serial pins
    Serial1.begin(9600); //Bluetooth
  //Prepare Inputs
    setInput(input,3);
  //Prepare Interrupts
    pinMode(blue[0],INPUT);//BlueInterrupt
    attachInterrupt(3,blueInterrupt,CHANGE); //Interrupt 3 -> Pin 20
  //System Setup Completed
    sendPulse(interface[0] , 200); //Starting buzz
}

//Bluetooth connection status change
```

```
56  void blueInterrupt()
57  {
58    if (BluetoothConnected() == false)
59      //Reset power down timer
60      mil = millis();
61    else
62    {
63      //Connection established!!
64      sendBluetooth("Bluetooth connection established!");
65    }
66  }
67
68  //Array of pins fast output setting
69  void setOutput(int arr[] , int len)
70  {
71    for (int i = 0 ; i < len ; i++)
72      {
73        pinMode(arr[i],OUTPUT);
74        digitalWrite(arr[i] , LOW);
75      }
76
77  }
78
79  //Array of pins fast input setting
80  void setInput(int arr[] , int len)
81  {
82    for (int i = 0 ; i < len ; i++)
83      pinMode(arr[i],INPUT);
84  }
85
86  //Car Engine Status
87  boolean eng = false;
88  boolean engineRunning()
89  {
90    if (digitalRead(input[1]) == HIGH)
91    {
92      eng = true;
93    } else
94    {
95      if (eng == true) //Safe check before deciding that engine has stopped running
96      {
97        delay(800);
98        eng = (digitalRead(input[1]) == HIGH);
99      }
100   }
101   return eng;
102 }
103
104 void loop()
105 {
106  if (Serial1.available() > 0)
107    readSerial();
108  if (BluetoothConnected() == false && ((millis() - mil) > 11000  || (millis() < mil)) &&
      engineRunning() == false)
109  {
```

```
    //System power down for power consumption
     pwd();
     delay(100);
     //System wake up
     mil = millis();
   }

   //If turning the car off by key and the engine was remotely started and bypassed at
   first
   //keep bypasser on
   //check for opposite scenario
   if (bypass == true && engineRunning() == false)
     setBypass(false);
  }

  //Microcontroller Power Down Mode
  void pwd()
  {
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sleep_enable();
    sleep_mode();
    //PWD
    sleep_disable();
  }

  void readSerial()
  {
    buffer1[pointer1++] = Serial1.read();
    switch (status1)
    {
      case 0:
        int val[3];
        val[0] = 'b'; val[1] = 'g'; val[2] = 'n';
        if (buffer1[pointer1-1] != val[pointer1-1])
        {
          pointer1 = 0;
        } else if(pointer1 == 3)
        {
          pointer1 = 0;
          status1 = 1;
        }
        break;
      case 1:
        if (pointer1 >= 4)
        {
          pointer1 = 0;
          processSerial();
        }
        break;
    }
  }

  void processSerial()
  {
    int val[4];
```

```
      //Non-Authorised orders:
      val[0] = 'a'; val[1] = 'r'; val[2] = 'm'; val[3] = '0';
      if (checkValue(buffer1 , val , 4))
        {
           setArm(0);
        }
      val[0] = 'a'; val[1] = 'r'; val[2] = 'm'; val[3] = '1';
      if (checkValue(buffer1 , val , 4))
        {
           setArm(1);
        }
      val[0] = 'i'; val[1] = 'g'; val[2] = 'n'; val[3] = '0';
      if (checkValue(buffer1 , val , 4))
        {
           setIgn(0);
        }
      val[0] = 'i'; val[1] = 'g'; val[2] = 'n'; val[3] = '1';
      if (checkValue(buffer1 , val , 4))
        {
           setIgn(1);
           sendBluetooth("Car Mode = ACC/");
        }
      val[0] = 'i'; val[1] = 'g'; val[2] = 'n'; val[3] = '2';
      if (checkValue(buffer1 , val , 4))
        {
          if (engineRunning())
          {
            setIgn(2);
            sendBluetooth("Car Mode = ON/");
          }
          //else //marked to avoid alarm
          //{
          //  setBypass(true);
          //  setIgn(2);
          //}
        }
      val[0] = 'e'; val[1] = 'n'; val[2] = 'g'; val[3] = 's';
      if (checkValue(buffer1 , val , 4))
        {
            if (engineRunning() == false && ignStatus < 2)
            {
              broadcast("Starting Engine.../");
              horn(85);  //horn
              setArm(1); //Disarm
              delay(100);
              startEngine(true);
              broadcast("Engine Started!!/");
            } else
            {
              //reply
            }
        }
      val[0] = 'h'; val[1] = 'r'; val[2] = 'n'; val[3] = '1';
      if (checkValue(buffer1 , val , 4))
        {
```

```
219             horn(1000); //horn
220           }
221         val[0] = 'h'; val[1] = 'r'; val[2] = 'n'; val[3] = '0';
222         if (checkValue(buffer1 , val , 4))
223           {
224             horn(85); //horn
225           }
226         val[0] = 'a'; val[1] = 'n'; val[2] = 'l'; val[3] = '0';
227         if (checkValue(buffer1 , val , 4))
228           {
229             Serial1.print(analogRead(A0));
230             Serial1.print("/");
231           }
232         val[0] = 'a'; val[1] = 'n'; val[2] = 'l'; val[3] = '1';
233         if (checkValue(buffer1 , val , 4))
234           {
235             //Serial1.print(analogRead(analog[1]));
236             delay(10);
237             //Serial1.print(analogRead(analog[1]));
238             Serial1.print("/");
239           }
240
241         //Authorised order:
242         //Finalization
243         status1 = 0;
244      // }
245    }
246
247    void setArm(int st)
248    {
249      sendPulse((st ==1) ? control[1] : control[0]);
250    }
251
252    void setLock(int st)
253    {
254      sendPulse((st ==1) ? control[4] : control[3]);
255    }
256
257    void horn(int duration)
258    {
259      sendPulse(control[2],duration);
260    }
261
262    void sendPulse(int pin, int duration)
263    {
264      digitalWrite(pin,HIGH);
265      delay(duration);
266      digitalWrite(pin,LOW);
267    }
268
269    void sendPulse(int pin)
270    {
271      sendPulse(pin,175);
272    }
273
```

```
274  void setBypass(boolean st)
275  {
276     bypass = st;
277     if (bypass == true)
278        digitalWrite(ign[4],HIGH);
279     else
280        digitalWrite(ign[4],LOW);
281  }
282
283  void startEngine(boolean lock)
284  {
285    switch (ignStatus)
286    {
287      case 0:
288      case 1:
289        setBypass(true);
290        setIgn(2);
291        delay(1000);
292        if (lock == true)
293         setLock(0);
294        delay(5900);
295      case 2:
296        setIgn(3);
297        delay(960);
298        setIgn(2);
299        break;
300      default:
301        break;
302    }
303  }
304
305  void setIgn(int mode)
306  {
307    switch(mode)
308    {
309      case 0: //Ignition OFF
310        digitalWrite(ign[3],LOW);
311        digitalWrite(ign[2],LOW);
312        digitalWrite(ign[1],LOW);
313        digitalWrite(ign[0],LOW);
314        delay(500);
315        //Bypasser might still be needed
316        if (engineRunning() == false)
317           setBypass(false);
318        break;
319      case 1: //Ignition ACC
320        digitalWrite(ign[3],LOW);
321        digitalWrite(ign[2],LOW);
322        digitalWrite(ign[1],LOW);
323        digitalWrite(ign[0],HIGH);
324        delay(500);
325        //Bypasser might still be needed
326        if (engineRunning() == false)
327           setBypass(false);
328        break;
```

```
      case 2: //Ignition ON
        digitalWrite(ign[3],LOW);
        digitalWrite(ign[2],HIGH);
        digitalWrite(ign[1],HIGH);
        digitalWrite(ign[0],HIGH);
        break;
      case 3: //Starter motor
        digitalWrite(ign[3],HIGH);
        digitalWrite(ign[2],LOW);
        digitalWrite(ign[1],HIGH);
        digitalWrite(ign[0],LOW);
        break;
    }
    ignStatus = mode;
  }

  boolean checkValue(int arr[] , int chk[] , int len)
  {
    for (int i = 0 ; i < len ; i++)
      if (arr[i] != chk[i])
        return false;
     return true;
  }
```