Comprehensive Document

Timothy Myung

Richard Vu

Dave Edwards

Leora Juster

Suren Abrahamyan

Jie Chen

Faculty Advisor - Russ Abbott

Graduate Advisor - Eric Liao

Table of Contents

Section 1: Executive Summary

EcoCAR2 is a three-year competition sponsored by the US Department of Energy and General Motors. The competition challenges 15 universities across North America to reduce the environmental impact of a car without compromising performance, safety, and consumer acceptability. General Motors provided each university with a 2013 Chevrolet Malibu for the modification. The competition involves a collaboration between the College of Engineering, Computer Science, and Technology and the College of Business and Economics, with Engineering handling the design and implementation of the vehicle systems and the Business handling budgeting, fundraising and promotion of the program.

One of the competition's requirements is always to display vital diagnostic information in the car. To accomplish this, the physical radio and hvac buttons were removed from the center console of the car to install a touchscreen LCD.

Our Computer Science team was responsible for programming the display to show the vital diagnostic information to the user as well as creating functional replacement buttons for the physical buttons that were removed. Information and commands are transported from the GUI to the I.MX 6 board provided by Freescale. The board communicates with vehicle's components via the CAN bus, which is the controller area network for the vehicle's microcontrollers.

The GUI and application software are developed in Qt using Qt Creator which is a cross-platform development environment that uses C++, JavaScript, and QML.

Our goal was to not only implement the required features, but provide a user-friendly environment.

Section 2: Introduction

The Computer Science team was responsible for replacing the radio and climate control buttons of the vehicle with a touch screen running on a Qt platform, on the Freescale i.MX 6 Sabre ARM board hardware using a Linux kernel for the vehicle. The team delivered a functional version on May 2014. This prototype maintained all of the functions of the climate control and radio that they were replacing and displayed several required battery live data signals. The most vital responsibility of the prototype was to efficiently read and send CAN signals and user interaction with the display and car which the end-product delivered.

Section 3: User Guide

Home Page

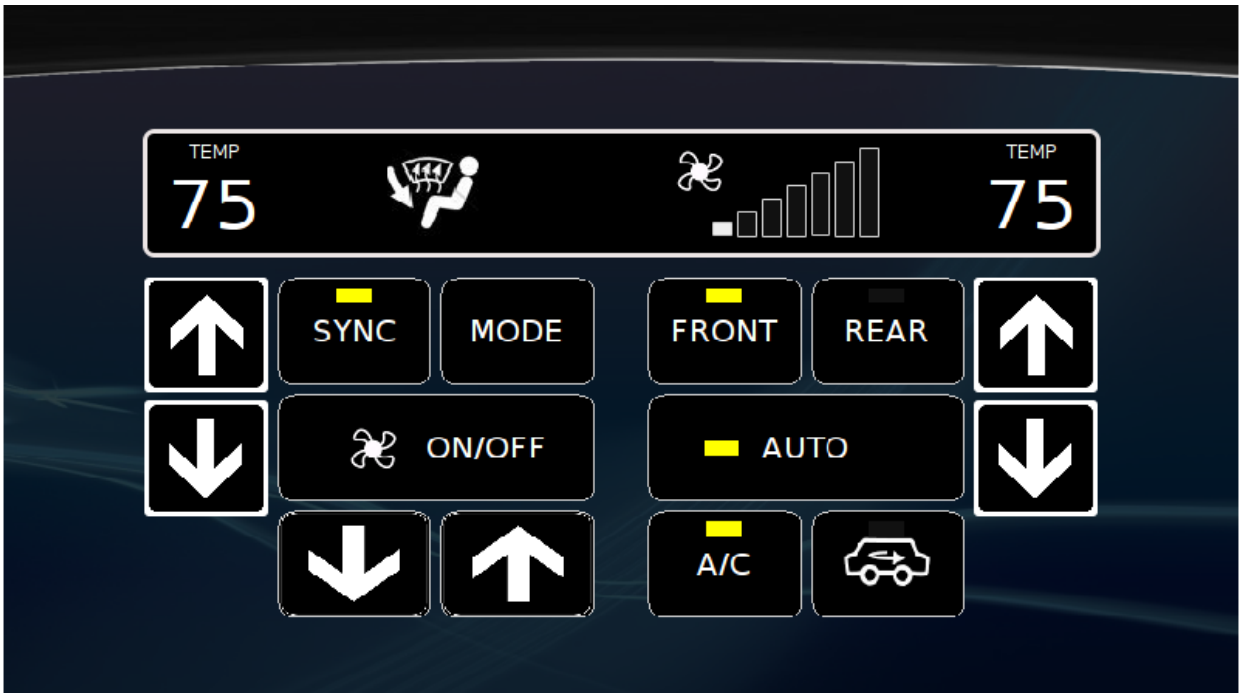
When the car is turned on, the welcome display screen appears. The user can click the top bar and a menu becomes available for the user to access any of the four following displays: Radio Settings, Air Conditioning Settings, Battery Signals, and Display Settings.

Navigation

Navigation can be done either by clicking on the menu or by using swipes.

- Swiping Right: Menu.
- Swiping Left: AC.
- Swiping Up: Diagnostic.
- Swiping Down: Radio.

A/C Page



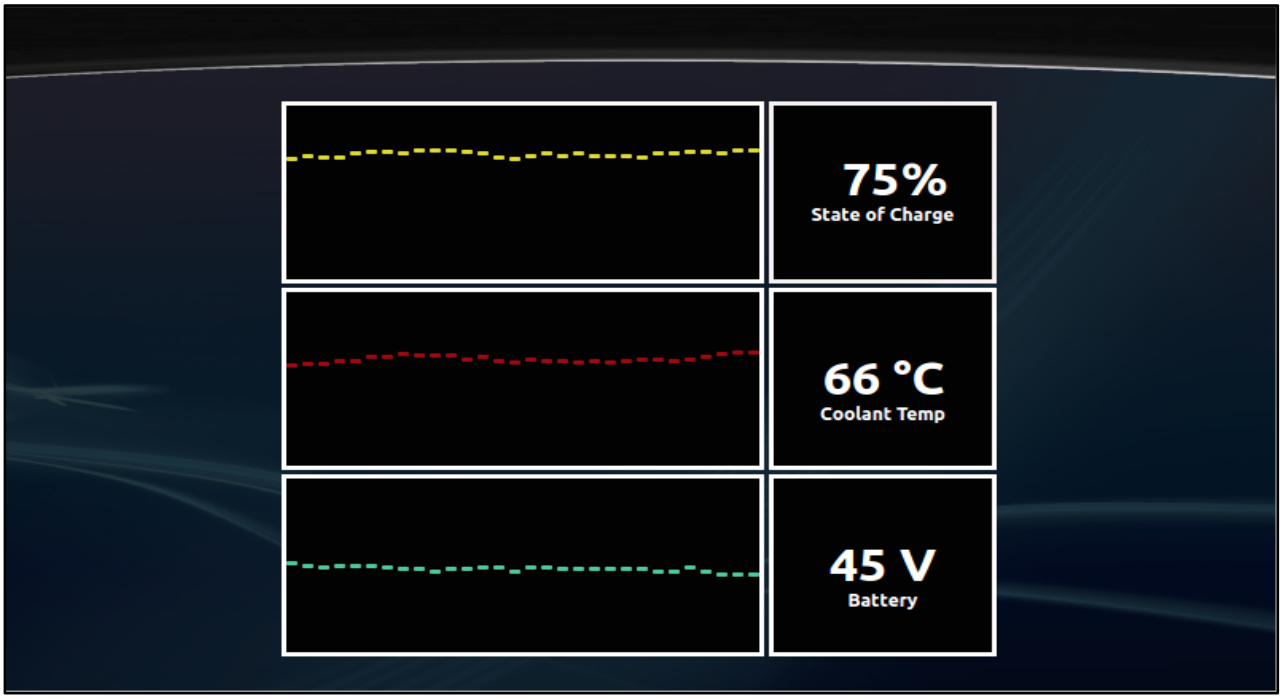
The A/C page contains all the physical HVAC buttons that were removed. The user can control fanspeed using the arrow buttons on the button or by clicking on the bars on the top. Clicking on “Mode” cycles through four different fan modes like using the top vents or bottom vents or combinations of them.

The user can have different temperatures for the driver and passenger seat. This can be changed using the left and right arrows. To quickly sync the two temperatures, the sync button can be pressed. The front and defrost buttons can only be pressed once every 5 seconds as per the car requirements.

Radio Page

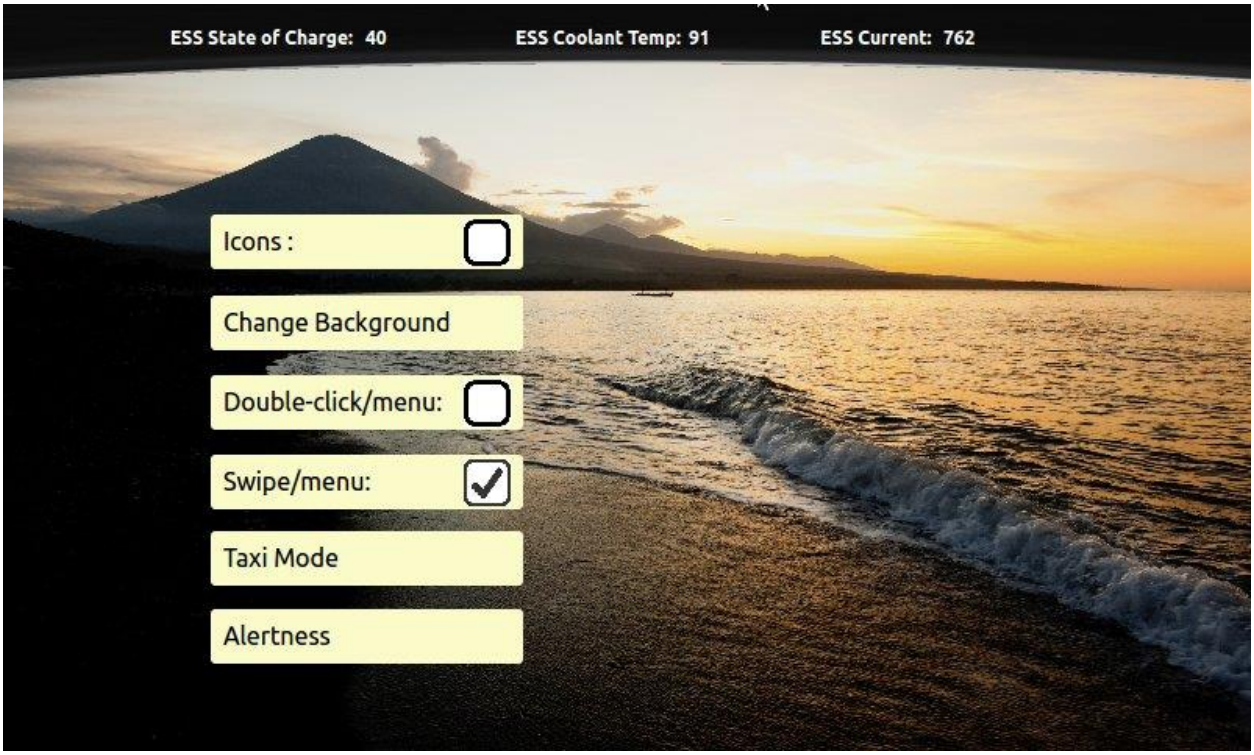
Although the physical radio panel that was removed contained more buttons, the Radio Page contains all the necessary buttons. A CD bay was removed from the car so buttons for those were not needed.

Diagnostic Page



The Diagnostic page displays three vital diagnostics (State of Charge, Coolant temperature and Battery Voltage). This is the primary requirement for our software. The graph updates every 5 seconds.

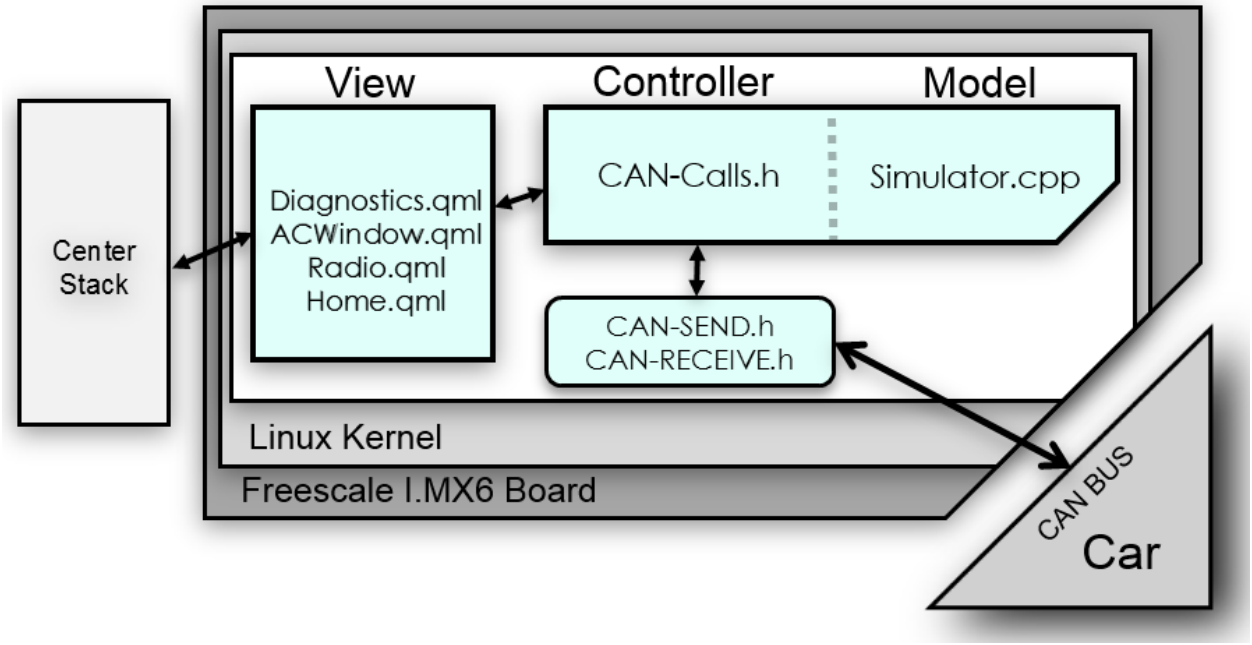
Settings Page



In the Settings Page, the user has many options to customize their display. They can choose between multiple display backgrounds. There is an option to toggle between text and icons on buttons. Swipe can be turned off. Taxi Mode can be turned on to track the drive. Alertness can test if the user is coherent enough to drive.

Section 4: Architecture and Design

The software was developed using Qt which is a cross-platform application framework generally used in phones and other mobile devices with touch interfaces. The team followed a model-view-controller architecture to complete the project.



Software Architecture

View

The view was built using Qt Modeling Language (QML). Each page (Radio, AC, Diagnostic, etc) was created using QML. QML offers many building blocks such as buttons, rectangles and text boxes. Because QML is a declarative language much like Javascript, actions such as swipe and animations were also implemented.

Controller

C++ and QML are used in the controller. The QML controller functions to navigate between the various pages. C++ is used to change information the settings and receive diagnostic information.

Model

Because the car was unavailable to the CS team for most of development, a simulator was created to mimic car statuses. This allowed the team to develop outside the laboratory where the physical motherboard resides. The team was able to perform integration testing between different components of the application due to simulator's flexibility. The development was further streamlined by templatizing and creating generic UI objects and behaviors that were used and applied across the application.

The software, now in the car, uses the simulator to store software settings.

Our software runs in the Chevy Malibu on the I.MX6 board donated by Freescale. A version of Linux was customized to run on the board using YOCTO. A connection to the car is established using C++ in Qt. Using Linux DOS commands, Qt can send and receive signals from the car using the vehicle’s Control Area Network (CAN) bus.

Section 6: Conclusions

Initially, the team attempted to develop an intricate system with multiple layers of frameworks and languages (Spring, AngularJS, HTML/CSS). While this allowed for more possibilities for integrating newly developed features and systems, we could not establish a connection to the vehicle's CAN bus.

Fortunately, we were finally able to connect to the car using Qt. Although disheartening, we scrapped everything and started from scratch using the Qt framework. This highlights one of the mistakes our team made in the project. We assumed we could connect our software to the hardware without knowing beforehand if it was possible.

For many of us on the team, this project marked the first time we developed software with hardware in mind. Many times we wanted to test our software but the hardware was unavailable. This is why creating the simulator was vital to our software development.

An issue the team had was that because of hardware limitations, the Qt framework that was implemented on the board was a lower version than what is currently available. This meant we didn't have access to many libraries available online. Because of this, we needed to code our functions for features such as swipe and displaying graphs.

This project was helped by Abdul Shabana who made the initial connection to the CAN bus. The support at Freescale was more than willing to offer help with any issues we had with their i.mx6 board. Jessica Mita designed the icons for the AC page.

As this is the final year of the EcoCAR2 competition, it is unlikely the project will be continued. However, as CSULA will be entering EcoCAR3, a few notes can be made for future

students on the EcoCAR CSULA team. For one, it is very important to start with what works and building from that. Our team inherited last year's project that was written in QNX but that didn't work in the car. Then, our team mistakenly used a combination of AngularJs/Spring that also didn't work and it ended up costing us valuable time.

It is very important to do testing as soon as possible. As the car was getting ready to be shipped, more teams needed access to the car so we had less time to work directly with the car. Luckily, most of our functions worked once the CAN connection was established but still, we would have liked more time to test.

Because this project involved a lot of different areas of expertise such as CS, Mechanical Engineering and Electrical Engineering, it was very important to build communication between all groups. Because of the many moving the parts, the project moved slowly.