Reverse Engineering and Enhancement of Automobile Infotainment System

Cole Brinsfield
Senior Project
ELECTRICAL ENGINEERING DEPARTMENT
California Polytechnic State University
San Luis Obispo
2022
## Table of Contents

Lists of Tables and Figure ........................................................................................................ iv

Abstract .................................................................................................................................. 1

Introduction ............................................................................................................................ 2

Background .............................................................................................................................. 3

Android Auto .......................................................................................................................... 3

Controller Area Network (CAN) ................................................................................................. 4

A Quick Guide to Vehicle CAN Bus Reverse Engineering .......................................................... 5

Reverse Engineering the CAN Bus Using Live Data ................................................................. 5

Reverse Engineering the CAN Bus by Playing Back Recorded Data ......................................... 7

RGBS Analog Video and its Relationship with VGA Video ......................................................... 8

Requirements ......................................................................................................................... 10

Design .................................................................................................................................... 11

Functional Decomposition ........................................................................................................ 11

  Level 0 Decomposition ......................................................................................................... 11

  Level 1 Decomposition ......................................................................................................... 12

Development ............................................................................................................................ 14

Determining the Correct Display Protocol ............................................................................. 14

Outputting RGBS from a Raspberry Pi ..................................................................................... 15

  Outputting Analog Video .................................................................................................... 15

  Outputting a Combined Sync Signal .................................................................................. 17

  Outputting at 15kHz ............................................................................................................ 18

Injecting an External Signal into the Stock Display ................................................................. 20

Decoding the CAN Bus of an Infiniti G37 .............................................................................. 25

  Headlight and Door Status .................................................................................................. 26

  Climate Data ....................................................................................................................... 27

  TPMS Data .......................................................................................................................... 28

  Engine Status ...................................................................................................................... 29

Software .................................................................................................................................. 30

Integration ............................................................................................................................... 33

  Wiring .................................................................................................................................. 33

  Power .................................................................................................................................. 33

  Adding a Touchscreen ........................................................................................................ 34
Lists of Tables and Figure

Table I Engineering Specifications ................................................................. 10
Table II Engineering Specifications ................................................................. 11
Table III Infotainment System Functional Breakdown .................................... 12
Table IV Module 1.1 Functional Breakdown .................................................. 13
Table V Module 1.2 functional breakdown ..................................................... 13
Table VI Module 1.3 functional breakdown .................................................... 13
Table VII Module 1.3 Functional Breakdown ................................................ 14
Table VIII CAN Message Functionality Map ............................................... 25
Table IX Headlight and Door Data Frame ...................................................... 26
Table X HVAC Status Data Frame ................................................................. 27
Table XI Climate Temperature Data Frame ................................................... 28
Table XII TPMS Data Frame ......................................................................... 28
Table XIII Engine Status Data Frame .......................................................... 29
Table XIV Steering Wheel Controls Functionality Map .................................... 36

Figure 1: Android Auto communication scheme .......................................... 3
Figure 2: A standard CAN Data Frame ............................................................ 4
Figure 3: cansniffer being used to view live CAN Bus data. Red coloring indicates a change in data .... 5
Figure 4: Using live data to reverse engineer locking Data Frame .................. 6
Figure 5: Using a CAN log to reverse engineer door unlocking [2] ................. 7
Figure 6: VGA Video Timing Diagram [8] ....................................................... 8
Figure 7: Level 0 Decomposition of Infotainment .......................................... 11
Figure 8: Level 1 Decomposition of Infotainment ......................................... 12
Figure 9: Technical Service Manual Expected Red Signal [9] .......................... 14
Figure 10: Technical Service Manual Expected Sync Signal [9] ..................... 15
Figure 11: Tomi Engdahl's Sync Combining Circuit [12] ............................... 17
Figure 12: Circuitry for outputting RGBS from a Raspberry Pi ................. 18
Figure 13: Boomerang, a PCB equipping a Raspberry Pi with RGBS Analog Video Output .... 19
Figure 14: Boomerang Pinout ................................................................. 20
Figure 15: Pinout of G37 Display Harness, viewed as looking towards the back of the display .... 21
Figure 16: Circuit capable of switching input sources for the stock display .......... 22
Figure 17: The G37 Screenslaver, a PCB which allows for external input to the stock display .... 23
Figure 18: Screenslaver Pinout ................................................................. 23
Figure 19: The G37 Screen Slaver installed in the stock display, replacing the original connector .... 24
Figure 20: A "DIY" PCB compatible connector ................................................. 24
Figure 21: Infotainment Software Stack ......................................................... 30
Figure 22: Dash main page ......................................................................... 30
Figure 23: Dash with Android Auto running, controlling media .......... 31
Figure 24: Dash with Android Auto running, navigation ........................... 31
Figure 25: Dash plugin for Infiniti G37 provides tire pressure and car status .... 32
Figure 26: Dash plugin for Infiniti G37 providing climate control information ........................................ 32
Figure 27: Replacement Infotainment High Level Wiring Diagram ............................................................ 33
Figure 28: Mechanical Stackup of Stock G37 Display ................................................................................. 34
Figure 29: Resistive Ladder of Steering Wheel Controls [16] .................................................................. 35
Figure 30: Steering Wheel Controls ........................................................................................................ 35
Figure 31: Steering Wheel Interfacing Circuit ............................................................................................ 36
Abstract

In 2011, luxury cars were released with displays built into the center. With modern computing power and technology such as Android Auto or Apple CarPlay, these older infotainment systems degrade the value of cars that are otherwise comparable to 2021 models. This project upgrades the infotainment system of one such car, a 2011 Infiniti G37. Through use of an additional microcomputer in the car, an infotainment upgrade can be achieved. This upgraded infotainment adds navigation, wireless audio, and data display capabilities to the car, while maintaining the functionality of the stock infotainment. While customized for the 2011 Infiniti G37, the methods and techniques used to develop this project could be applied to other similar vehicles. Outside of the addition of an external switch, this infotainment upgrade is otherwise seamlessly integrated into the car.
Introduction

The 2011 Infiniti G37 features an approximate 7” display in the center console of the car. This display is used for basic car configuration, such as maintenance reminders and automatic headlight sensitivity, as well as media controls, such as radio. When compared to more modern cars, the infotainment capabilities of the G37 are lacking. This project aims to modernize the infotainment capabilities of the G37, without interrupting the stock capabilities of the car. This infotainment upgrade should serve to enhance the driving experience of the G37, providing navigation, wireless audio, and data displays to the user (driver).

The display in the G37 is capable of RGBS (Red, Green, Blue, Sync) and composite video as described in reference [2]. Composite is used by the reversing camera of the car, whereas the RGBS input is used for the stock infotainment. RGBS video is a four-channel video protocol that uses analog red, green, and blue channels, and a combined horizontal and vertical sync signal that runs at 15 kHz [2]. RGBS shares similarities with VGA video, except for a combined sync signal and slower clock speed. A microcomputer, such as a raspberry pi, is capable of outputting VGA video and is a candidate for hosting replacement infotainment software [4]. By combining the horizontal and vertical sync signals using a logical XOR, and running the VGA output at 15khz, a pseudo-RGBS output can be generated from a microcomputer [3]. Touchscreen capabilities can be added to the stock display using a capacitive touchscreen overlay, wired into the microcomputer of use.

An additional microcomputer’s display signal can be injected into the stock screen of the G37 using a video MUX, such as the TS5V330 [6]. A MUX such as the TS5V330 would preserve the stock capabilities of the car, by allowing an external input to select the display signal being fed to the screen. With the exception of an added external input (switch), all wiring can be done internal to the car, out of the view of the user.

The Infiniti G37 has an internal network between car modules named a CAN Bus [1]. By tapping into this bus, the infotainment upgrade can listen in on data previously not exposed to the user. This data could include information that ranges from current speed of the vehicle, to tire pressure and headlight status. While this data is undocumented, reverse engineering is possible [2].

Android Auto is a candidate for adding wireless audio and navigation to the upgraded infotainment. Android Auto, while not open source, has been partially reverse engineered in the past [8]. Recent updates to Android Auto have added wireless capabilities, allowing the use of the head unit without plugging in a user’s phone. As a backup, Bluetooth audio facilitated through a microcomputer could be used as a wireless audio solution.
Background

Android Auto
Launched in 2015 by Google, Android Auto is an Android interface designed for in car use. Android Auto has three main features: navigation, phone call support, and audio playback. Android Auto’s ultimate goal is to provide these core functionalities in a hands-free and distraction-less manner while a user is in control of a car [3]. When connected to a supported car, Android Auto acts as an external display and human-machine interaction (HMI) layer between the driver and their phone. That is, Android Auto is to an Android phone as a keyboard, mouse, and monitor are to a desktop computer. All the processing and computing is done on the Android phone, Android Auto is just a means of interface.

While Android Auto is closed source and only available to a specific set of first- and third-party infotainment manufacturers, there have been a few attempts to reverse engineer a working Android Auto implementation. These projects analyze Android Auto through decompilation of the application installed on an Android device (internally named “Gearhead”). These implementations have varying levels of functionality, but outline Android Auto’s internal structure.

![Android Auto communication scheme](image-url)
Android Auto primarily receives video and audio streams and sends user input (both physical and audio) back to the Android device. There are a few bidirectional communication paths between the Android device and Android Auto, in the form of services. Services are responsible for the transport and processing of miscellaneous data, such as reporting to Android the current headlight status (which is used to determine if Android Auto should present itself in a dark or light theme). Note that the only interaction with the Android device, that is not directly done through Android Auto, is the processing of phone call audio; phone call audio is processed directly by a HFP (Hands Free Profile) Bluetooth device, such as a separate system within a vehicle’s stock infotainment device. The reasoning behind this is not entirely clear, though it could be that this is done for a compatibility reason.

Controller Area Network (CAN)

The Controller Area Network (CAN) Bus is a communications protocol for low-cost, high-speed serial communication. CAN is resilient to messaging faults and conflicts, which when combined with its low implementation cost, makes it ideal for a real time environment such as for communication within automobiles [1]. Most modern vehicles have at least one CAN bus within them, facilitating communication between modules in a vehicle as cars become increasingly technologically complex.

A standard CAN message (otherwise known as a data frame) is composed of seven data fields. Most of these data fields are abstracted away at a lower level by CAN transceivers (the circuit which handles reading and writing to the CAN bus).

![Figure 2: A standard CAN Data Frame](image)

The important fields for an end user are the identifier (11 bits) and data (0-8 bytes) fields. Within a CAN bus, an identifier is typically used to distinguish modules (for instance, to distinguish between the body control module in a car, and the engine control module), or to distinguish “groups” of messages (for instance, all messages that update the gauge cluster of a car might share a common identifier). In most
configurations, a CAN bus is a broadcast network— that is, all frames are visible to all devices attached to the network. For this reason, it is usually a trivial matter to tap into a CAN bus and monitor all communication between modules within. Deciphering the frames received, however, is another matter.

A Quick Guide to Vehicle CAN Bus Reverse Engineering

Given a vehicle that utilizes the CAN Bus, there are two main means of reverse engineering messages within the stream of data running across the wires. Take for instance, the scenario of trying to find the message on a vehicle’s CAN Bus that controls the locking and unlocking of the driver’s side door.

Reverse Engineering the CAN Bus Using Live Data

The first means of identifying the proper message on the CAN Bus is to utilize the live stream of data. This method requires software such as “cansniffer” [4] or “Kayak” [5]. Such software filters the incoming stream of CAN messages such to only display to the end user messages which have changing data. These software kits also accept manual filtering of Data Frame identifiers.

![Figure 3: cansniffer being used to view live CAN Bus data. Red coloring indicates a change in data.](image-url)
In the case of trying to identify the message that controls the lock status of the driver’s side door, one would follow a simple loop.

Figure 4: Using live data to reverse engineer locking Data Frame
Reverse Engineering the CAN Bus by Playing Back Recorded Data

It is not always feasible to utilize live data to identify a specific Data Frame. In such a case, an alternate method presents itself. Through recording the CAN Bus activity during an action of interest, and then playing back the recorded data in chunks, it is possible to identify the Data Frame of interest.

Take again the scenario of identifying the Data Frame that controls the locks on the driver’s side door.

First, record a chunk of CAN Bus data that contains within it a segment of time in which you cause the door to unlock. This can be done with a tool such as “candump” [6]. Then, play back chunks of the logged messages through a tool such as “canplayer” [7]. An efficient means of doing this is by using a “binary search” – play half of the recorded log at once, identifying which half unlocks the door. Break that half into halves and repeat the process. Eventually you will deconstruct the log into discrete messages and be able to identify which Data Frame unlocks the door. This process is outlined graphically below.

![Diagram](image.png)

*Figure 5: Using a CAN log to reverse engineer door unlocking [2]*
RGBS Analog Video and its Relationship with VGA Video

Analog video, as the name implies, operates using varying values of voltage to represent different data. Want to send the value “40”? With analog video, this can be as simple as sending out a 40mV signal. As one might suspect, the term “analog video” applies to a wide range of video transmission standards and technology.

Though it is becoming less common, VGA video remains one of the most widespread forms of analog video. The blue d-sub connector of a VGA cable is burned into the memory of all those who were into computers in the early 2000s. VGA video transmits analog video over five signaling wires, Red, Green, Blue, Horizontal Sync, and Vertical Sync.

Figure 6: VGA Video Timing Diagram [8]

A video frame (a single image of which video is composed of) is transmitted pixel by pixel, left to right, top to bottom. The color signals (Red, Green, Blue) output analog values the color contained in each pixel, while the horizontal and vertical sync signals output a pulse to mark the end of a row, or end of a frame, respectively. VGA typically outputs this data at horizontal clock speed (the speed of each pixel data in a row) of 31.469 kHz.

VGA is an evolution of all the RGB analog video standards that came before it. For this reason, many of the values VGA uses as part of its standard are carryovers from the older standards.
One such older standard, is RGBS.

RGBS (Red, Green, Blue, Sync) video is like the father of the more familiar and modern VGA video. While VGA brought analog video to a standardized consumer-focused protocol, RGBS powered arcade machines. RGBS video operates on the same principle but has the vertical and horizontal sync pulses combined into one composite sync signal (which can be thought of like an XOR of the individual sync pulses). RGBS video also runs at a horizontal frequency of 15.6 kHz, roughly half that of VGA.

But there are more similarities between RGBS and VGA than differences, with properties such as color channel load impedance and voltage range being equivalent. It is plausible, then, that technology created to output VGA signals, can be adapted to output RGBS video. Simply combine the sync signals, and slow down the output clock frequency, and it should work.
### Requirements

**TABLE I**  
**ENGINEERING SPECIFICATIONS**  
**JUSTIFICATION**

<table>
<thead>
<tr>
<th>CUSTOMER REQUIREMENTS</th>
<th>ENGINEERING SPECIFICATIONS</th>
<th>JUSTIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>H, E</td>
<td>1. System runs off the car battery (~12-14V DC)</td>
<td>This allows the system to be powered off the car, as opposed to an external battery that would need to be charged, as well as allows wiring to remain internal to the car, preventing a change in looks.</td>
</tr>
<tr>
<td>E, C</td>
<td>2. System tracks vehicle ignition status, and powers on/off appropriately.</td>
<td>By powering on only when the car’s ignition is on, and powering off otherwise, this prevents battery drain while the car is off.</td>
</tr>
<tr>
<td>A, F</td>
<td>3. Has capacitive touchscreen</td>
<td>Touchscreen is required for optimal android auto support</td>
</tr>
<tr>
<td>B</td>
<td>4. Supports Bluetooth Audio</td>
<td>Bluetooth is the most typical way of supporting wireless audio from phones</td>
</tr>
<tr>
<td>D, E</td>
<td>5. Allows switching between stock infotainment and upgraded infotainment displays</td>
<td>In order to preserve stock car functionality, there must be a means of allowing the stock infotainment to be displayed</td>
</tr>
<tr>
<td>G</td>
<td>6. Has CAN Bus capabilities</td>
<td>Internal vehicle data is transmitted on CAN Bus.</td>
</tr>
</tbody>
</table>

**LIST OF CUSTOMER REQUIREMENTS:**
A) SUPPORTS ANDROID AUTO  
B) WIRELESS AUDIO SUPPORT  
C) DOESN’T AFFECT STANDBY CAR BATTERY LIFE  
D) USES STOCK DISPLAY IN CAR  
E) DOESN’T AFFECT NORMAL CAR OPERATION  
F) TOUCHSCREEN SUPPORT  
G) CAN VIEW TIRE PRESSURE  
H) DOESN’T AFFECT CAR LOOKS
## TABLE II
### ENGINEERING SPECIFICATIONS

<table>
<thead>
<tr>
<th>SPEC #</th>
<th>PARAMETER</th>
<th>TARGET</th>
<th>TOLERANCE</th>
<th>RISK</th>
<th>COMPLIANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Operating Voltage</td>
<td>12 VDC</td>
<td>+5 VDC</td>
<td>H</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>Minutes powered on after car engine turned off</td>
<td>5 Minutes</td>
<td>Maximum</td>
<td>L</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>Simultaneous Touch Points</td>
<td>5 Points</td>
<td>Minimum</td>
<td>L</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>Bluetooth Profiles</td>
<td>HFP, A2DP</td>
<td>Exact</td>
<td>M</td>
<td>I</td>
</tr>
<tr>
<td>5</td>
<td>Number of supported video inputs</td>
<td>2 inputs</td>
<td>Minimum</td>
<td>L</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>CAN Bus Speed</td>
<td>1Mbit/s</td>
<td>Maximum</td>
<td>L</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>Main Module Physical Size</td>
<td>5”x5”</td>
<td>Maximum</td>
<td>L</td>
<td>I</td>
</tr>
</tbody>
</table>

### Design

#### Functional Decomposition

**Level 0 Decomposition**

![Level 0 Decomposition Diagram](image)

*Figure 7: Level 0 Decomposition of Infotainment*
# INFOTAINMENT SYSTEM FUNCTIONAL BREAKDOWN

<table>
<thead>
<tr>
<th>MODULE</th>
<th>INFOTAINMENT SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUTS</td>
<td>• I2C input from touchscreen</td>
</tr>
<tr>
<td></td>
<td>• Enable signal from stock/replacement infotainment switch</td>
</tr>
<tr>
<td></td>
<td>• Audio input from microphone</td>
</tr>
<tr>
<td></td>
<td>• USB input from android device</td>
</tr>
<tr>
<td></td>
<td>• CAN input from vehicle</td>
</tr>
<tr>
<td></td>
<td>• Nominal 12V DC voltage from car battery</td>
</tr>
<tr>
<td></td>
<td>• Impedance from steering wheel controls resistive ladder</td>
</tr>
<tr>
<td>OUTPUTS</td>
<td>• Audio output to vehicle stereo</td>
</tr>
<tr>
<td></td>
<td>• CAN output to vehicle</td>
</tr>
<tr>
<td></td>
<td>• Display signals for vehicle display (RGBS)</td>
</tr>
<tr>
<td>FUNCTIONALITY</td>
<td>Replacement infotainment system for Infiniti G37. Given raw vehicle data, present</td>
</tr>
<tr>
<td></td>
<td>relevant data to user. Act as Android Auto client for attached Android device.</td>
</tr>
<tr>
<td></td>
<td>Integrate into steering wheel controls and stock vehicle display.</td>
</tr>
</tbody>
</table>

## Level 1 Decomposition

![Figure 8: Level 1 Decomposition of Infotainment](image-url)
### TABLE IV
**MODULE 1.1 FUNCTIONAL BREAKDOWN**

<table>
<thead>
<tr>
<th>MODULE</th>
<th>1.1</th>
</tr>
</thead>
</table>
| INPUTS | • CAN from vehicle  
         |   • USB from 1.3 |
| OUTPUTS| • CAN to vehicle  
         |   • USB to 1.3 |
| FUNCTIONALITY | Translates CAN to USB and USB to CAN; A USB CAN interface |

### TABLE V
**MODULE 1.2 FUNCTIONAL BREAKDOWN**

<table>
<thead>
<tr>
<th>MODULE</th>
<th>1.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUTS</td>
<td>• Impedance from vehicle steering wheel</td>
</tr>
<tr>
<td>OUTPUTS</td>
<td>• USB to 1.3</td>
</tr>
<tr>
<td>FUNCTIONALITY</td>
<td>Translates impedance from steering wheel resistor ladder to USB inputs</td>
</tr>
</tbody>
</table>

### TABLE VI
**MODULE 1.3 FUNCTIONAL BREAKDOWN**

<table>
<thead>
<tr>
<th>MODULE</th>
<th>1.3</th>
</tr>
</thead>
</table>
| INPUTS | • I2C from touchscreen  
         |   • Audio Signals from microphone  
         |   • USB Signals from Android device  
         |   • USB Signals from 1.1  
         |   • DC Voltage from car battery  
         |   • USB Signals from 1.2 |
| OUTPUTS| • Audio Signals to Vehicle Stereo  
         |   • GPIO Signals to 1.4  
<pre><code>     |   • USB Signals to 1.1 |
</code></pre>
<p>| FUNCTIONALITY | Hosts infotainment software, processes incoming data into display signals in the form of GPIO parallel interface. Hosts Android Auto, piping button presses, audio, and display info from/to Android Auto. |</p>
<table>
<thead>
<tr>
<th>MODULE</th>
<th>1.4</th>
</tr>
</thead>
</table>
| INPUTS   | • GPIO Parallel Data from 1.3  
|          | • Enable Signal |
| OUTPUTS  | • Display Signals to Vehicle Display |
| FUNCTIONALITY | When enabled, converts digital data from 1.3 to analog data for vehicle display. When disabled, does nothing. |

**Development**

**Determining the Correct Display Protocol**

To provide a replacement display signal for the Infiniti G37 display, the signal protocol needs to be determined. From the wiring diagram presented in the technical service manual, it is known that the display uses four signals: Red, Green, Blue, and Sync. Furthermore, the technical service manual provides an example of a “good” signal.

![Figure 9: Technical Service Manual Expected Red Signal](image-url)
From this expected waveform, it is determined that the video signal is analog in nature. Furthermore, by analyzing the example waveform for the sync signal, it is seen that there is a pulse approximately every 60 µs, or at 16kHz. Accounting for inaccurate measurement, the closest standardized video display frequency is 15kHz.

![Waveform Image]

*Figure 10: Technical Service Manual Expected Sync Signal [9]*

Of standard display protocols, RGBS video is the only one that matches a 15kHz clock speed, analog video, and four wire interface.

**Outputting RGBS from a Raspberry Pi**

There are three impediments to outputting RGBS from a Raspberry Pi: outputting analog values from a system designed for digital, outputting a combined sync signal, and outputting all the above at 15kHz.

**Outputting Analog Video**

The Raspberry Pi is designed to output a digital video signal in the form of HDMI, whereas the stock Infiniti G37 display takes in an RGBS (analog) video signal. Luckily, the raspberry pi also supports a Parallel Display Interface (DPI) as an alternate configuration on its General Purpose Input Output (GPIO) pins [10]. The Pi’s DPI interface by default operates in a six bit per color channel configuration, with a GPIO pin dedicated to each bit. Given that a byte is eight bits, in order to fit the same values into six bites, the lower two bits are dropped from the output. This means that to output a pixel that has a red value of 60, which has a binary value of 0b00111100, four of the GPIO pins dedicated to the red channel would be output HIGH, while the other two (the most significant bits) would be output LOW. With a bit of math, a resistive ladder can be built that allows each color channel to output an analog value corresponding to its digital output. Said resistor ladder takes the following form.
Given that RGBS (and VGA) have a 75 Ω load on each color channel and expect a 700mV signal to correspond to a full color channel value [11], as well as the Pi having a 3.3V logic HIGH voltage, the following equation can be used solve for the parallel equivalent impedance at full brightness.

\[ R_s = 3.3V \times \frac{(3.3V - 0.7V) \times 75\Omega}{0.7V} = 278\Omega \]

This impedance can be broken down to its parallel components by solving for the following equation, which considers the scaling factor of each bit.

\[ \frac{1}{R_s} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4} + \frac{1}{R_5} + \frac{1}{R_6} \]

Where:

- \( R_1 = x \)
- \( R_2 = 2x \)
- \( R_3 = 4x \)
- \( R_4 = 8x \)
- \( R_5 = 16x \)
- \( R_6 = 32x \)

Calculated out, individual resistances are found, which can be normalized to standard valued resistors. [12]
<table>
<thead>
<tr>
<th>Resistor</th>
<th>Calculated Resistance</th>
<th>Standardized Resistance</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>547.3 Ω</td>
<td>499 Ω</td>
</tr>
<tr>
<td>R2</td>
<td>1094.63 Ω</td>
<td>1 kΩ</td>
</tr>
<tr>
<td>R3</td>
<td>2189.25 Ω</td>
<td>2 kΩ</td>
</tr>
<tr>
<td>R4</td>
<td>4378.5 Ω</td>
<td>4 kΩ</td>
</tr>
<tr>
<td>R5</td>
<td>8757 Ω</td>
<td>8 kΩ</td>
</tr>
<tr>
<td>R6</td>
<td>17514 Ω</td>
<td>16 kΩ</td>
</tr>
</tbody>
</table>

Repeated for each color channel, this resistor ladder allows for analog video output from the Raspberry Pi.

Due to resistor manufacturing tolerances, as well as the standardized resistance conversion, it is a good idea to run each color channel through an additional series trim potentiometer to allow for post manufacturing color correction of each overall channel.

**Outputting a Combined Sync Signal**

A RGBS (and VGA) frame is output to the display one pixel at a time, left to right, top to bottom. For a VGA signal, the end of a row is indicated with a Horizontal Sync pulse, whereas the end of a complete frame is indicated with a Vertical Sync pulse. RGBS combines these sync pulses into one Combined Sync signal, which is essentially an Exclusive Or (XOR) of the Vertical and Horizontal Sync components. While a simple XOR gate would likely work in more modern displays to combine the discrete sync signals, a preferred sync combining circuit applies some applications of filtering and debounce to the outputted signal, in order to work with a wider variety of displays. A reliable sync combining circuit was created by Tomi Engdahl in 1993 [13].

![Figure 11: Tomi Engdahl’s Sync Combining Circuit](image)

This circuit produces a clean Combined Sync signal compatible with the Infiniti G37 display.
Outputting at 15kHz

The DPI of the Raspberry Pi is highly configurable. The “dpi_timings” line within the Raspberry Pi’s configuration file allows for customizing nearly every parameter that makes up the video output. Among these, the clock frequency of the output video is set through the “pixel_freq” parameter [10]. This allows for the setting of a 15 kHz clock signal (the correct clock for RGBS). An example working configuration for RGBS video to the G37 display from the Raspberry Pi is present below. This timing, along with configuring the correct clock speed, configures an output resolution of 480x234 pixels.

```
dpi_timings=480 1 31 35 64 234 1 12 10 10 0 0 0 60 0 9600000 1
```

With each individual impediment solved, the solutions can be combined into a schematic and PCB for the Raspberry Pi.

![Figure 12: Circuitry for outputting RGBS from a Raspberry Pi](image-url)
Figure 13: Boomerang, a PCB equipping a Raspberry Pi with RGBS Analog Video Output
Injecting an External Signal into the Stock Display

The wiring harness that plugs into the stock display of the G37 has a pinout that is documented within the Factory Service Manual. Said pinout is displayed below.
From this pinout, we can see that the RGBS signal from the stock infotainment runs along pins 17 (red), 6 (green), 18 (blue), and 19 (sync). If these pins were to be disconnected from the display, and the RGBS output from the Raspberry Pi substituted in, the Raspberry Pi should output to the display. Using this pinout, a 2:1 MUX can be added in, which would allow switching between the stock infotainment signals, and external (Raspberry Pi) signals.
The TS5V330D [14] MUX is selected for use in said switching circuit due to its design being based around RGB video signals. Due to the TS5V330D operating on a nominal 5V supply voltage, a L78L05CD13TR [15] voltage regulator is used to create a stable 5V source from Pin 2 of the harness (Inverter VCC, 12V). Through extensive research, it was determined that the stock harness is of the TE Connectivity TH/.025 Connector System series of connectors [16]. Specifically, the 1318917-1 (which mates with a 1376111-2 connector on the PCB of the display). Using an additional set of these mating connectors, a PCB can be created with the appropriate circuitry which plugs directly into the stock display, replacing the existing harness connector. This allows for the injection of an external signal into the stock display without any modification to the wiring harness in the vehicle (such as any cutting or splicing of wires). An external switch is used to toggle between the injected and stock display signals.

Figure 16: Circuit capable of switching input sources for the stock display
**Figure 17**: The G37 Screenslaver, a PCB which allows for external input to the stock display

**Figure 18**: Screenslaver Pinout

**Infiniti G37 Screenslaver**

PCB to facilitate injecting external video into an Infiniti G37 display

- Analog Video
- Car Connector
- Control
- Ground
- Power
However, there is no PCB mount version of the 1318917-1 connector, as it is intended for use with a wiring harness only. There only exists a wire crimp version of the connector. A PCB mount compatible version can be created by crimping the connector onto short lengths of solid core wire, which take the place of through hole pins that a (hypothetical) PCB mount version would include.
Decoding the CAN Bus of an Infiniti G37

A peek into the Infiniti G37 Factory Service Manual section on the LAN System shows a “CAN Communication Signal Chart” [9]. This chart outlines many of the messages present on the CAN Bus, and the modules that interact with them. While this chart is not all inclusive, it gives a good idea of the type of data that exists on the Bus. From this chart, we can pinpoint a few messages of interest that replacement infotainment software can benefit from. These signals are listed below.

<table>
<thead>
<tr>
<th>Message of Interest</th>
<th>Use in replacement infotainment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Headlight Status</strong></td>
<td>Given the G37 has automatic headlights, if the headlights are on, it can be assumed that it is dark outside. This can be useful as a means of triggering a switch between a Light and Dark theme of the infotainment.</td>
</tr>
<tr>
<td><strong>HVAC/Climate Status</strong></td>
<td>The G37 usually presents this information to the drivers/passengers on the stock display. If the stock system is to be replaced, this data will no longer be present to the end user unless displayed by replacement infotainment software as well.</td>
</tr>
<tr>
<td><strong>Engine Status</strong></td>
<td>Detecting the transition of Engine Off -&gt; On, or Engine On -&gt; Off could be useful for automatic pause / play of audio.</td>
</tr>
<tr>
<td><strong>Door Open Status</strong></td>
<td>Similarly, knowing when the doors open or close could also be useful information for automatic media playback control.</td>
</tr>
<tr>
<td><strong>TPMS Data</strong></td>
<td>Tire Pressure Monitoring System data could allow replacement infotainment to display tire pressures and alert end user when a tire is low on air.</td>
</tr>
</tbody>
</table>
Following the procedures detailed in *A Quick Guide to Vehicle CAN Bus Reverse Engineering*, these signals were deciphered as follows.

**Headlight and Door Status**

<table>
<thead>
<tr>
<th>ID</th>
<th>Byte</th>
<th>Bit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x60D</td>
<td>0x60D</td>
<td>A</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Rear Right Door</td>
<td>0: Closed, 1: Open</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Rear Left Door</td>
<td>0: Closed, 1: Open</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Front Right Door</td>
<td>0: Closed, 1: Open</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Front Left Door</td>
<td>0: Closed, 1: Open</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Running Lights</td>
<td>0: Off, 1: On</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Low Beams</td>
<td>0: Off, 1: On</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x60D</td>
<td>B</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Fog lights</td>
<td>0: Off, 1: On</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Right Turn Signal</td>
<td>0: Off, 1: On</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Left Turn Signal</td>
<td>0: Off, 1: On</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>High Beams</td>
<td>0: Off, 1: On</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>G</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TABLE X
HVAC STATUS DATA FRAME

<table>
<thead>
<tr>
<th>ID</th>
<th>Byte</th>
<th>Bit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x54B</td>
<td>A</td>
<td>7</td>
<td>Climate On/Off</td>
<td>0: On, 1: Off</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>Mode</td>
<td>0b100: Defrost + Legs</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td>0b001: Head</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td></td>
<td>0b010: Head + Feet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>0b011: Feet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>0b101: Defrost</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>Dual Climate Status</td>
<td>0b11: On</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td>0b00: Off</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>Recirculate Off</td>
<td>0: Recirculating Air, 1: Not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>Recirculate On</td>
<td>Recirculating Air</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>Fan Level</td>
<td>Decimal 0-7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### TABLE XI
**CLIMATE TEMPERATURE DATA FRAME**

<table>
<thead>
<tr>
<th>ID</th>
<th>Byte</th>
<th>Bit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x542</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td></td>
<td>Driver’s Side Temperature</td>
<td>Decimal, 60F-90F</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td>Passenger’s Side Temperature</td>
<td>Decimal, 60F-90F. Note that this field is only updated when Dual</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Climate is enabled</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### TPMS Data

### TABLE XII
**TPMS DATA FRAME**

<table>
<thead>
<tr>
<th>ID</th>
<th>Byte</th>
<th>Bit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x385</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td>Front Right Tire Pressure</td>
<td>Contains Tire Pressure in PSI * 4</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td></td>
<td>Front Left Tire Pressure</td>
<td>Contains Tire Pressure in PSI * 4</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td></td>
<td>Rear Right Tire Pressure</td>
<td>Contains Tire Pressure in PSI * 4</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td></td>
<td>Rear Left Tire Pressure</td>
<td>Contains Tire Pressure in PSI * 4</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>7</td>
<td>Front Right Pressure Valid</td>
<td>0: Data not Valid, 1: Data Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Front Left Pressure Valid</td>
<td>0: Data not Valid, 1: Data Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>Rear Right Pressure Valid</td>
<td>0: Data not Valid, 1: Data Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Rear Left Pressure Valid</td>
<td>0: Data not Valid, 1: Data Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Engine Status

## TABLE XIII
ENGINE STATUS DATA FRAME

<table>
<thead>
<tr>
<th>ID</th>
<th>Byte</th>
<th>Bit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x551</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D</td>
<td></td>
<td>Engine Status</td>
<td>Sequence is as follows: 0xA0: Engine Off, Ignition Pressed 0x20: Engine Turning On 0x00: Engine Turning On 0x80: Engine Running 0x00: Engine Shutting Down 0x20: Engine Off  So, a transition away from a 0x80 status indicates engine shutting off</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Software

OpenDash’s Dash project [17] is used as the primary infotainment software in the replacement infotainment system. Dash provides Android Auto functionality through a fork of OpenAuto [18], while implementing an additional graphical wrapper to support local audio playback, additional application integration, and a car specific extension API. To fully integrate with the Infiniti G37, a G37 specific plugin is written for Dash [19].

![Infotainment Software Stack](Figure 21: Infotainment Software Stack)

![Dash main page](Figure 22: Dash main page)
Figure 23: Dash with Android Auto running, controlling media

Figure 24: Dash with Android Auto running, navigation
Figure 25: Dash plugin for Infiniti G37 provides tire pressure and car status

Figure 26: Dash plugin for Infiniti G37 providing climate control information
The Raspberry Pi that powers the infotainment replacement requires a 5V source of power. With car batteries operating at marginally 12V, conversion is needed. A Mausberry 3A Car Supply [20] is chosen to provide this 5V power to the Raspberry Pi. This supply not only handles power conversion, but also provides “soft shutdown” capability to the Raspberry Pi, tracking the ignition state of the car. That is, when the car is turned on, the Mausberry supply will provide power to the Raspberry Pi. When the car is turned off, the Mausberry supply asks the Raspberry Pi to power off. Once the Raspberry Pi has safely
shutdown, the Mausberry supply stops providing power. This solution not only protects the car battery from being drained by an always on load, but also protects the data integrity of the Raspberry Pi, allowing it to shut down gracefully, instead of the equivalent of pulling out a power cable.

Adding a Touchscreen
The non-navigation trim of the Infiniti G37 comes with a display that does not feature a touchscreen. In order to fully utilize Android Auto, a touchscreen is recommended. Touch screen capabilities can be added to the stock display with a quick modification to the assembly.

The stock display has three basic layers that assemble to form the display unit.

![Figure 28: Mechanical Stackup of Stock G37 Display](image)

A capacitive touchscreen overlay, such as the lattepanda 7" touchscreen overlay [21] can be substituted for the screen protector in the mechanical stackup, resulting in a touchscreen capable display at the sacrifice of some durability. The touchscreen can then interface with the Raspberry Pi through i2c and can be calibrated for the underlying display geometry.
Steering Wheel Controls

The steering wheel controls of the G37 operate through a resistive ladder. Each button on the steering wheel corresponds to a difference in resistance seen by the stock infotainment unit.

For instance, a driver pressing “Source” would result in the infotainment unit seeing 0 Ω (a short) between pins 17 and 14, whereas pressing “Menu Down” would result in the infotainment unit seeing approximately 221 Ω between pins 17 and 14.

For this project, it makes the most sense to de-pin (remove from the stock connector) pin 14, which grants the ability to detect four buttons on the steering wheel, “Source”, “Menu Up”, “Menu Down”, and “Voice Assistant / Talk”.

Figure 29: Resistive Ladder of Steering Wheel Controls [9]

Figure 30: Steering Wheel Controls
To maximize user experience, these buttons will be mapped to the following functionality.

<table>
<thead>
<tr>
<th>Steering Wheel Control</th>
<th>Replacement Infotainment Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Play/Pause</td>
</tr>
<tr>
<td>Menu Up</td>
<td>Previous Track</td>
</tr>
<tr>
<td>Menu Down</td>
<td>Next Track</td>
</tr>
<tr>
<td>Voice Assistant / Talk</td>
<td>Voice Assistant (Google Assistant)</td>
</tr>
</tbody>
</table>

To intercept the resistive ladder, a Teensy microcontroller is used. The circuit used for implementation is displayed below.

![Steering Wheel Interfacing Circuit](image)

*Figure 31: Steering Wheel Interfacing Circuit*

With this circuit setup, the Teensy can detect steering wheel impedance using a voltage divider and is able to pass this impedance to the stock system (through use of a replicated resistive ladder), or intercept the impedance based on the state of an external switch. This allows the stock functionality of the steering wheel controls to be preserved when it is not desired to use the replacement infotainment system. The Teensy operates as a USB keyboard, inputting keypresses that correspond with steering wheel buttons, when set to intercept button presses. This functionality can be implemented with the following code.
#include <ADC.h>
#define DEBUG 0

ADC *adc;
void setup() {
  adc = new ADC();
  pinMode(A9, INPUT); // PIN 14 from steering wheel
  pinMode(10, INPUT); // external control switch
  adc->adc0->setResolution(12); // 12 bits of resolution for decent detection

  // below writes/modes set replicated resistive ladder outputs to HIGH-Z
  // (faux open) state
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(4, INPUT);
  pinMode(5, INPUT);

  if(debug) Serial.begin(115200);
}

// ADC values that correspond with a button press
/*SRC = 0v
 * MENU_UP = 0.7V
 * MENU_DOWN = 1.3V
 * PHONE = 2.0V
 * OTHERWISE 3.3V
 */
#define SRC 10
#define MENU_UP 2050
#define MENU_DOWN 2980
#define PHONE 3510
#define REST 2000

// amount of tolerance used to blur the lines between discrete buttons
#define TOLERANCE 100

void loop() {
  // external switch input state
  bool hijackControl = digitalRead(10);
  if(debug) Serial.println(hijackControl);
// grab voltage divider state
double a_in = adc->analogRead(A9);

if(MENU_DOWN-TOLERANCE<a_in&&a_in<MENU_DOWN+TOLERANCE){ //Send Button N for next track
    if(DEBUG) Serial.println("MENU_DOWN");
    if(hijackControl) Keyboard.press(KEY_N);
    else{
        // Mimic the resistor ladder if not hijacking the input
        digitalWrite(2, LOW);
        digitalWrite(3, LOW);
        digitalWrite(4, LOW);
        digitalWrite(5, LOW);
        pinMode(2, INPUT);
        pinMode(3, INPUT);
        pinMode(4, OUTPUT);
        pinMode(5, INPUT);
    }
}
else if(PHONE-TOLERANCE<a_in&&a_in<PHONE+TOLERANCE){ //Send Button M for voice command
    if(DEBUG) Serial.println("PHONE");
    if(hijackControl) Keyboard.press(KEY_M);
    else{
        // Mimic the resistor ladder if not hijacking the input
        digitalWrite(2, LOW);
        digitalWrite(3, LOW);
        digitalWrite(4, LOW);
        digitalWrite(5, LOW);
        pinMode(2, INPUT);
        pinMode(3, INPUT);
        pinMode(4, INPUT);
        pinMode(5, OUTPUT);
    }
}
else if(SRC-TOLERANCE<a_in&&a_in<SRC+TOLERANCE){ //Send Button B for play/pause
    if(DEBUG) Serial.println("SRC");
    if(hijackControl) Keyboard.press(KEY_B);
    else{
        // Mimic the resistor ladder if not hijacking the input
        digitalWrite(2, LOW);
        digitalWrite(3, LOW);
        digitalWrite(4, LOW);
digitalWrite(5, LOW);
pinMode(2, OUTPUT);
pinMode(3, INPUT);
pinMode(4, INPUT);
pinMode(5, INPUT);
}
}
else if(MENU_UP-TOLERANCE<a_in&&a_in<MENU_UP+TOLERANCE){ //Send Button V for prev track
  if(DEBUG)Serial.println("MENU_UP");
  if(hijackControl) Keyboard.press(KEY_V);
  else{
    // Mimic the resistor ladder if not hijacking the input
    digitalWrite(2, LOW);
digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
    pinMode(2, INPUT);
    pinMode(3, OUTPUT);
    pinMode(4, INPUT);
    pinMode(5, INPUT);
  }
}
else if(REST-TOLERANCE<a_in){ //Release all Buttons (no buttons being pressed on wheel)
  if(DEBUG)Serial.println("REST");
  Keyboard.release(KEY_B);
  Keyboard.release(KEY_V);
  Keyboard.release(KEY_N);
  Keyboard.release(KEY_M);
  // Mimic the resistor ladder
  digitalWrite(2, LOW);
digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
    pinMode(2, INPUT);
    pinMode(3, INPUT);
    pinMode(4, INPUT);
    pinMode(5, INPUT);
}
if(!hijackControl){
  // if we're not stealing input, don't send any button presses (this prevents an edge case of external button
    // switched as steering wheel button pressed )
  Keyboard.release(KEY_B);
Keyboard.release(KEY_V);
Keyboard.release(KEY_N);
Keyboard.release(KEY_M);

} if (DEBUG) {Serial.print("CURRENT VALUE: ");Serial.println(a_in);} delay(50);

User Experience
As this project is user facing, it’s essential that the user experience be a positive one. The replacement infotainment system has a number of small quality of life improvements over the original infotainment system, including but not limited to:

Automatic Day/Night Theming:

Using the automatic headlight status as a source of data on the light level outside of the vehicle, the replacement infotainment automatically switches between a dark and light theme, sparing the eyesight of the user.

Automatic Device Connection:

Upon start of the car, the replacement infotainment system attempts connection with the last Android device used for Android Auto. If the same driver repeatedly uses the vehicle, this results in them not even needing to remove their phone from their pocket.

Delayed Shutdown:

The Raspberry Pi powering the system does not shut down until four minutes after the car has been shut off, this means that for quick stops (such as forgetting something at home), the infotainment does not need to power on and reconnect to the driver’s device, saving time.
Conclusion

This project achieved what it set out to do: provide an upgraded infotainment system for a 2011 Infiniti G37 that is seamlessly integrated to an end user. While all the goals of the project were met, there are many areas in which this project could be improved and expanded in the future. The largest category of these future improvements is further integration with the stock radio and car computer systems. There are many communication lines within the Infiniti G37 that have yet to be tapped into and reverse engineered. These include (but are not limited to), a secondary CAN Bus connecting the AV system, radar sensors, and end user controls, as well as serial lines exposed on the motorized seat and mirror system, or the small LCD display in the middle of the instrumentation cluster. There are a multitude of ways that these untapped systems could be integrated into the upgraded infotainment system, such as displaying turn by turn navigation steps in the instrumentation cluster display.

Many challenges were encountered during the development of this project, primarily software in nature. The single board computer chosen to host the replacement infotainment software, a Raspberry Pi 4, proved to be a double-edged sword during development. The Raspberry Pi was the only of the single board computers evaluated that matched the specifications required for the infotainment upgrade (especially those around analog video and hardware accelerated video decoding). However, at the time of development, the Raspberry Pi software ecosystem was going through many growing pains. With the release of Raspberry Pi OS Bullseye on November 8th, 2021, many software architecture components were changed. For instance, Bullseye came with the release of a new kernel graphics driver for the onboard GPU of the Raspberry Pi. This driver changed how analog video (as well as hardware decoding) were performed on the Raspberry Pi. As a result, legacy documentation no longer applied to the most recent software versions. Many bugs were also encountered and reported to the Raspberry Pi team through this project, meaning that this project served to further develop and stabilize the Raspberry Pi ecosystem. Similar issues were encountered with the Raspberry Pi Bluetooth stability (necessary to enable wireless Android Auto), due to similar driver migrations to the Linux kernel. All problems encountered were resolved or worked around in order to provide a stable infotainment end-user experience.
References


Appendices

Senior Project Analysis

Summary of Functional Requirements
This project retrofits a 2011 Infiniti G37 with an updated infotainment system. This updated infotainment system includes new capabilities, such as Android Auto, a Bluetooth receiver, and a data page presenting information to the user, such as tire pressure.

Primary Constraints
The largest constraint of this project was the integration with the Infiniti G37. The software integration was relatively simple, as there were less constraints imposed by the vehicle itself. The electrical integration, however, posed a significant challenge. Trying to integrate electrically with the G37 imposed constraints on power usage (both duty cycle and overall consumption), as well as constraints on video protocol, audio protocol, and CAN message formatting. Add into these constraints an overall lack of documentation on the electronics within the G37, and the difficulty of this project emerges; it’s difficult to reverse engineer the specifics of a black box.

Economic
This project has an estimated direct cost of $136. Funding is provided by the developer of the project. In addition to the direct cost of components and services, there are indirect costs of labor (research and development, as well as installation), and test/development equipment. This development of this project required a power supply, waveform generator, oscilloscope, and logic analyzer. There is no expected return from this project. Most of these costs incur at the research and development (initial) phase of the project. Once installed, there is no maintenance cost to the user, nor continuing cost of development. Specific project breakdowns are specified below.

Human Capital
Development of this project required the knowledge and time of an engineer knowledgeable in various communication protocols and software architecture.

Financial Capital
This project required the use of many high-cost items, such as

i. 2011 Infiniti G37
   - Target vehicle for integration of project, costs approximately $12k, used.

ii. Bench Power Supply
    - Allows for development of project outside of vehicle, approximate $1k cost, used.

iii. Waveform Generator
     - Allows for development of project outside of vehicle, approximate $500 cost, used.

iv. Logic Analyzer
    - Allows for development of project outside of vehicle, approximate $250 cost, new.

v. Oscilloscope
    - Allows for development of project outside of vehicle, approximate $300 cost, new.
Manufactured or Real Capital
There are labor costs associated with the design of the PCB. Within the scope of this project, though, there is no manufactured capital, as the PCB manufacturing is outsourced.

Natural Capital
The retrofitted infotainment consists of PCBs and electronics that contain materials such as fiberglass and silicon. There are also other resources, such as etching acids, used during the manufacture of these components.

If manufactured on a commercial basis
While there are no plans to sell the components of this project individually, at a manufacturing cost of about $136 per unit, the unit would sell to customers at a price of around $300, in order to offset R&D cost. This profit margin could potentially be increased by around 15% if there was enough demand to produce the manufactured units at a large enough scale, which would bring down the cost per unit. This project could sell upward of 500 units a year, which would equate to $150,000 in sales a year.

Given how closely related the infotainment systems are across the Infiniti brand, as well as across the Nissan brand, this product could be easily adapted for multiple vehicle models and years. Releasing units compatible with multiple vehicles could significantly increase the units sold per year, from 500 to upward of 5000. This would equate to $1.5 million in sales yearly. However, due to the software technologies used within this project, the project may not be eligible for commercial sale. Further exploration and legal consultation of the licenses of various software libraries would need to be conducted prior to any plans for commercialization.

Environmental
As with many electronic projects, this project influences the environment due to the materials used within the electrical components. The resources and chemicals used to produce the components, as well as the byproducts of the manufacturing process have a negative effect on the environment. However, much of the environmental factors this project touches upon are indirect. As this project is installed within an internal combustion engine vehicle, there is a negative effect of encouraging more driving within the vehicle.

With the negative effects come the positive as well. This project indirectly affects several factors, from reducing roadkill to minimizing accidental car idle time, by encouraging safer and distraction-less driving from the operator of the vehicle it is installed within.

Manufacturability
There was a significant challenge in the manufacturing of this project, due to the impact of worldwide supply chain challenges and electrical chip component shortages brought on in part by the COVID-19 pandemic. There was an additional challenge with regards to sourcing high temperature tolerant flavors of components. As the interior of a vehicle can reach temperatures up to 172 °F, the components within this infotainment system must be tolerant of extreme heat. Each component used within the design had to be sourced as to meet this environmental constraint.

Sustainability
As the retrofit of infotainment to the vehicle adds modern features to the vehicle which did not exist in 2011 model vehicles (such as Android Auto, or Bluetooth audio streaming), this project
promotes sustainability. Rather than purchasing a whole new vehicle in order to gain newer features, this retrofit allows them to be installed in an older vehicle, encouraging a vehicle owner to keep an older car on the road, rather than sending it to a junkyard and purchasing a new vehicle.

The retrofitted unit is designed to contain minimal components, meaning that there is a small impact on the user’s finances and environment to repair or replace the system when/if it fails. Compared to the cost (both financial and environmental) of manufacturing and purchasing a new car, this cost is the equivalent of a drop of water when compared to the ocean.

Ethical
This project follows the IEEE Code of Ethics. Implementation of this project actively promotes IEEE Code of Ethics Section I.1, by engaging in sustainable practices as previously noted in section 7, Sustainability. The development of this project also followed all IEEE Code of Ethics pertaining to individuals, such as Section I.4, and Section II [1]. This project was developed with an Ethical Egoism framework, as the project primarily impacts one or two people at most (a driver and their passengers) and aims to promote the self-interests of those in the vehicle at any one time, by giving them a better driving experience. However, this project is also ethical when looked at through a Utilitarianist lens. By promoting safe driving practices through the incorporation of Android Auto (which has a primary focus on hands free usage and distraction-less driving), this project can improve the safety of the community the retrofitted vehicle is used within (thus maximizing happiness for the most amount of people).

Health and Safety
As this project involves modification of a vehicle, great care was taken to not impact the safety of the driver and passengers contained within. By utilizing the original display and user interface controls of the vehicle, this project avoids introducing any distraction or danger to the driver. Both the original controls and interface of the vehicle, as well as the software of Android Auto, have been accessed and approved by the Department of Transportation.

This project also includes tapping into the data lines of the vehicle, which arises potential concerns around the disruption of vehicle functions, which could endanger the health and safety of the occupants. However, this data is accessed through the OBD-II protocol and system, which is a standard in most vehicles post 1996 used for car diagnostics. These systems and protocols are isolated from the operation of the vehicle itself, and data access is done through standardized API calls, which are read only, and thus cannot impact the vehicle.

Social and Political
This project has direct and indirect stakeholders. Direct stakeholders include those directly interacting with the retrofitted vehicle and its infotainment, such as vehicle drivers or passengers. Indirect stakeholders include those who may encounter the vehicle otherwise, such as a driver in a separate vehicle on the same street as the modified vehicle, or a bicyclist or pedestrian passing alongside the modified vehicle.

This project benefits both the direct and indirect stakeholders. The direct stakeholders are benefited by the increase in functionality of the vehicle’s infotainment system – allowing them to ride in the vehicle with added navigation and audio capabilities. The indirect stakeholders are benefited from the secondary effects of the benefits to the direct stakeholders; By providing the driver of the vehicle with built in navigation and media through Android Auto, the driver can
focus on the road, rather than trying to understand navigation or audio on a mobile phone or printed out directions. Removing distractions from the driver makes those around the vehicle safer in return.

Development
This project forced me to become intimately familiar with analog video encoding, and the RGBS signaling protocol. Reverse engineering the CAN Bus of the vehicle also cause me to develop not only general reverse engineering skills, but also knowledge of the CAN specification and methodology, as well as data signaling methods such as CRC and CAN transceiver conflicts. The reverse engineer process also caused me to become familiar with the use of Saleae logic analyzers.

Parts List and Costs
Software
All software designed as a part of this project has been published on GitHub [22].

Hardware
All hardware designed as a part of this project has been published on GitHub [22].