

ECE 499 Research Project Report

Shafkat ul Haque
User ID: kshaque
Student ID: 20382969

Instructor: Professor Sebastian Fischmeister

Spring 2016

Summary

The Advanced Driving Assistance System (ADAS) hardware and software setup in the Real Time Embedded Systems lab under Sebastian Fischmeister's supervision has progressed very rapidly and successfully in the last few years. Right now, in brief, the system works by having a host computer running all the software talk to a remote controller for a small RC car that is placed on a treadmill. The treadmill is also connected to the host computer. There is a camera that is placed directly above the treadmill that is used as part of a closed loop feedback system that guides the car on the treadmill.

In this current setup, the areas that needed important development work were

- The bridge between the host computer and the remote controller that talks to the RC car
- A diagnostics framework that would allow researchers to determine how the system is functioning in cases either a component of the system has been replaced, or to check if the remote control car is working as per specifications

The host computer is running ROS which is talking to the remote control hardware over an Arduino Mega as an I2C bridge. This particular bridge setup was prototyped on a breadboard with loose fly wires and jumper wires. This prototype worked as needed, however it was very fragile and prone to damage. This was a weak hardware link in the system.

At this point in the development of the system, it is very important for the system to have a good diagnostic framework to be able to routinely check and evaluate the status of the system. More and more researchers and students are contributing to this project and the ROS software for the system is being designed and implemented in a way that focuses on modularity so that many can contribute to the code base and add to the research without breaking any of the existing features. The diagnostic framework is thus important to ensure the continuously updating code base is not disrupting the system. Also if in the future any other researcher wants to replicate this system, then a diagnostic framework will help check and ensure the system is rebuilt properly to specifications.

The solutions implemented in this report are of a custom designed hardware which will act as a bridge from the host computer to the remote controller hardware. It has been designed in a generic way to allow for a wider range of compatibility for any future remote control hardware changes.

In this report I will talk about the solutions to the above problems and go over the design and implementation of the solutions with a conclusion with the results of the solutions and possible improvements and recommendations.

Table of Contents

<i>Summary</i>	1
<i>List of Figures</i>	3
<i>1.0 Introduction</i>	4
<i>2.0 Specifications</i>	6
<i>3.0 Analysis of Possible Solutions</i>	7
<i>4.0 Implemented Solutions</i>	11
<i>5.0 Conclusions</i>	16
<i>6.0 Recommendations</i>	18

List of Figures

Figure 1: Arduino Mega Cape Design	7
Figure 2: USB to I2C Bridge Design.....	8
Figure 3: Diagnostic HW Design 1	9
Figure 4: Diagnostic HW Design 2	9
Figure 5: Schematic of Arduino Mega Cape	11
Figure 6: Arduino Mega Cape PCB Layout	12
Figure 7: Assembled Arduino Mega Cape (Top View)	12
Figure 8: Schematic of USB to I2C Bridge	13
Figure 9: USB to I2C Bridge PCB Layout with Highlighted Header Pins	13
Figure 10: Assembled USB to I2C Bridge (Top View).....	14
Figure 11: Assembled USB to I2C Bridge (Back View).....	14
Figure 12: Prototype Bridge Hardware	17
Figure 13: Arduino Mega Cape Design	17
Figure 14: USB to I2C Bridge Design.....	17

1.0 Introduction

The ADAS project's primary goal is to help create a platform for researchers all around the world to develop software and algorithms for autonomous vehicles. This area of autonomous driving has become a very important area of research in the last few years and now most major car manufacturers are investing heavily in autonomous driving technology. One of the biggest proponents of this technology in the recent years has been Tesla Motors and Google where they have been spearheading the development and implementation of autonomous driving.

All of this excitement and interest in this technology, which in theory can be orders of magnitude safer than human drivers is causing researchers all around the world to contribute to this area of technology development. As this technology is safety critical and mistakes in implementation can cause people to lose lives, research into safety is crucially important. It is not safe or financially sound for researchers to develop and test their algorithms and software on a real vehicle and drive around the streets. This is where this ADAS platform comes in. This platform will allow researchers to quite easily replicate the system at a relatively low cost and start contributing to the code base and even perhaps hardware.

One of the newest components in the ADAS platform is the use of a smaller RC car that does not carry any extra hardware. Previously the setup used a bigger RC car that had extra hardware added to it and ran QNX on board. The drawbacks to that setup was all that extra hardware was degrading the RC car's performance. The car required a lot of extra power to run the extra hardware and the weight of the extra hardware was draining the battery faster. This is why the smaller RC car is being used and instead of adding all the hardware on the car itself, the hardware is being off-loaded to the host computer running ROS. Instead of manually sending commands to the car directly, now the commands are being sent to the remote controller for the car and the car can function normally.

This new setup where the car commands are being fed directly into the remote controller needed a USB to I2C bridge allowing ROS to send appropriate analog signals to the remote controller. This is the part of the platform that is right now weak and has room for improvement. The bridge hardware was prototyped using a breadboard and connected to the remote controller using fly wires and jumper wires. Two DACs were used to convert the digital signals coming from the host computer to be converted to analog signals and then some level shifting circuitry was used to bring the analog signal down to appropriate levels for the remote controller. An Arduino Mega was connected to the host computer and the bridging hardware was connected to the Arduino in turn.

Another recent change made to the ADAS platform is the use of ROS instead of QNX. Due to the merit of ROS having a very modular design, it has made the process of adding different software components to the system much easier. However, as this platform becomes more modular and more people start contributing to this platform, it becomes more and more important to have diagnostic tests that can be performed to ensure the system is performing according to specifications and helps maintain a benchmark for the

system performance. Another important aspect of having diagnostic tests is that can check if the system is performing properly is that replacing hardware components become easier. For example, if the small RC car used on the treadmill was replaced by a new one, by running system diagnostic tests, it can be very quickly determined if the new car needs any calibration to be compatible with the platform.

As highlighted before in this section, one of the areas of improvement in the platform is the host computer to remote controller interface. During the course of ECE 499, I dedicated my time in researching into ways of implementing a bridge interface for this problem. It is an important problem to solve from the point of view of scalability as more and more cars are added or tested on the platform. Having a customized generic interface that can communicate with a given set of remote controllers can be very beneficial.

My second focus of research was coming up with a diagnostic framework which can easily determine the performance of the system. As mentioned earlier, as more new models of cars are introduced to the platform, with each car having its own set of physical properties (dimensions, axle length, etc.), it is important to be able to quantitatively check how well the system is performing.

The following sections will elaborate more on the problem descriptions and mainly on how I derived possible solutions and implementation details of chosen solutions.

With advisement from Professor Fischmeister I decided on the what kind of technologies to learn or improve on during the course of this course. I wanted to gain experience in designing hardware and the Professor wanted some diagnostic software to be written for this ADAS platform. Therefore, in the course of this course, my learning objectives were set to be the following:

1. Analyze system requirements and develop hardware solution
2. Design hardware schematics
3. Design custom PCB layout
4. Fabricate PCB and populate the PCB with components
5. Gain soldering skills (through-hole and surface mount)
6. Debug and integrate new hardware into existing system
7. Write software that interfaces with new hardware
8. Design and write a diagnostic framework

2.0 Specifications

The following specifications for the problem areas were determined after consultation from David (Donghyun) Shin and Professor Sebastian Fischmeister.

2.1 Communication between Host Machine and Remote Controller

- The hardware bridge/interface should be able to integrate into the ROS code that already exists.
- The hardware interface should not require any major code changes or changes to the firmware stacks already used
- Hardware interface should be forward compatible with other remote controllers of similar category
- Hardware interface should allow a 6V power supply to be routed to the remote controller so that the remote controller does not have to depend on batteries for power.
- The hardware should be safe, reliable and portable.

2.2 Diagnostic tests to evaluate the response of the system

- Diagnostic framework should be able to produce data on the performance of the system
- Code should be written in a way to allow other diagnostic components to be added in a modular fashion

3.0 Analysis of Possible Solutions

3.1 Arduino Mega Bridge

This design is intended to be a direct replacement of the current prototype board. Using this board will require no changes to the code base and the system. If the board is functioning properly, then there should be no changes in behavior of the system and the no changes in the way the software sends data to the remote controller hardware.

As the Arduino Mega is still used as part of the bridge communication hardware, a design decision was made to make the level shifting DAC circuit in the form of a cape that can be plugged into the Arduino Mega. The level shifting DAC circuit relies on the SDA, SCL, 5V and GND signals from the Arduino Mega. This is why it makes sense to simply plug in the extra circuitry on top of the Arduino Mega.

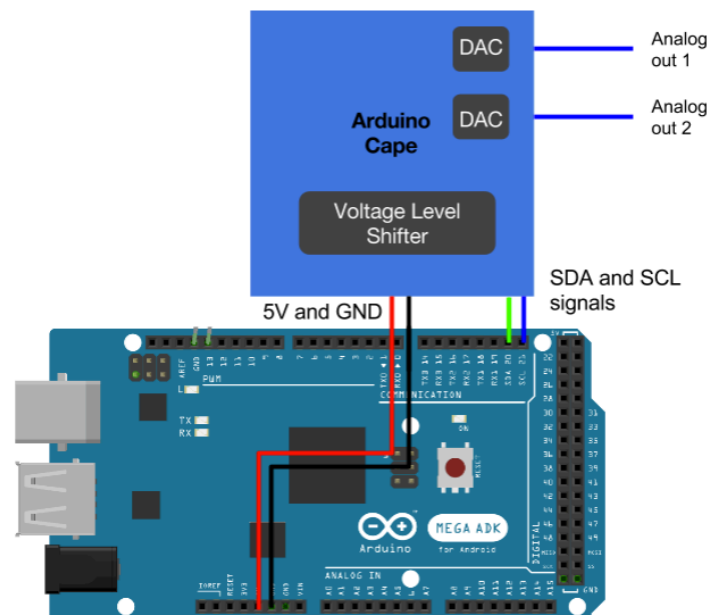


Figure 1: Arduino Mega Cape Design

3.2 MCP2221 USB to I2C Bridge

In this solution, the Arduino Mega is removed from the setup. This has many benefits, the primary being we are using less hardware now. The Arduino Mega was too powerful for a simple task as relaying data from USB to I2C. To replace this board, the MCP2221 chip has been used. This chip provides the same functionality of being a USB to I2C interface but being much smaller. The MCP 2221 chip is selected because it has good linux library support available from the manufacturer and should work well with the DACs used which are also from the same manufacturer. Other USB to I2C chips that can be used are for example FTDI FT201X which has similar features. I am selecting the MCP 2221 chip as I have more knowledge about this particular chip.

This version of the bridge should also be much smaller in PCB footprint as less components are being used compared to the previous design. Also as we are not relying on

the Arduino Mega anymore, there is no need to make this PCB big enough to be a cape. We can easily shrink it as much as possible making it very portable and cheap.

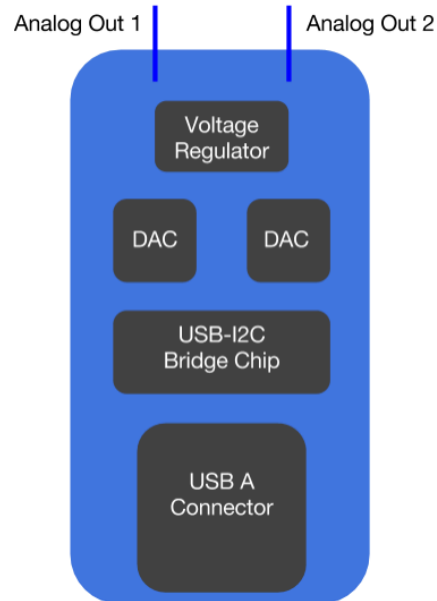


Figure 2: USB to I2C Bridge Design

Also in this design, the level shifting circuit used on the prototype can be removed and replaced with a linear voltage regulator. This removes the number of passive components required on the PCB and makes the design simpler.

3.3 Diagnostic Framework

To tackle the problem of doing diagnostics on the system, the first step in that direction should be creating a framework which allows for various types of diagnostic tests to be run on the platform. The test themselves should be very easy to add to the system, this would lower the barrier to adding rigorous tests which would only enhance the overall quality of the platform. The ROS architecture is very helpful for this solution where the diagnostic framework can simply be a node that can be used whenever system diagnostics are required.

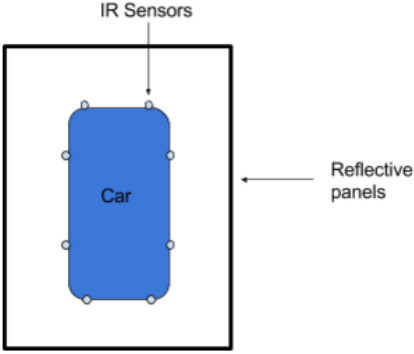
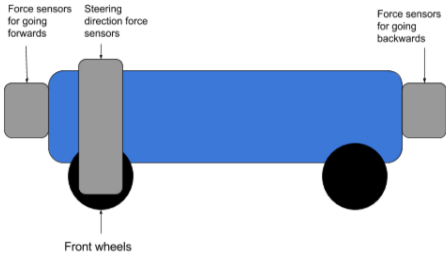
3.4 Diagnostic Tests

The diagnostic tests themselves should be very focused. This will ensure that the data produced is granular and easy to analyze. For example, one simple diagnostic test can be to set a destination for the RC car, perhaps straight ahead of its point of origin. So essentially this test would require the car to go forward a certain pre-defined distance. By doing this simple action, it would be possible to determine the system response delay, error in the position of the car, possibly pointing to the error in the camera, etc. All this data when quantized and visualized will be a very powerful aid in helping fine tune the system.

3.5 Diagnostic Hardware

For further analysis of the RC car itself, it is possible to build a test rig in which the car is placed to simulate the treadmill. Force sensors around the harness of the rig where the car will be placed would be able to detect if the car is pushing forwards or reverse. Also force sensors on the sides of the wheels would be able to detect the angular force of the wheels trying to turn left or right. Using this method, the RC cars can be thoroughly tested and benchmarked before testing them on the treadmill.

Some possible designs of this test rig can be:

Design	Description
<p>Use of LED/IR sensors to track motion of card on the treadmill.</p>  <p>Figure 3: Diagnostic HW Design 1</p>	<ul style="list-style-type: none"> The IR Sensor “rig” will be attached to the car and its movements can be detected from the IR signals reflected from the highly reflective panels around the car. The IR sensors can be connected to a host computers via wired connection. Or the sensors can be connected to a small on board computer such as a mini Raspberry Pi and the data can be sent over wifi to a host machine. Data collected can be processed on the host machine
<p>Use of force sensors attached to front and back wheels of the car to detect motion of steering and also detect RPM of the wheels.</p>  <p>Figure 4: Diagnostic HW Design 2</p>	<ul style="list-style-type: none"> Pressure/Force sensors will be able to detect the force of the front wheels turning and can be translated into the angle of the steering motion. The force/pressure sensors in the back and the front will be able to detect the car’s forward and backward motions and that can be translated to speed of the car. The hardware for the sensors can be wired to a host machine as the car doesn’t actually move (the car will be clamped down).
<p>Probe directly onto the car hardware to measure the signals</p>	<ul style="list-style-type: none"> This method will be the easiest to setup, however will have a margin of error as we

going into the servo motors for steering and the main DC motor for going backwards and forwards.

are not taking into account the systematic error that comes from the servo motor and the DC motor. We will only be measuring the signal going into the motor mechanism, not the actual motion of the motors.

- This method can of course be added to methods 1 and 2 for more collection of data. In fact, it is highly recommended that the signals going into the motors be collected along with the data on the motion of the car itself. This will aid in better comparison and analysis of any errors.
- This method is going to be rather straightforward, with the data collected being fed onto a host machine for visualizing results.

4.0 Implemented Solutions

After due consideration, I selected the following solutions to implement. The design decisions, implementation details and other finer details about the solutions are highlighted in the following sections.

4.1 Arduino Mega Bridge Cape

Descriptions of this design has been described in previous sections so let's jump directly into the schematics and PCB layouts for further implementation details. The schematic of the design is as follows:

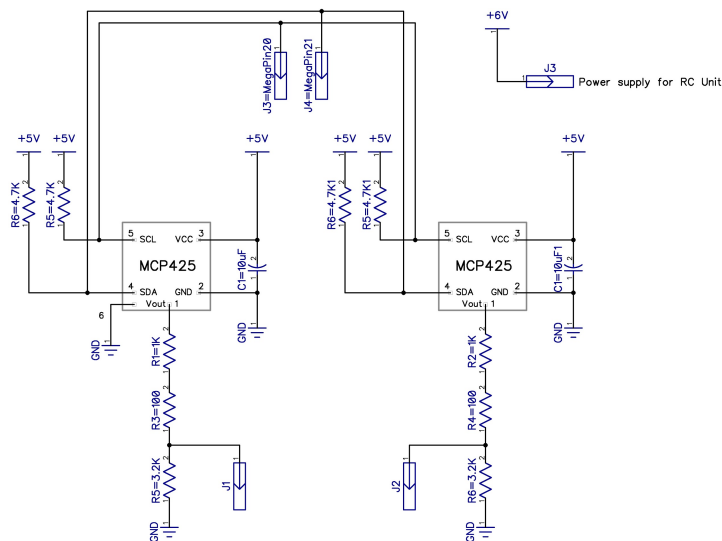


Figure 5: Schematic of Arduino Mega Cape

This design is for the most part based on the prototype design. After replicating the existing design, I made additions of an external power supply pin to allow for powering the remote controller hardware directly from a wall outlet rather than using batteries. Not much was changed in the original design of this board in terms of the schematic. As can be seen in the PCB layout diagram below, the form factor of the and the layout of the design was changed.

The PCB was designed to be a cape that could be easily placed on top of the Arduino mega board. This would allow the bridge PCB to be tightly coupled with the Arduino board and removing any need of extra wires, which was a weakness of the prototype board.

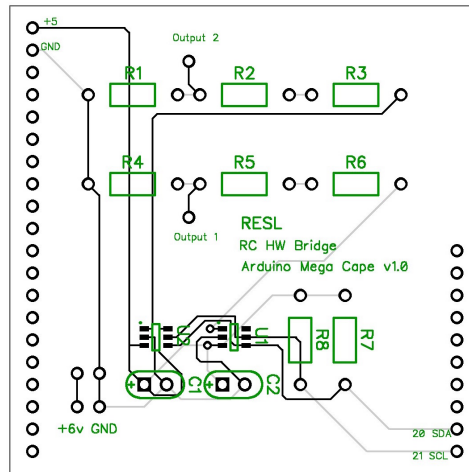


Figure 6: Arduino Mega Cape PCB Layout

The image below shows the final PCB fully populated with all components placed on top of the Arduino Mega as a cape.

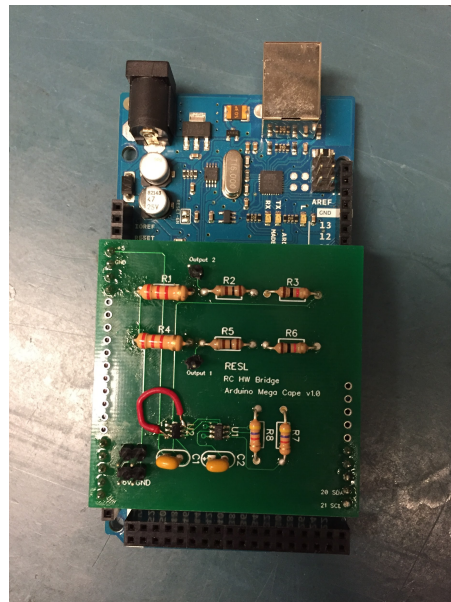
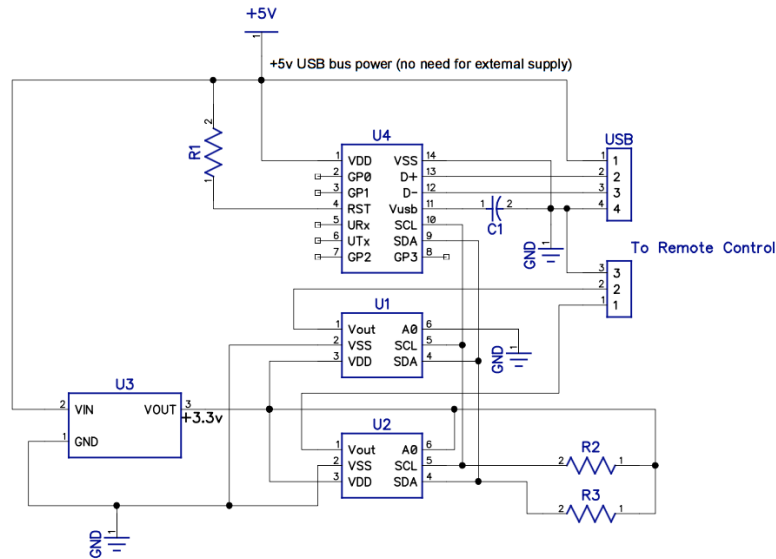


Figure 7: Assembled Arduino Mega Cape (Top View)

4.2 MCP2221 USB Bridge

This solution with MCP2221 chip is a more advanced solution to the host-to-car bridge problem. This is also a more sophisticated solution as it can be used for any many types of use cases where data from the host machine needs to be sent over to the RC car. As this solution eliminates the need for the Arduino Mega board, this solution is much cheaper and portable. The following diagram shows the schematic of the board.



A few finer details about this design. Just like the previous design this board provisions extra header pins to supply remote controller hardware with power from the wall outlet as can be seen in the diagram below, pins in section 1. Also, just like the previous design, the board has the necessary 2 analog output pins to be connected to the remote controller hardware, labeled A1 and A2 in section 2 of Figure 8 below.

A new addition to this design that is different from the previous design is the addition of debugging pins so that data going to the remote controller can be externally monitored. Also there are 2 GPIO pins available on this particular MCP2221 chip which enables more advanced interactions with this hardware. These pins are shown in section 3 of the diagram below.

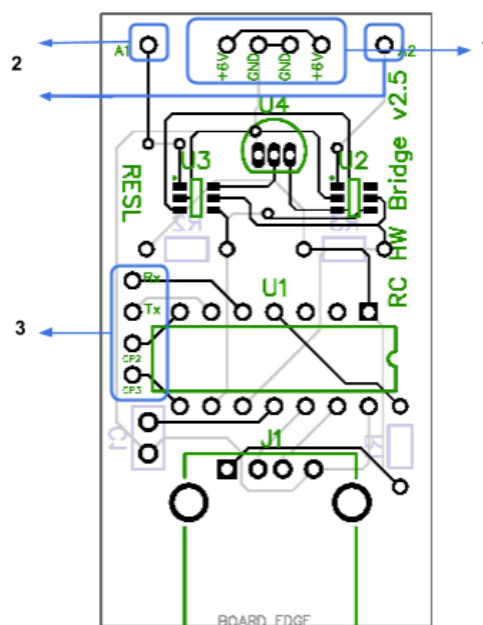


Figure 9: USB to I2C Bridge PCB Layout with Highlighted Header Pins

Having debug pins that can be used to monitor data traffic on the bus is a very big security vulnerability, but as this system is strictly going to be use for research and development purposes, having these pins should not pose any risks. In fact, they can be used to monitor data traffic for diagnostics purposes.

The GPIO pins available here can be programmed to do performance measurements of the system. In order to do worst-case-execution-time (WCET) analysis, it is very important to not only perform static code analysis, dynamic code analysis from perhaps timestamps, but also to have pin toggles associated with various events in the software. Using these GPIO pin toggles along with the software based measurement techniques, this hybrid approach can bring best of both types of analysis and produce more accurate results than simply using static code analysis and timestamps from program runtime. The hope is that using this bridge hardware, not only can the essential command data be sent over to the RC car, but also can be used to do valuable performance analysis of the code and the system overall.

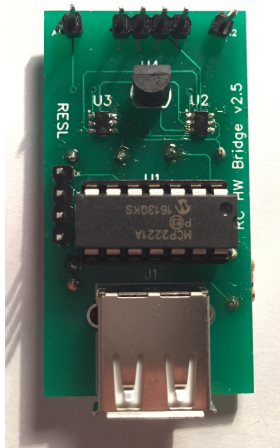


Figure 10: Assembled USB to I2C Bridge (Top View)

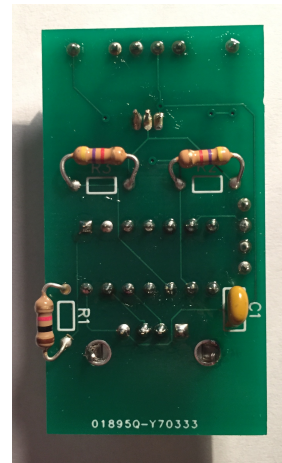


Figure 11: Assembled USB to I2C Bridge (Back View)

This design however cannot be integrated into the system right away. The benefit of the previous design is that it relies on the Arduino Mega board to connect to the host machine to retrieve data. The software in ROS has been to specifically communicate with this Arduino board. The software uses library stacks that are for specific to Arduino Mega. As this design of the bridge is not using the Arduino Mega, the current software will not work with this board. The software needs to be modified and the underlying libraries need to be changed to support the MCP2221 chip. This software change is going to take some time and will be not be feasible in the span of this course. However, MCP2221 has all necessary libraries available and a lot of code is already provided from the manufacturer of the chip which will speed up the software refactoring process.

In order to test and demonstrate that this hardware is functional and works, I have developed a small program that can send manual commands to the remote controller hardware. This program can be expanded on for diagnostics use or debugging purposes, but for now it is meant for only testing and demonstrating that the hardware is functional.

The small program that was written to verify the board is functional is very simple. It takes in user data from the terminal and sends it to the respective DACs. The MCP4725 DACs allow addresses ranging from 0x62 to 0x63. This is convenient because we only need to address only 2 DACs in the board. So using the respective address of the DAC, the program writes to the I2C bus.

4.3 Diagnostic Framework

The diagnostic framework that was implemented in the form of a node in ROS was very simple in design. The diagnostic node contains a simple test that can be triggered from command line with respective settings. In the future when more tests have been developed, they will all reside in this node. The node advertises the coordinates to which it wants the car to go so that it can measure the car's performance and the node subscribes to the lane occupancy message that contains information on the position of the car. The data from the lane occupancy message is then pushed into a data file for post processing and analysis which can be found in /home/user/data.txt. The format of the data inside the data file is <ROS Time stamp>:<Car's mid point x value>. The time stamp will make it easier to analyze the location of the car. This data file is specific to the simple test already included in the node. For every test, there should be a specific data file generated with appropriate data formatting.

5.0 Conclusions

The overall process for the development of hardware for the ADAS platform went really well. It gave me further insight into the potentials of the platform and helped strengthen my engineering skills. The hardware that was produced for the host-to-RC bridge proved to be very helpful in furthering the development of the platform. It made the system more stable and will allow more RC cars to be added to the platform without any further difficulty. Also the design of the bridge using the MCP2221 paves out the platform for making the system more robust and simplifying the hardware.

The development of the basic diagnostic framework was started which will help facilitate future development on this platform. Now a ROS node has been created in the code base that can easily include more and more types of tests improving the test coverage of the system.

I was able to accomplish most of the learning goals set in the beginning of the course. Starting from designing the schematics, PCB layouts to printing the PCB layout from vendors in China proved very insightful. I learned a great deal about the hardware development process and also was able to strengthen some other related skills such as soldering components on the PCB.

Developing software for 2nd version of the board with the MCP2221 chip proved challenging. In the time left after building and assembling the hardware I was able to write some basic software that allows commands to be sent manually to the remote controller hardware. This was done to demonstrate that the hardware works. This program can also be used to verify other copies of the hardware which may be assembled at a future date. I was also able to isolate the libraries and the software stacks required to implement code in ROS to communicate in a Linux environment.

Minor mistakes in the hardware design that should be pointed out:

- The cape bridge hardware that was developed had minor errors in the design. The voltage divider resistor values were incorrect where they should have been 2.0K Ohm and 1.1K Ohm, instead incorrectly 3.2K Ohm and 1.1K Ohm were used. But this is a minor problem that can be easily fixed by replacing the 3.2K Ohm resistor with a 2.0K Ohm resistor.
- Another minor issue in the cape bridge hardware was that there was a missing trace between pin 6 and pin 3 of one of the DACs. This was fixed promptly by adding a small wire between the two pins and connecting them.

I am satisfied with the final result of the hardware projects. The cape hardware design has been tested and is performing according to specifications and the USB-I2C bridge design also is behaving within specifications. The diagnostic node has been made and should work properly on the system.

The following figures quickly summarize the impact of the newly designed hardware:

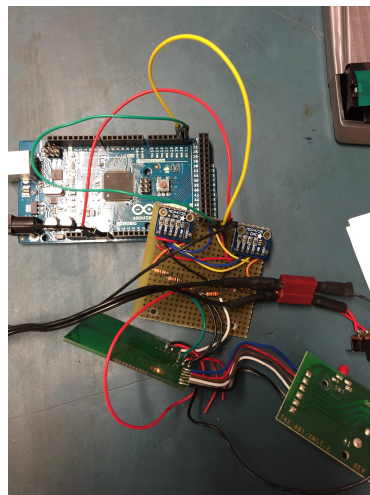


Figure 12: Prototype Bridge Hardware

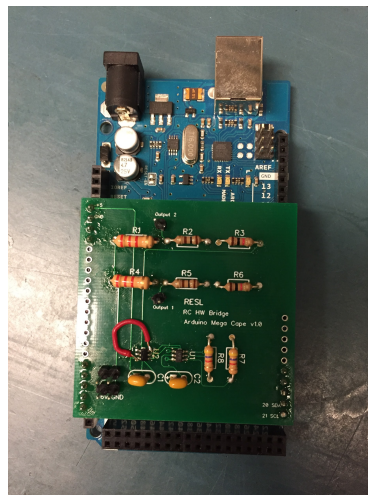


Figure 13: Arduino Mega Cape Design



Figure 14: USB to I2C Bridge Design

Initial solution:

- Extremely messy
- Tons of loose wires
- Difficult to produce
- Prone to error and damage

First design:

- Cleaner and simpler
- Compact and drastically reduced wiring
- Reliable and safer
- Easy to reproduce
- Still dependent on Arduino Mega
- Easy to produce

Second design:

- Arduino Mega dependency removed
- Smaller and compact
- GPIO pins for measurements
- Passive components reduced
- Compatible with many RC hardware
- Easy to produce

6.0 Recommendations

A possible iteration on the hardware that was developed using the MCP2221 chip can be that the design is more generic and able to handle communication link between multiple remote controller hardware. The current design is built so that 1 bridge is used per host-remote controller connection, i.e. 1:1 connection. However, if the hardware design was changed so that more DACs are used on the bridge hardware to accommodate multiple connections to many remote controller hardware, this way, the MCP2221 can simply use a multiplexer to switch between the pairs of DACs. As the DACs used in the designs only allow addresses of 0x62 and 0x63, the only way to have more than 2 DACs on the hardware would be to turn on/off the DACs not being used at any given moment. This can be quite easily achieved by using the 4 GPIO pins available to the MCP2221. These 4 pins can be used as 4 bits to address up to 16 pairs of DACs. This means it would be possible to control up to 16 RC cars at a time using only one bridge. For any address of 4 bits, the corresponding DAC pair can be turned on and all the other DAC pairs can be turned off by toggling the power pins to ground.

This would lead to a 1:n setup where n is the number of simultaneous connection the system requires. The MCP2221 chip does not have any inherent limitations that would prevent it from addressing different pairs of DAC at any given time leading a very scalable and robust hardware solution.

Another feature that might be of benefit is adding more support on the board for debugging and diagnostics. An example of this can be the adding more pins that can be probed to monitor the signals being sent via the DACs. This feature would further aid in performance measurements in code execution times.

The only drawback to this above mentioned design is that we will be losing granular control over each remote controller hardware. This could affect things like parallel performance tests, software execution analysis, etc. But I recommend this is an area should be explored further.