Interfacing a Wired PS2 Controller with PIC Microcontroller

Abhi Agrahari

January 26, 2018

Abstract

The main goal of this project was to create an RC car that would function normally, even if flipped upside down. However, due to time constraints, and other projects that were also being worked on (Gramophone for art students), this project was temporarily put on hold. The scope of this project was redefined, to interfacing a PS2 controller, with a PIC microchip.

Table of Contents

Introduction

Any intermediate programmer that is experienced with hardware, would be able to understand this report. The main technology involved is a wired third-party PS2 controller.

For my summative, I wanted to create something that combined mechanical/hardware aspects, and programming/software aspects. Last year I had created a robotic hand with Eric, but this year I wanted to make a vehicle, and wanted to make my own circuit board for it. I have always been interested in remote control cars, so I thought that this would be a good opportunity to create one.

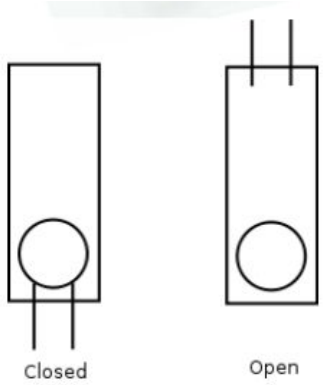
This report intends to cover the design process behind both the mechanical aspects of the interface, as well as the code created for this project. All additional information relating to this project (data sheets, sketches, schematics, technical drawings) are in the docs folder on my github here.

Hardware Design

For this project, I designed all hardware related components except the PS2 controller, of course. My main hardware focus this semester was creating my circuit board. I had various issues, but was able to solve them as outlined below.

Circuit Board Design

My circuit board design can be found here. I used the PIC16F1459 chip. My circuit board is powered through the BEC in an ESC. The chip is programmed through USB. There are 2 controllable headlights, a beeper, and a servo output. There is one push button, that I am using as a reset button. There two tilt switches for inputs. They work as the image shown below. When the tilt switch is rightside up, the two pins will connect through the metal ball inside. If it is upside down, the ball doesn't connect the two pins. By using 2 of these, I can redundantly check the orientation of the RC car.



Another important part of my circuit board is the PS2 Interface Connections. 7 of the 9 PS2 connection wires are used. These connections can be found at the bottom of my circuit board, and are in proper order.

- Data (Controller → Chip)
- Command (Chip → controller)
- Vibration Motor Power (Chip → controller)
- Ground
- Power (3.5V)
- Attention (Chip → controller)

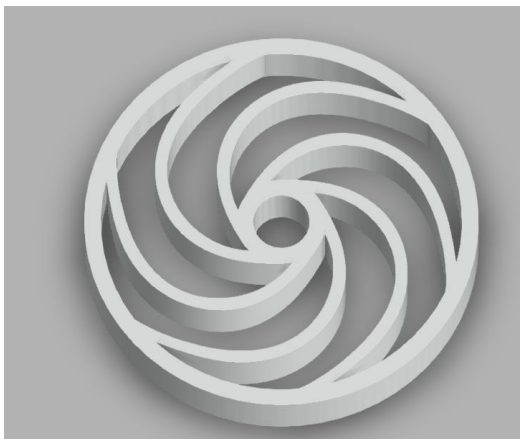
One issue I had with this setup, is that the Data pin needs a pullup resistor, as the controller can only set it to low. I had found this out earlier, and based my circuit design on the fact that I would use an internal pullup resistor for this pin. However, when I did that, it turned out that the bits were not able to transition quite fast enough. This was solved by changing to an external pullup resistor (which made the board look like quite a mess). Refer to *Pull-up Resistor On Data Line* in the Appendix.

Flexi-Wheel Design

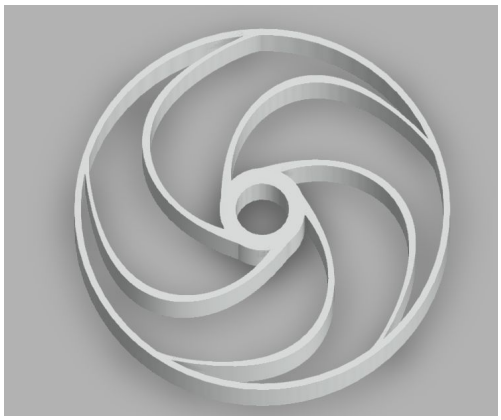
For any RC car, suspensions are a must. Specially for off-road enthusiasts. Since I wanted my RC car to be flippable, I would have to have suspensions on both top and bottom. However, this would turn out very bulky and may not work that well. My Computer Engineering teacher, Mr. Rampelt, suggested that I make a wheel design where the wheel *itself* has the suspension *built-in*, like Michelin's Airless tires.

The first version I created proved to be too rigid when 3D printed. I fixed this in version 2 by changing the, thinning them out, and reducing the number of spokes. Unfortunately, this 2nd version was not flexible enough, as one of the spokes snapped after I put too much pressure on the wheel.

To get a grip from the floor, I had put a thick rubber band on the outside of the wheel.



Version 1



Version 2

Software

An explanation/walkthrough of the PS2 protocol can be found on in the README.md file on my github, at https://github.com/AbhinavA10/PS2-RC-Car. This is the file I used to record all my research and notes about the protocol. It is recommended that the README.md file be read as a supplement to this report.

Comments with explanations are also in my main code/file. I created all 4 of the RCCar.c, RCcar.h, PS2RCcar.c (main), and the PIC16F1459.c files, as they were specific to my new circuit board.

Interfacing the Controller

A long amount of time was spent in understanding and resolving the PS2 Protocol. It started off very confusing, but as I read more and more, it got clearer. I used multiple sources, as indicated at the bottom of the README.md file.

I was very fortunate to find one webpage (CuriousInventor) that had an overview and in depth explanation for an original PS2 controller. I had found this website in October 2017. Luckily I had saved a copy of the page onto my desktop, as when I went back a week later, the site was remodeled, rendering the link broken.

Oscilloscope Reading Results

The combined image of data transfers from the oscilloscope can be found in the Appendix under *Oscilloscope Results.* Below will talk about ideas, referring to the image.

In my first attempt to get the PS2 controller working, I was using a wireless Logitech PS2 controller. As nothing was working, I decided to connect my circuit to the scope to find out if data was being sent back and forth between my board, and the receiver of the controller properly. Essentially, working backwards.The first image shows the Attention Line being pulled low, the next 5 images are byte transfers, and the last image shows the attention line being pulled high again. After taking snapshots of the 4 lines (Data - Yellow on the scope, Command - Blue, Attention - Pink, and Clock - Green), I got the following results.

Byte	Attention	Clock	Command	Data
1	Goes low	✓	0b00000001	0b11111111
2	Still Low	✓	0b01000010	0b01000001
3	Still Low	✓	0b00000000	0b01011010
4	Still Low	✓	0b00000000	0b11111111

5	Goes High at end	✓	0b00000000	0b11111111

If we were convert these binary values that show up on the scope to hexadecimal, we would get a table that looks like this:

Byte	Command	Data
1	0x01	0xFF
2	0x42	0x41
3	0x00	0x5A
4	0x00	0xFF
5	0x00	0xFF

As explained in the README.md file, this was exactly the data I expected to see! This showed the data was in fact being transferred correctly. I switched to a wired controller, and my basic program worked! It turned out that the wireless controller and wireless receiver weren't syncing with each other: not something that I am capable of controlling.

Summary

Overall, this project was very successful. I have learned many things from this project, such as how to design a circuit board, and what it actually means to "reverse engineer". If I had the chance to, I definitely would like to create something like this again. I was able to demonstrate the sensing of digital buttons by playing a music note when pressed, and demonstrated the sensing of analog joysticks by turning a servo with the x-axis of the right joystick.

Future Improvements

Voltage Regulator

Currently, to "convert" my 5V power supply into 3.5 volts, I am using a 200Ω resistor. The ideal value would be 75Ω at 20mA, however the controller does not seem to draw that much. Now, I am aware of how poor a solution this is to changing the voltage. One improvement, would be to add a voltage regulator, such as the LM317Tz.

Using the formula VO = VREF (1 + R2/R1), I have calculated some values for R2 based on the following.

- VO = 3.3V
- VREF = 1.25
- R1 = 220Ω or R1 = 330Ω

However, more research would need to be done into this part of the project to be able to properly implement it.

Due to the above improvement, and the external pullup I had to place, I may want to design a new board overall.

Wireless Controller

Another improvement to this project would be to create the actual RC car, and have this interfacing method, control the RC car. Since I would be making an RC car, I would also have to look into why a wireless controller wouldn't work.

Another issue is that the current wired controller has a very large center deadband area, and is left with little space to actually change values. I could look into adding a lookup table to reduce the sensitivity near the edges of the controller.

I look forward to adding onto this project next semester during Robotics Club.

References

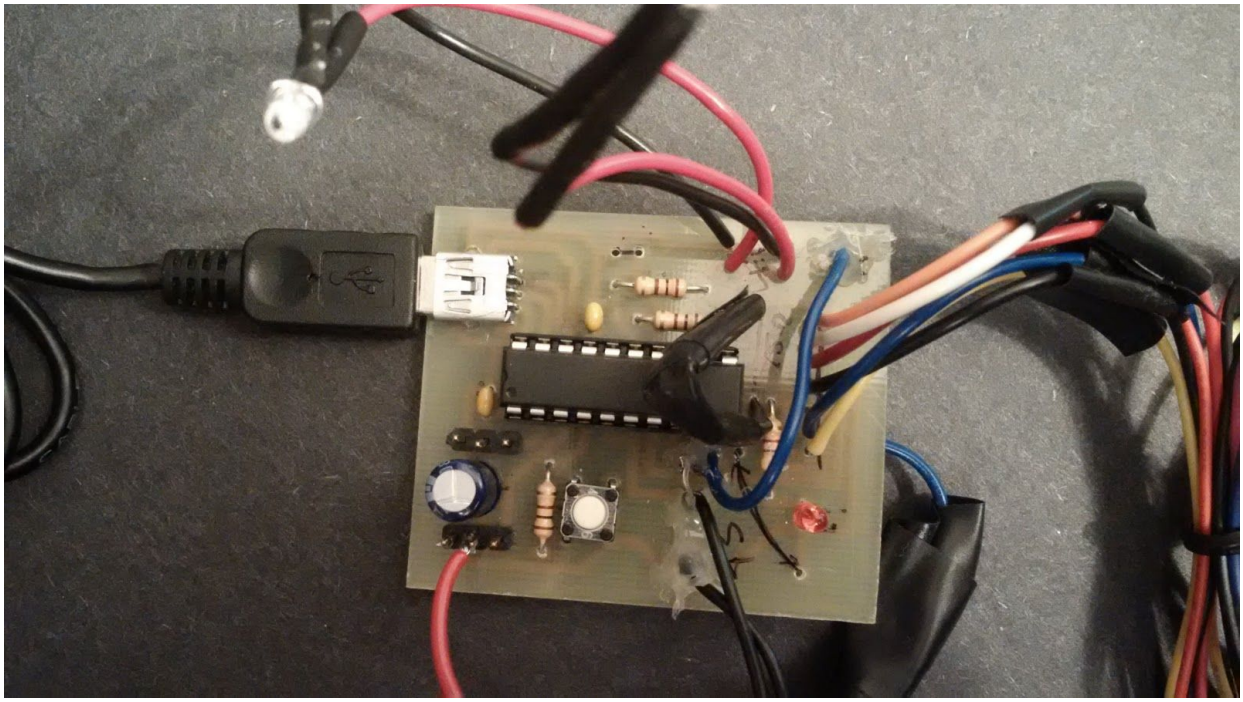
My sources are in the README.md file on my github

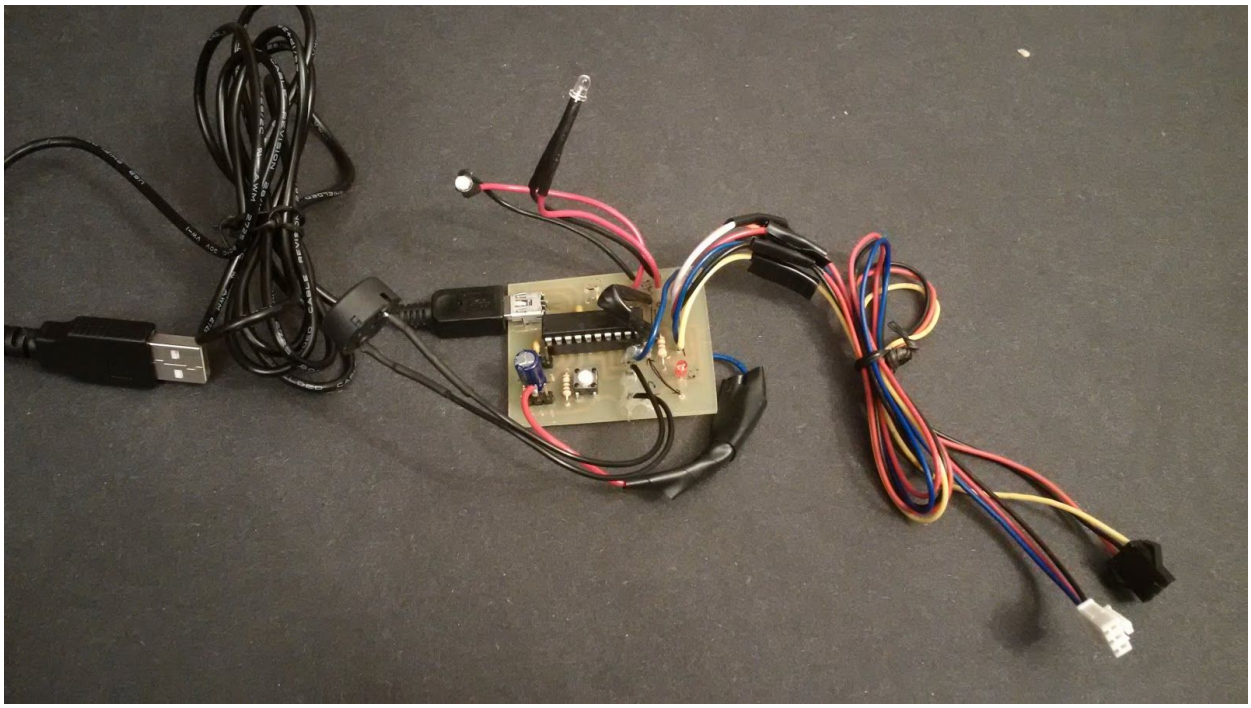
Appendix

All additional information relating to this project (data sheets, sketches, schematics, technical drawings, and oscilloscope readings) are in the docs folder on my github here.

3D Printed Wheel Designs

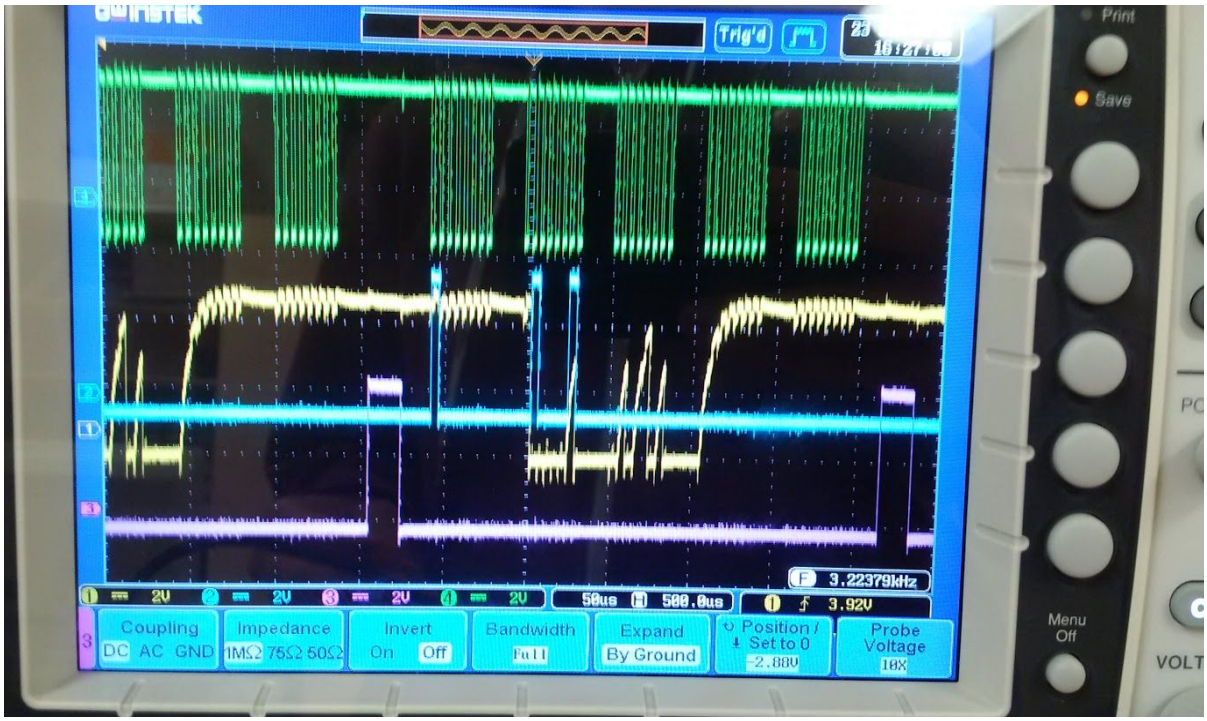
PS2 Interface



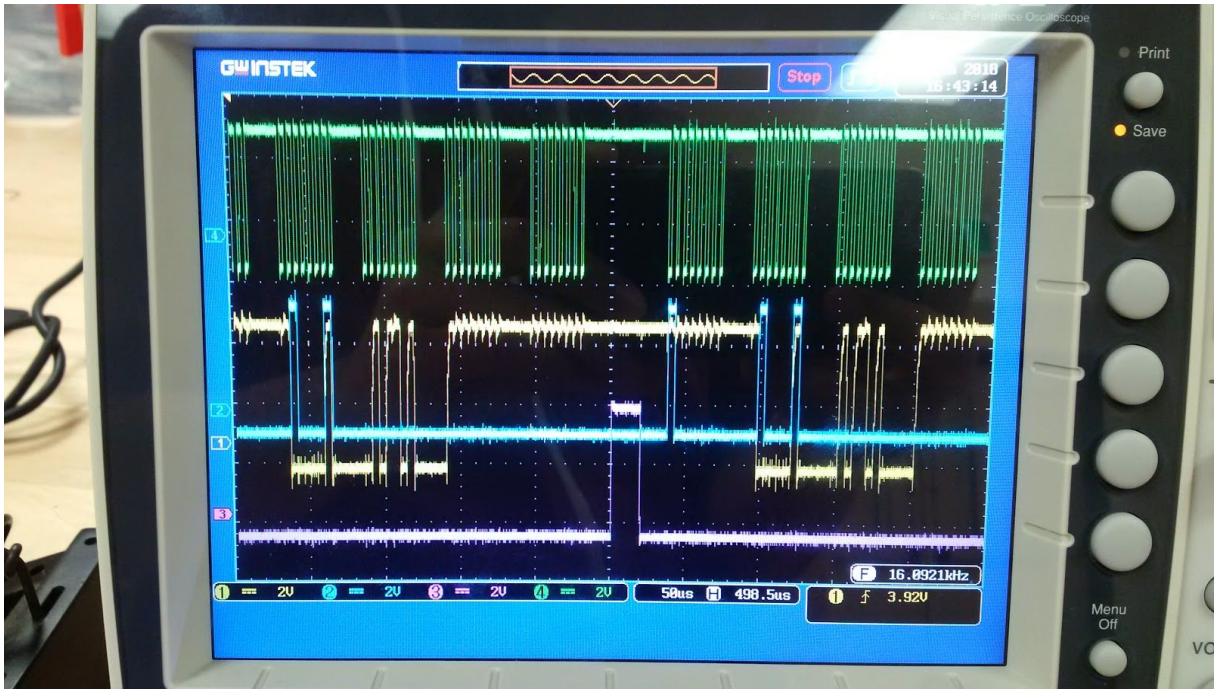


Pull-up Resistor on Data Line

The data line is yellow on the oscilloscope.
Using an Internal Pullup, waveform is curved, and data is not set in time:



Using an external pullup, waveform is square, and data is received properly:



Oscilloscope Results