User-interface for DTU solar car

DTU ROAST peripherals team - TOAST

Sebastian Zeest Rydahl (s173931)
Louise J. Sternholdt-Sørensen (s173937)

Special Course

2021

DTU Electrical Engineering
Department of Electrical Engineering

User-interface for DTU solar car, DTU ROAST peripherals team - TOAST

Report written by:
Sebastian Zeest Rydahl (s173931)
Louise J. Sternholdt-Sørensen (s173937)

Advisor(s):
Jens Christian Andersen
Claus Suldrup Nielsen

DTU Electrical Engineering
Technical University of Denmark
2800 Kgs. Lyngby
Denmark

elektro@elektro.dtu.dk

Project period:	1 Februar- 26 Juni
ECTS:	5
Education:	Special Course
Field:	Electrical Engineering
Class:	Public
Edition:	1. edition
Remarks:	This report is submitted as partial fulfillment of the requirements for graduation in the above education at the Technical University of Denmark.

Abstract

This paper will create an overview of how we designed and created a prototype user-interface for DTU solar car. A short introduction will be given as to how this fits into the DTU ROAST solar car project. This paper goes into detail on both the Hardware and Software side of things and how we approached the problem solving aspects and what design considerations we had. Finally the paper will cover how the COVID situation affected the state of the project and what is needed for the project to develop further.

The purpose of the paper is to create a steering wheel prototype and show the design process and what needs to be considered when making a new kind of layout for vehicles. It also serves as the foundation for any group to continue working on the project without needing to start from scratch.

Contents

CHAPTER 1
Introduction

This project describes the design process behind the steering wheel design for the DTU ROAST solar team car. The steering wheel is the drivers way of interacting and communicating with the car besides the pedals. It is also critical in controlling the car and ensuring optimal conditions.

DTU ROAST is the DTU roadrunner's solar team. It has three subgroups, electrical, software and mechanical, divided as such to better manage the different challenges of a solar powered car. The wheel design is part of the peripherals team (also dubbed TOAST), which is a subteam of the electrical subgroup. The team was formed late 2019, and a first version of a steering wheel design has already been created. This report aims to make a comprehensible and clear-cut documentation of the steering wheel so future team members can easily understand and improve upon the work. It builds upon the findings of the report made by Lukas Fredrik Cronje and Louise Sternholdt-Sørensen[CS20].

Originally the team and car was supposed to take part in the Bridgestone World Solar Challenge 2021[Cha20] but it was decided to postpone the participation as COVID-19 made it impossible to complete the car on time and have it transported to Australia. The competition itself was cancelled later on, meaning the new goal is to compete in the 2023 competition. Before then, the car has to be tested with safety cars on danish roads as well as closed-off road sections. The steering wheel design is a combination of the team wishes and regulations given for the 2021 challenge. As no regulations have been published for the 2023 competition, the regulations may be subject to change but will for now be assumed to be the same.

As mentioned earlier, a steering wheel helps the driver control different parts of the car. Everything from indicators to indicate a turn, the horn for accident aversion and since no dashboard will be built into the car, all controls have to be mounted on the steering wheel. This includes all buttons the driver would normally find in a center console, as an example, the hazard button. Therefore it is crucial that the steering wheel is easy to operate, communicates efficiently across the CAN BUS network and is simple in its design, to prevent distracting the driver.

1.1 Objectives

Our Objective was to:

- Make an interface for the users to interact with the car
- Research optimal designs for driver safety and ease of use
- Connecting and implementing it with the CAN bus
- Find low power alternatives for current planned construction and their benefits/-drawbacks
- Power optimization

1.1.1 Missing hardware

DTU DANSTAR were helping PCB prototyping of the PCB that connects the individual nodes in the CAN BUS network. Due to COVID-19 and unforeseen circumstances, it was not possible to get the PCB design ready so that it would arrive in time. This means the Arduino MEGA used for controlling the steering wheel interface, could not connect and communicate with the CAN BUS hub. In addition to the PCB design not being ready, the display modules for the new design did also not arrive in time.

CHAPTER 2
Interface design

2.1 Introduction

This section describes the thought process and design considerations around the physical design of the steering wheel. The section compares the previous design with the new design and specifies why certain design considerations were changed.

2.2 Previous wheel design

The original design is from an introductory project about the steering wheel of the solar car. The report was made by Lukas Fredrik Cronje and Louise Sternholdt-Sørensen[CS20].

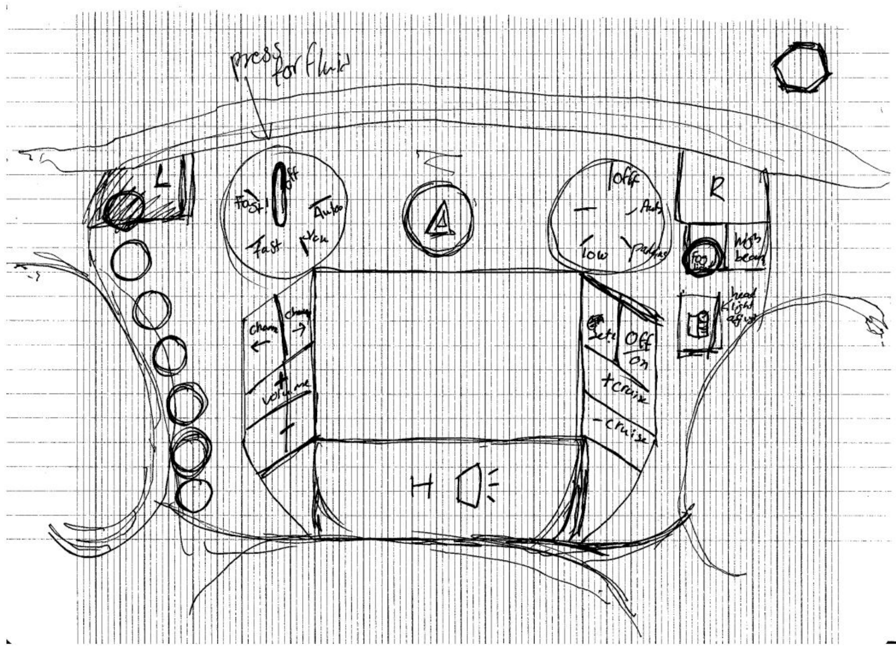

Figure 2.1: First prototype design made in the introductory project, made in 1:1 scale for the components

In this design the primary design consideration was implementing a touch display on the center of the steering wheel. This display would show information about battery status, speed, sensor data as well as allow for deeper insights into how the whole system is operating.

2.3 New wheel design

To streamline and optimize the usage and functionality of the wheel, features have been limited to what is only deemed necessary for the driver. For the new design, the touch display has been deemed not important for the driver. The display is too distracting for the operator as well as difficult to operate while driving as it requires the driver to take their attention off the road for longer. Instead a new design has been proposed for this project.[VW06]

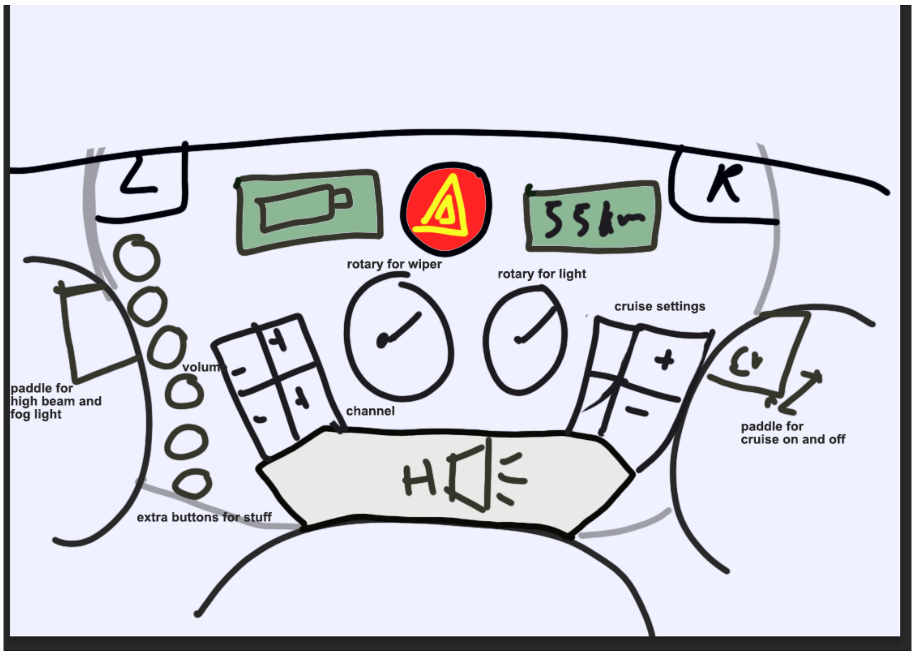

Figure 2.2: Proposed new design

This new design keeps only the parts relevant to the driver. All driver controls are placed on the wheel as a dashboard wont be available in the solar car. The steering wheel features 2 rotary switches to control what type of lights the car should use and the settings of the wiper. To the left of the left rotary switch is a button for blinking the left indicator lights and vice verse to the right of the right rotary switch. On the right side, controls for cruise control can be found. These controls make it possible to increase, decrease and set the cruise control speed. The paddle on the back right side of the wheel is used for toggling cruise control on and off. On the left side is another paddle, this is for flashing the high beam while pressed. On the left front side are controls for volume of the speaker placed in the bottom center of the wheel, as well as buttons for switching radio channels. At the top center is of course a hazard button and to the left, a 6 digit display will me used to give the driver information about optimal speed for battery efficiency and on the right will be a 6 digit display with the vehicle's current speed. On the left there are extra buttons, without assigned functions.

2.3.1 Button assignment

The steering wheel design has a lot of buttons that have counterparts that have not been developed or added to the car. As an example, a wiper rotary switch but no wiper. They are there because they were a design requirement from the team. The team also requested buttons without a specific purpose, for the purpose of adding extra functionality to the steering wheel in the future. In that case, these buttons can be used instead of having to redesign the wheel. Currently only one of these extra buttons is configured in the code for the prototype steering wheel, but should more need to be added, the code and setup can simply be copied.

2.3.2 Wiper

The windshield wiper is used to wipe away rain and other things on the windshield that can obscure the vision of the driver. If the wiper is too fast, it can make high pitched ”squeak” noises as the wiper drags over the windshield, which can distract the driver. On the other-hand if the wiper is too slow, it won't be able to wipe away the obstructions.

We have 5 different states for our wiper, off, automatic, slow, normal and quick. In a perfect world you would be able to only have automatic and off, the automatic being able to sense the current conditions perfectly and use the proper speed. The way an automatic wiper works, is by having a rain sensor with an infrared beam reflected on the outside surface of the windshield into a infrared sensor array, so when the beam is being obstructed e.g. by rain, the wiper turns on. The speed of the wiper is adjusted according to how much of the beam that is being obstructed. [aut68].

However in case of a sensor error or simply for preference sake, we have also included the 3 other states. Slow is when the wiper stops momentarily between each swipe, normal and fast is were the wiper keeps wiping but at different speeds. Some cars even have a rolling wheel for the speed of the wiper. Generally there isn't a standard for how many states, and how fast the wipers should be in the normal and quick, but usually they have a intermittent setting (slow), and a continuous setting(normal) and another continuous but with a higher speed(fast). [Nic21]

2.3.3 Light

For the light we have 6 states, Off, Automatic light, Low beam, High beam, Parking lights and Fog lights. This is the standard for most cars[mag19]. Off is of course off, then there is low beam, which is also called dipped lights. It is a low beam, which means it is short beam of light that is angled towards the road and covers around 40 meters. It is required for cars both in Australia [LR06] and Denmark [Teo], to be on when there is little or no light.

As for the regulations for the now cancelled Bridgestone event, headlamps are not required[Cha20], but since we plan on testing it on the danish roads they are a must. The high beam is a stronger beam and points straight and has a coverage of around 100 meters. They can blind other drivers and are usually only used in rural areas with no other cars. The parking lights activates the tail lights, side lights as well as the license plate lights, without turning on the front lights. As the name suggest it is used for when the car is being parked, or if you are parked or at a standstill in a location with poor visibility. Fog lights are used when there is low visibility on the road, usually in extreme weather.
Automatic lights are automatically adjusted lights that are controlled by the car, using light sensors and sometimes a small camera to assess the lighting conditions and will automatically set the light setting.

For the indicator lights, the prototype is designed such that neither of the left and right indicator lights can be turned on at the same time without pressing the hazard light. If one indicator light is turned on and the button for the other is pressed, the currently blinking indicator will be turned off before the other is turned on.

2.3.4 Space

The first concept of the design 2.1 was made in a 1:1 scale on an A4 paper. The plate which the buttons and display are supposed to be on is approximately 120mm in height and 180mm in length. Since there were more components on the older design, space was not seen as an issue.

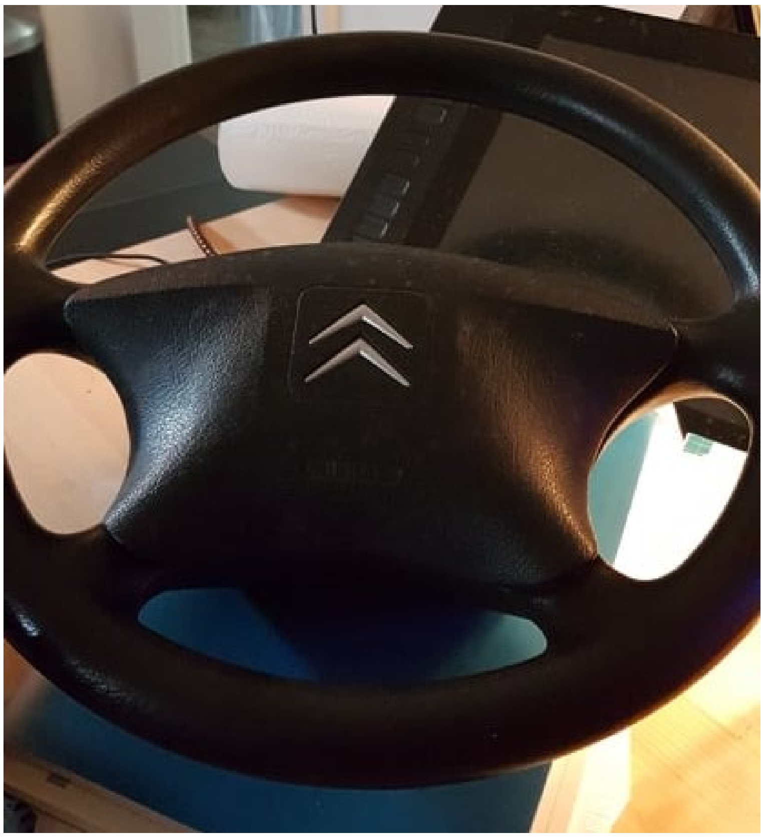

Figure 2.3: The repurposed Citröen steering wheel, with a removeable plate

The rotational switches with their knob are 15mm in diameter, and the CAD part around can be adjusted, which we assume is only gonna be around 45mm in diameter at worst. The Arduino is going to be behind the plate.

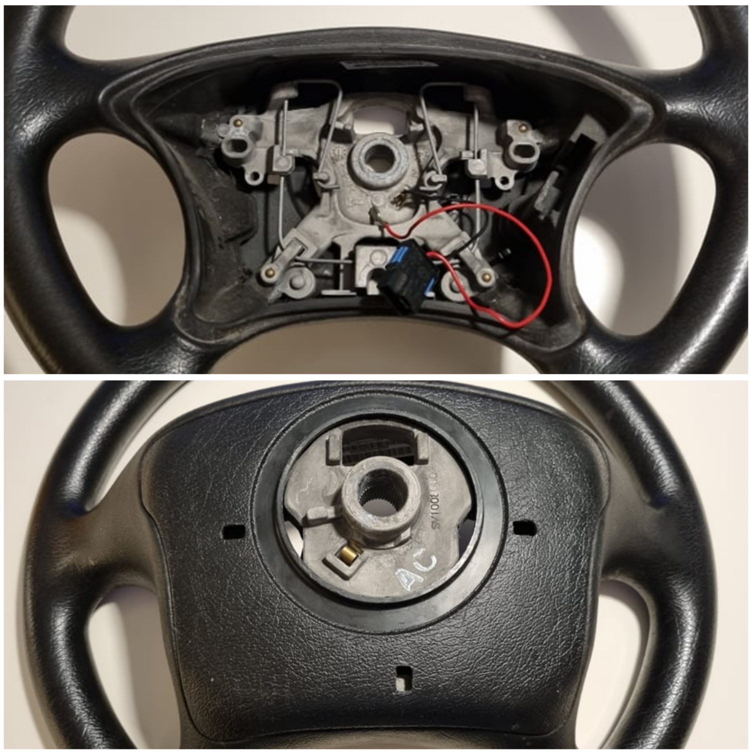
Figure 2.4: The inside of the steering wheel and the backside

This whole setup will be connected above the steering column, with wires coming out the hole on the backside. This means the ground connection, the voltage rail(s) and CAN High and Low (the CAN network connection).

Since we use a refurbished wheel it has the standard form for a steering wheel connector and due to safety reasons it will be connected with a quick release since it is required by the competition [Cha20]. For that, a universal steering wheel quick release hub can be bought.

2.3.5 Noise

Most of our circuit as seen in Figure 3.2 are connected with resistors to avoid noise. We also added a 100ms delay in the code, and all this has eliminated most noise from our circuit. This is required due to the poor buttons used on the car, better buttons could eliminate this delay completely. It could also be mitigated by switching out our breadboard with a prototyping board or making a PCB, since the noise we experienced occurs when there is a small movement near the board.

2.3.6 Rotary switches

We use rotary switches for the light and wiper, both of them being 1 pole 8 way switches. This means that we can move our pole in 8 different positions, giving us the ability to trigger 8 different inputs if needed. We have only connected as many positions as we need, so we can switch to positions where we don't have a connection. To prevent this, a physical prevention has to be designed.

2.3.7 Displays

A speedometer is essential for the driver to stay within the legal speed limit but also to cross-reference the current speed with that of the calculated optimal speed. Thus two displays would be needed to get the job done. A single LCD matrix display could achieve this, but then power considerations come into effect as well as legibility of the display. For optimal reading, ordinary 7-segment style layouts will be used for the displays.

2.4 CAN bus implementation

CAN stands for Controller Area Network. It is equivalent of a nervous system in a car, it helps every device in the car communicate with each other, without having a brain/host computer.

2.4.1 CAN BUS messaging

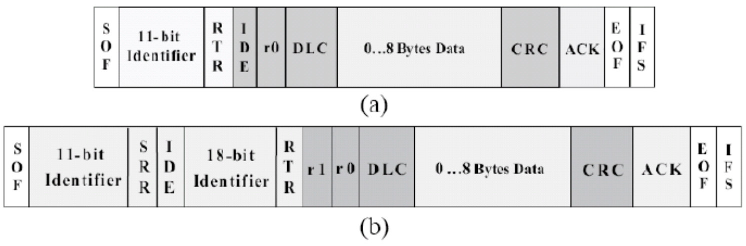

Figure 2.5: Frame of a) standard CAN and b) extended CAN[Lun+10]

As seen in 2.5,the CAN message is made up of 8 different segments. Depending on the protocol, the length of the identifier segment can vary. The standard uses 11 bit, and the extended has 29 bits, our network is standard CAN.

We use a modified version of a function that was made in the before mentioned CAN BUS report[CS20] that is called CreateData. We pass on two integers from our main script, one to define the ID and one to define the state. We could pass on more information than that since we have made our dlc (Data length code) as long as possible, which is 8 byte long. For now, the transmission is all 8 bytes but not the full 8 bytes are used, to show it is a possibility. Since an int is two byte, we can only send 4 different integers. The dlc have to be defined before the data is declared, since it works like an array. Since the CAN BUS can only read one message at a time, we give them an ID, the lower it is the higher priority it is.

```
void CreateData(int ID, int STATE) {
    canMsg.can_id  = ID;          //The ID - for each component, the lower - the m
    canMsg.can_dlc = 8;          // Data Length Code
    canMsg.data[0] = STATE;      //State
    canMsg.data[1] = 0x00;        //Set rest to 0
    canMsg.data[2] = 0x00;
    canMsg.data[3] = 0x00;
  }
```

2.4.2 PCB design

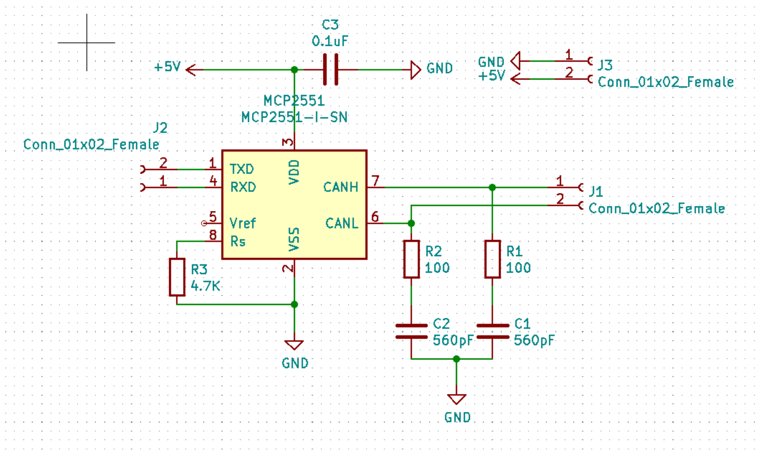

Figure 2.6: PCB design of the CANBUS connector

The PCB is a adjusted version from the CANBUS report[CS20], that is designed to easily connect an MCU to the rest of the CAN network. This should make it possible for people with less knowledge of the network, able to use the code from the CANBUS report[CS20] and connect their systems to it, and thus be connected to the rest of the car. It is around 25x22mm and has 6 connectors, 2 of them for 5V and GND, 2 for the CAN bus network, the High and low, and 2 for the microcontroller connect TXD and RXD, to transmit and receive.

2.5 Co-driver

The display design from the initial steering wheel design should not be discarded but rather re-purposed instead. The proposed usage for the touch display will be as a diagnostics and statistics tool in the hands of a co-driver. A little box with electronics and the touch display attached on top could then be tethered to the CAN network. This means it would still be possible to perform any diagnostics needed on the car while the driver is operating the vehicle as well as remove all the distraction from the touch display, from the driver. It would also make diagnostics easier as it would not have to be done through the steering wheel but could be placed wherever a cable can be run within the vehicle.

2.6 Safety considerations

The most important part of the new design is to make the drivers job as simple as possible. Giving the driver only the information critical to operating the vehicle is of utmost importance in the case of the solar car.

The vehicle is not road legal and can therefore not be tested on public roads before the competition in Australia without safety cars. In Australia, the vehicle will not be driven before competition start. In addition to this, Australia is one of the few countries with left-hand traffic. This means the steering wheel will be located in the wrong side of the car giving the driver worse visibility.

The unfamiliarity of controlling the vehicle from a different side of the road is one of the biggest reasons for a simplified wheel design. In addition to the vehicle being different to operate compared to right-handed traffic, the vehicle is not equipped with the same safety features a modern car is usually equipped with, primarily crumble zones and airbags. This means the car has to be designed for preventative and evasive maneuvering and thus easy control of the wheel is important. On top of the unfamiliarity of the vehicle design, unfamiliarity of the driving environment is also an important factor to consider. Other drivers behavior as well as signage can be different from what may be experienced driving on the road in Denmark, requiring more attention and focus from the driver. This means the driver has to constantly stay very alert and taking the eyes of the road should be kept to a minimum. This is what the new design aims to improve.

The wheel has been equipped with physical buttons and dials. Operation of touch interfaces does not allow for the operator to easily know what element is being pressed or interacted with without visually confirming it first. Using a touch display will in most cases require the driver to take their eyes of the road for longer. Physical buttons and switches can be more easily remembered and can additionally be felt before being pressed/rotated. This allows the driver to learn the wheel controls and only look at the LCD displays that show current speed and optimal speed.[KP13] [RBB12] [YLC13]

CHAPTER 3

Electronics design

3.1 Button designs

We made a prototype of the steering wheel as seen in figure 2.2. Besides our objective of having an energy efficient, easy to use, reliable and safe steering wheel, we also wanted our design to be simple since we have to hand it over to others.

Since most of the car and its parts are not made yet, and the PCB delay, we unfortunately could not test it as part of the car, but we could test the behaviour on the arduino IDE and see if we got what was wanted.

3.1.1 Hardware

The circuit consists of 2 rotary dials, 6 momentary push buttons, 2 short lever micro switches, 3 long lever micro switches, 1 5V relay, 1 Arduino Mega 2560, 11 10K resistors, 1 latching 12V led button and 1 latch/momentary button. The micro switches and latching buttons are NO (normally open), to save whatever little power we can.

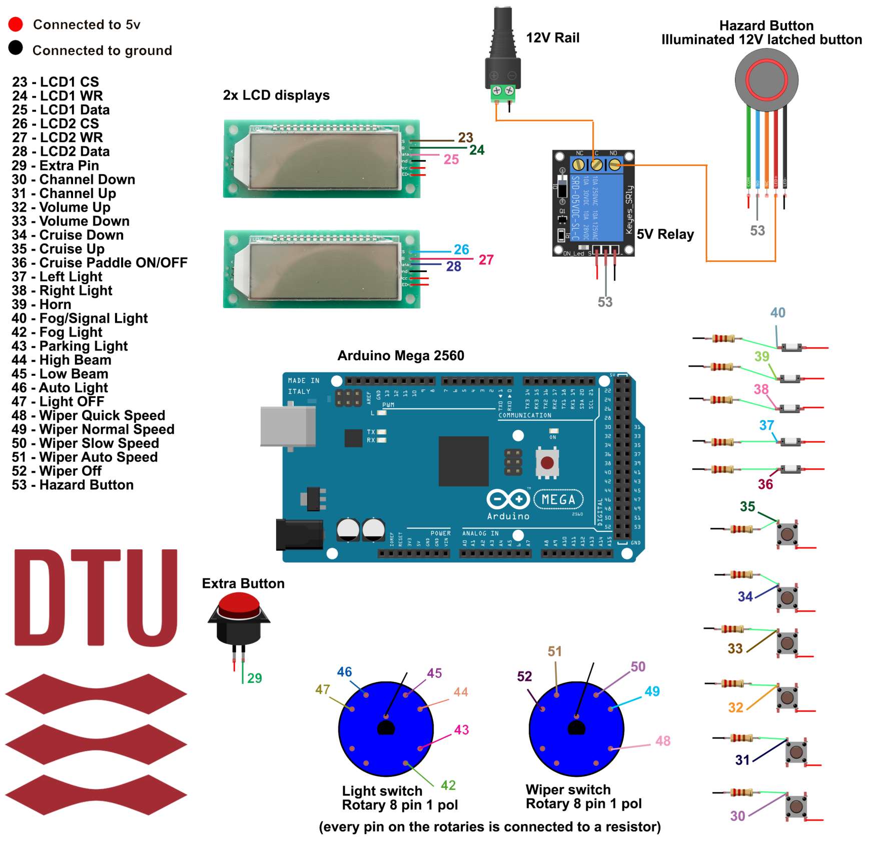

Figure 3.1: Wiring diagram - The numbers on the figure correspond to the equivalent digital output port on the arduino

This enables it to work according to the Bridgestone solar race rules and the wishes of the team.

The Hazard button is a latching push button with a red LED ring [www]. The LED needs 12V, which in the car can be supplied by the 12 V rail. We put a 5V relay that could handle up 250V(DC), before the LED, and connected it to NO on the relay and connected its controlling port to the power to the button signal, so that when the button isn't pushed down, the LED is unpowered, thereby saving power.

The 6 momentary push buttons are single pole single throw switches, similar to those found in a keyboard. They each are coupled with a 10K resistor to eliminate noise. They are used for the channel, volume and cruise up and down buttons.

We also have 5 micro switches, 2 with short arm with a roll actuator at the end, and 3 with long arms. They are also coupled with a 10K resistor each. The 2 with the short arms, are used as left and right indicator. The 3 with the long arm are used as the 2 paddles, the one fog light signal indicator and the other as cruise control on/off, the last one is used as the horn.

The solar team also wanted extra buttons, so we have 1 latch/momentary button that we have added to the prototype.

All these buttons are hooked up to the Arduino Mega 2560 microcontroller[Ard], where all the code is stored. We choose the mega over the uno, because of the extra pinouts, however if needed it could be an uno instead. If we run out of pin outs on the mega in the future there are workarounds such as PISO's or resistors, but then there would be needed to add some extra code on the microcontroller. 4 digital ports as not seen on the schematic 3.1, are reserved for the CAN bus communication PCB. The 4 connections are (sck (serial clock), MOSI (Master Output Slave Input), MISO (Master input Slave Output) and CS (Chip select)), however as mentioned before we had to forego this in the prototype, since the PCB was delayed due to COVID-19.

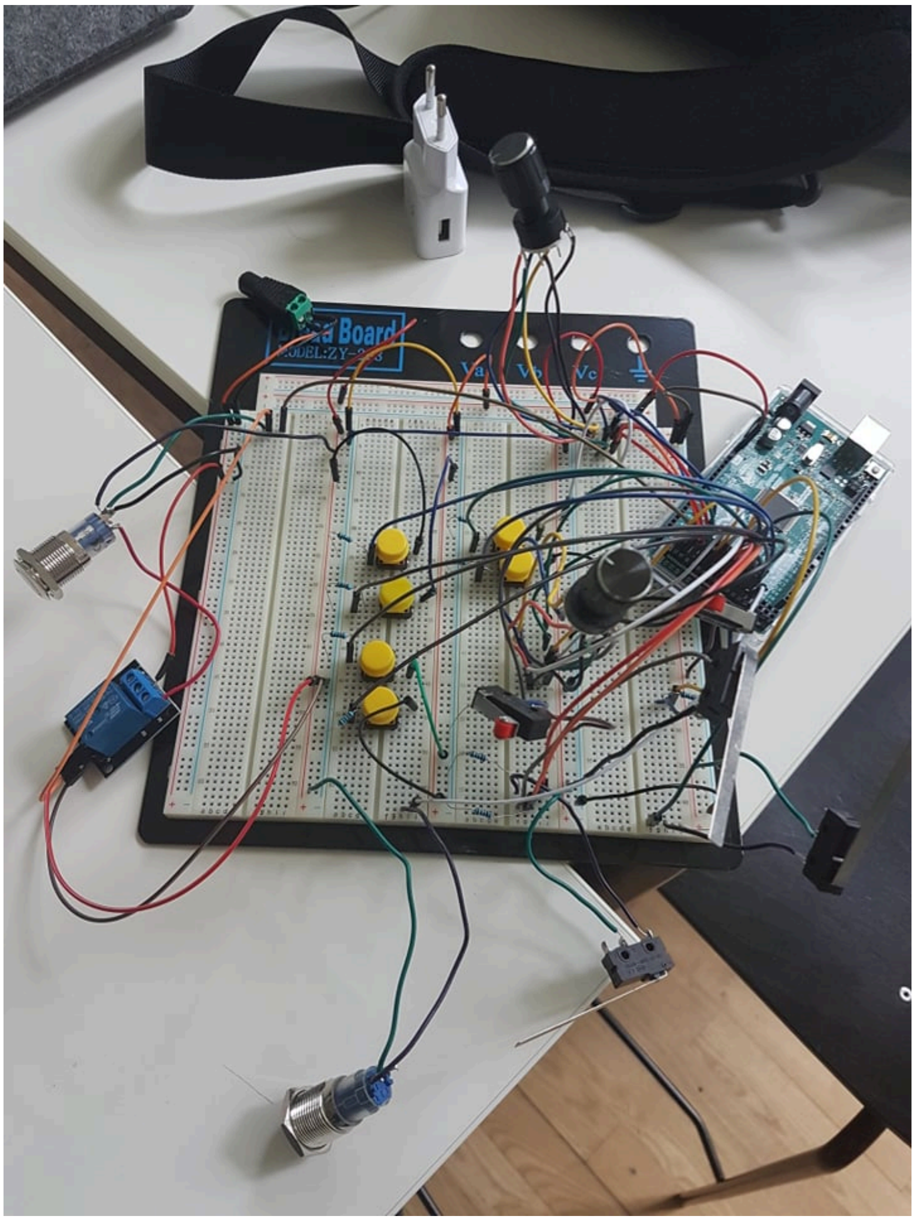

Figure 3.2: The design realized on a breadboard

3.1.2 Software

With the software we kept it simple, since the arduino has loads of space, and we didn't want to risk having the code optimized to a point where it could be hard for someone else to read it. As of writing the sketch (code) only takes up 2% of the storage space, and the global variables take up 8%.

The code for most of the buttons is simple enough, if it is pressed it does x thing. But for example with the rotary dial we had to read the state of the switch and then we have a 6-bit value composed of the bits for each state. This 6-bit value can be represented as a hex value and whatever the hex value is, is what state the dial will be in. The 6-bit value only updates once a new position is reached where the hex value is then calculated and a new state is assigned.

```
//ReadRotaryLight
if (LightState != LightLastState) {
  if (LightState == 0x3E) {
    //LightOFF
    CreateData(2, 0);
    mcp2515.sendMessage(&canMsg);
  } else if (LightState == 0x3D) {
    //LightAuto
    CreateData(2, 1);
    mcp2515.sendMessage(&canMsg);
  } else if (LightState == 0x3B) {
   //LightLow
    CreateData(2, 2);
    mcp2515.sendMessage(&canMsg);
  } else if (LightState == 0x37) {
    //LightHigh
    CreateData(2, 3);
    mcp2515.sendMessage(&canMsg);
  } else if (LightState == 0x2F) {
    //LightNormal
    CreateData(2, 4);
    mcp2515.sendMessage(&canMsg);
  } else if (LightState == 0x1F) {
    //LightQuick
    CreateData(2, 5);
    mcp2515.sendMessage(&canMsg);
  } else {
    delay(10);
  }
}
```

To use our arduino with CAN BUS pcb, we have to install the MCP2515 library?? and the code from the CAN BUS documentation ?? It doesn't take up very much space so that is not an issue.

```
CreateData(x,y);
mcp2515.sendMessage(&canMsg);
```

As before mentioned we use the Createdata to create our message, by sending two integers. The integers are represented with x and y, which is the ID and STATE, to send out the state of our button (if it's pressed or not), and its importance (the ID). For now we have given them arbitrary numbers, but when the CAN network is set up, there has to be common agreement between the team members about what takes priority.

Since the physical CAN bus as well as the CAN bus PCB are not within our possession, the code may not work when implemented as it has not been tested.

3.2 Display design

This section will cover the design thoughts around picking an optimal display for the solar car steering wheel. Due to the component not showing up within our expected time-frame, this section is not as complete as it should have been, particularly lacking a showcase of the code working on a physical display. Instead this section has been expanded to add more details describing the research that went into the design decisions regarding display choice as well as how the display driver chip functions and can be used.

3.2.1 Display technologies

Many forms of display technology exists and all with their advantages and disadvantages. For the purpose of a speedometer that only needs to show digits, simplified and non-customisable displays can be used. For the wheel design it was important to have a display that not only is able to be very power efficient but also work during the day as well as night. The two most common types of displays are based on either LCD or LED technology.[Gee18] Typically many LED branded displays still use LCD technology.[Sil17]

3.2.1.1 LED technology

In its essence, LED displays are powered by small diodes that emits light in a display matrix, making an image visible. Some displays will be branded LED but actually function like LCD displays that use white LEDs for its backlight.

Technologies such as OLED and MicroLED are more popular LED technologies that uses no backlight but instead uses red, green and blue LEDs to emit the light, thus making the display thinner as it requires less optical elements. However, OLED and MicroLED technology is not necessarily as power efficient as an LCD display with an LED backlight, but are more so focused on color accuracy. As a speedometer is not in need of a color display to show the speed of the vehicle, any color display component can be discarded. Instead black and white OLED displays can be utilized. These displays could prove useful for the speedometer application but still requires a complex display driver to function, making them less power efficient.

A more appropriate approach could be using 7-segment LED displays. No other part than the individual segments emit light, theoretically giving you the largest amount of brightness for the least amount of power, given that the diodes in the segments are energy efficient. A 7-segment display requires no display driver and can in theory be controlled directly by turning on and off each segment, however for convenience sake, a display driver is recommended. While 7-segments are likely to be more efficient, they still fall within the category of emmisive displays like the rest of the displays in this category. While emissive displays are good in dark environments, the problem of always being emissive can only be solved if looking at LCD displays.

3.2.1.2 LCD technology

LCD displays usually fall within 3 categories of either reflective, transflective or transmissive. Reflective displays require an external light source that shines on the display. The light will then either be reflected or blocked depending on the configuration of the display. This type of display can often be found in calculators, older clocks, handheld consoles and digital watches amongst others. These displays consumes very little power and can be powered by batteries in some cases for up to years.[USA14]

Transmissive displays are similar to reflective displays with the main difference being the light source coming from a backlight in the display. These displays are the most common LCD displays but also the least energy efficient. Like with LED displays they only function when emitting light. This display type is very typical in modern TVs and computer monitors, many subtypes of transmissive LCDs exist with varying features and technologies that will not be described in this report.

The final LCD display type is the transflective display allowing the features of both transmissive and reflective displays in one. Unlike the reflective display, transflective displays feature a lower contrast due to the half-mirror that needs to both allow reflection of external light but also the emission of light from the backlight. With the ability to enable and disable the backlight of the display, this display allows it to be very energy efficient if an external light source is available and less energy efficient but still on par with standard LCDs, if powered with the backlight.

3.2.2 LCD choice

The solar car will be driving in Australia around the end of October 2023 for the Bridgestone: World Solar Challenge. At that time of year, Australia experiences around 12 hours of daylight [TD21] meaning that as driving only occurs during the day, using a reflective or transflective display would be sensible as the display can be extremely energy efficient. But to allow for increased versatility and if any need for increased contrast should arise, a transflective display will be used to have the option of providing backlight. Transflective LCD displays were also common in older cars as seen here in a 2006 Range Rover Sport HST.

Figure 3.3: 2006 Range Rover Sport HST Dashboard (Off state)[Spy10]

3.2.3 Display design choice

For ease of control, a display driver will be used. The HT1621 from Holtek is a suitable driver for use with a 6 digit LCD display as it can drive up to 32x4 segments with an internal oscillator. A compatible Arduino library built specifically for the display driver also exists making implementation simpler and allowing for easy control of the display.[va21]

3.2.3.1 6-digit LCD display

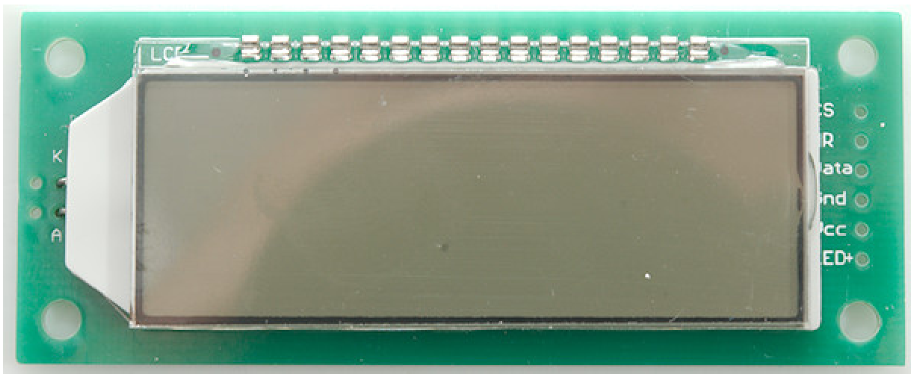

Figure 3.4: HT1621 6 digit LCD display

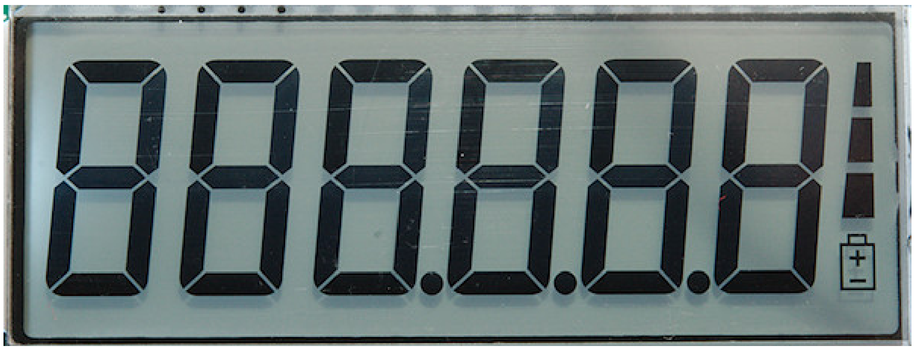
Figure 3.5: HT1621 6 digit LCD display all segments enabled

For the display choice, no proper data sheets exist meaning there is no way to get accurate information about the display. From the stores that sell these displays, an estimate of 0.4mA without backlight and 4mA with backlight is given.[Com21] The backlight value seems very low as a normal single segment on a 7-segment LED display uses around 10mA at the same voltage. It could be that the display has low backlight brightness but without the component there is no way to verify it. The display works by splitting it up in a 12x4 grid to control it with the HT1621 driver.[Jen21]

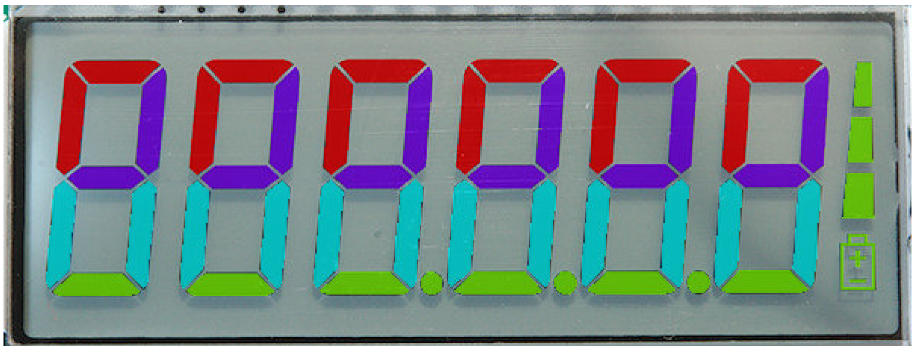
Figure 3.6: HT1621 6 digit LCD display all segmented parts

COM/Seg	0	1	2	3	4	5
COM0	DP1	0D	DP2	1D	DP3	2D
COM1	0C	0E	1C	1E	2C	2E
COM2	0B	0G	1B	1G	2B	2G
COM3	0A	0F	1A	1F	2A	2F

COM/Seg	6	7	8	9	10	11
COM0	BAT1	3D	BAT2	4D	BAT3	5D
COM1	3C	3E	4C	4E	5C	5E
COM2	3B	3G	4B	4G	5B	5G
COM3	3A	3F	4A	4F	5A	5F

Table 3.1: HT1621 6 digit LCD display all segmented memory parts based on 7-segments[Fer17]

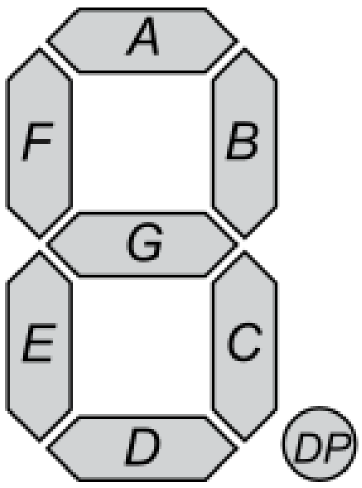

Figure 3.7: 7-segment labelling[Uln18]

Each digit from 0 to 5 are counted from right to left on the display. Bat1 corresponds to the bottom battery icon as well as the bottom bar. Bat2 is the middle bar on the battery icon. Bat3 is the top bar on the battery icon.

3.2.3.2 HT1621

The display driver can drive up to 32 segments on 4 COM lines, essentially making it compatible for a 32x4 segment LCD display. As the 6-digit LCD display is split into 12x4 segments the HT1621 driver is more than capable of driving the display. The HT1621 chip is designed to take in a binary stream of data that is then handled according to what command is sent along with the data. To communicate with the HT1621 chip, chip select (CS), read clock output (WR) and a data pin is used to send a message. To write data to the RAM memory of the chip, the transmission must begin with 101 followed by the data in binary. 101 puts the chip into writing mode. The data that follows are 6 bits for the memory address followed by the actual data.

The wheel design requires a need for 2 displays, 1 for the current speed and 1 for the optimal speed. Since the display only takes up 12 segments of the RAM memory, it should be possible to run both displays via a single HT1621 driver. To achieve this, another display module has to be used, as with the proposed one, the display driver is a part of the display module. To drive 2 displays with this one driver, a new PCB would have to be designed with the two displays attached.

3.2.4 Code implementation

The code implementation is based on the previous CANBUS report[CS20]. It is based on the CAN Receiver Script and adds a few new lines to make the display simple to control. Since the physical CAN bus interface as well as the display modules are not within our possession, the code may not work when implemented as it has not been tested.

3.2.4.1 Display Script

To implement the displays, the HT1621 library will be used. To initiate a display, `HT1621` followed by the name of the display can be used. The script initiates 2 displays, one for the current speed and one for the optimal speed. The script also requires some variables, namely `backlightStatus` that keeps track of whether or not the backlight is turned on as well as the longs with the values for the current and optimal speed.

```
#include <HT1621.h>

HT1621 currentSpeedLCD;
HT1621 optimalSpeedLCD;
bool backlightStatus = false;
long currentSpeed;
long optimalSpeed;
```

Next the displays will be instantiated and readied for use where-after the RAM memory of the two displays are cleared. They should by default be cleared when powering up but this is still done as a precautionary measure.

```
currentSpeedLCD.begin(13, 12, 11, 10); // (cs, wr, Data, backlight)
optimalSpeedLCD.begin(9, 8, 7, 6); // (cs, wr, Data, backlight)

currentSpeedLCD.clear();
optimalSpeedLCD.clear();
```

In the main loop the code is based on the CAN Receiver Script. The values that are checked for are not as of yet known so the values shown here `0x0F1` and `0x0F2` are placeholders. When the values are read, the variables are updated and at the end of each loop, data is written to the RAM memory of the displays using a function `updateDisplays`. This code also assumes only one message is sent at a time via the CAN bus and should in reality only write to one of the display if values change but those are some of the improvements to the code that can be worked on.

```
if (canMsg.can_id == 0x0F1){
  currentSpeed = canMsg.data[1];
}

if (canMsg.can_id == 0x0F2){
  optimalSpeed = canMsg.data[1];
}

updateDisplays(currentSpeed, optimalSpeed);
```

Here is the function that sends new data to the HT1621 chips memory.

```
void updateDisplays(long currSpeed, long optimSpeed) {
    currentSpeedLCD.print(currSpeed);
    optimalSpeedLCD.print(optimSpeed);
}
```

3.2.4.2 HT1621 Arduino Library

The HT1621 Arduino library allows for simple communication with the HT1621 driver via 3 pins, functioning as a decoder from normal numbers to binary data transmission the HT1621 chip can decode. In the HT1621 library, an additional pin can be configured for backlight but is not necessarily required however for the use case of the solar car, it is needed if the readability of the display is too low. The HT1621 library uses the HT1621 chips write mode with successive address writing. This means the library can write all the data for the whole display in one continuous transmission but for ease of use, only 2 segments are written at a time, meaning a single digit is written at a time, starting from the right, going left.[Hol19]

3.2.4.3 Example

To visualize how this work, an Arduino UNO is used to print ”123” on the display. The CS, WR and DATA pins are then all measured with a DS1054Z oscilloscope, where-after the data was saved and visualized here.

```
long num = 123;
lcd.print(num);
```

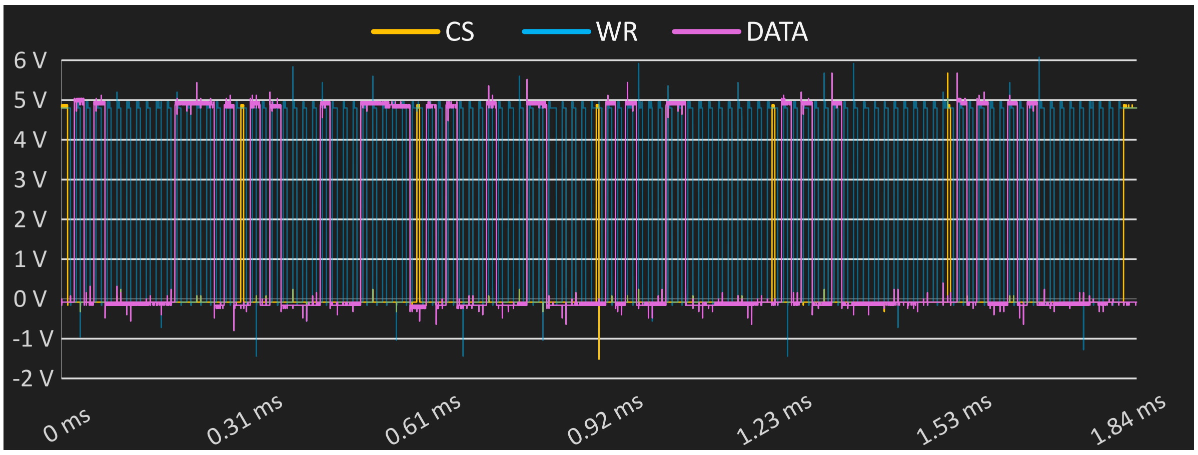

Figure 3.8: HT12621 library transmission when printing 123

From Figure 3.8 it is clear to see that the data transmission happens in 6 sections, one for each of the 7-segments on the displays. Starting from the left in Figure 3.8, data is written for the far right 7-segment on the display in Figure 3.5. The final section on the right in Figure 3.8 is the left-most 7-segment in Figure 3.5.

For each of the sections, we can decode what is being transmitted with Figure 3.7, Table 3.1 andFigure 3.8.

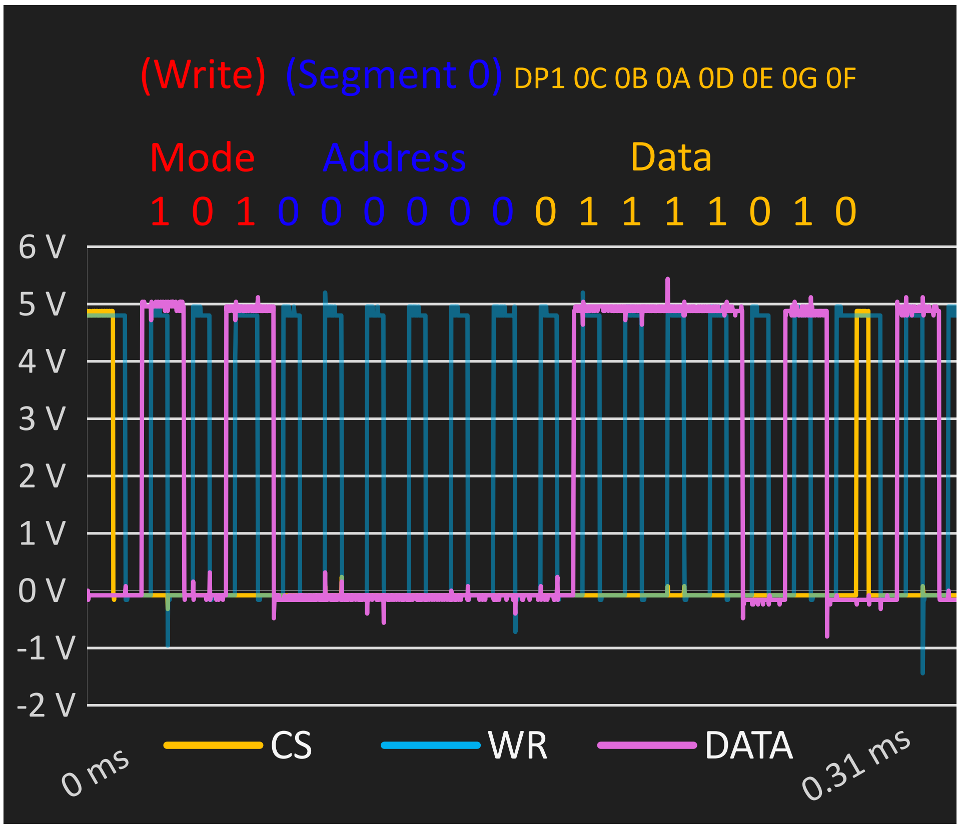

Figure 3.9: Writing to memory location 0 to 7 (Segment 0+1) on the HT1621 chip

The data is written to the first data position on the RAM memory and can be decoded to display the number 3.

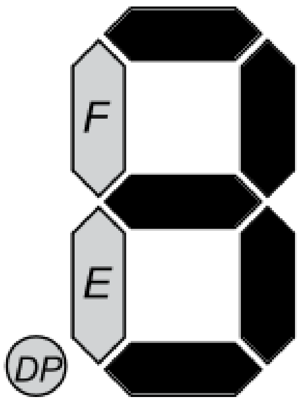

Figure 3.10: First 8 bit decoded to 7-seg format

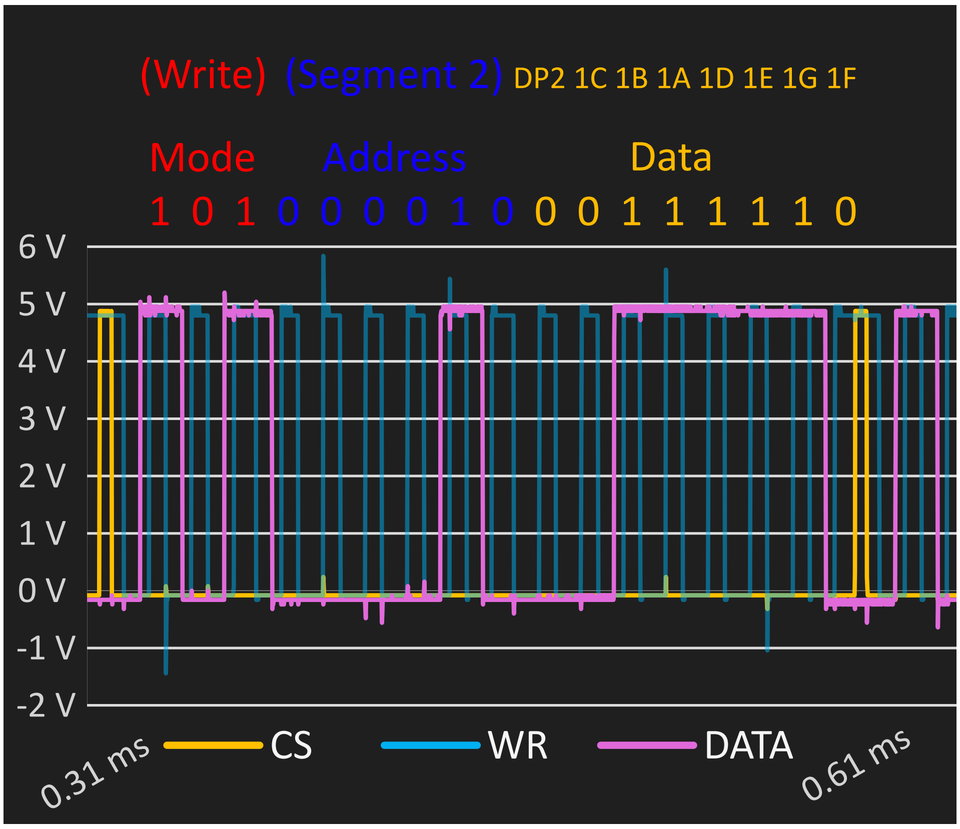

Figure 3.11: Writing to memory location 8 to 15 (Segment 2+3) on the HT1621 chip

The data is written to the third data position on the RAM memory and can be decoded to display the number 2.

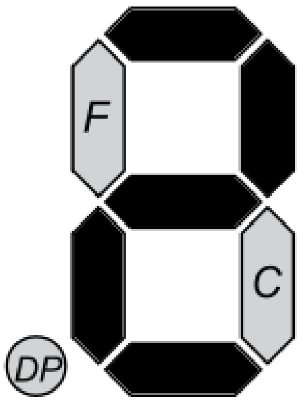

Figure 3.12: Second 8 bit decoded to 7-seg format

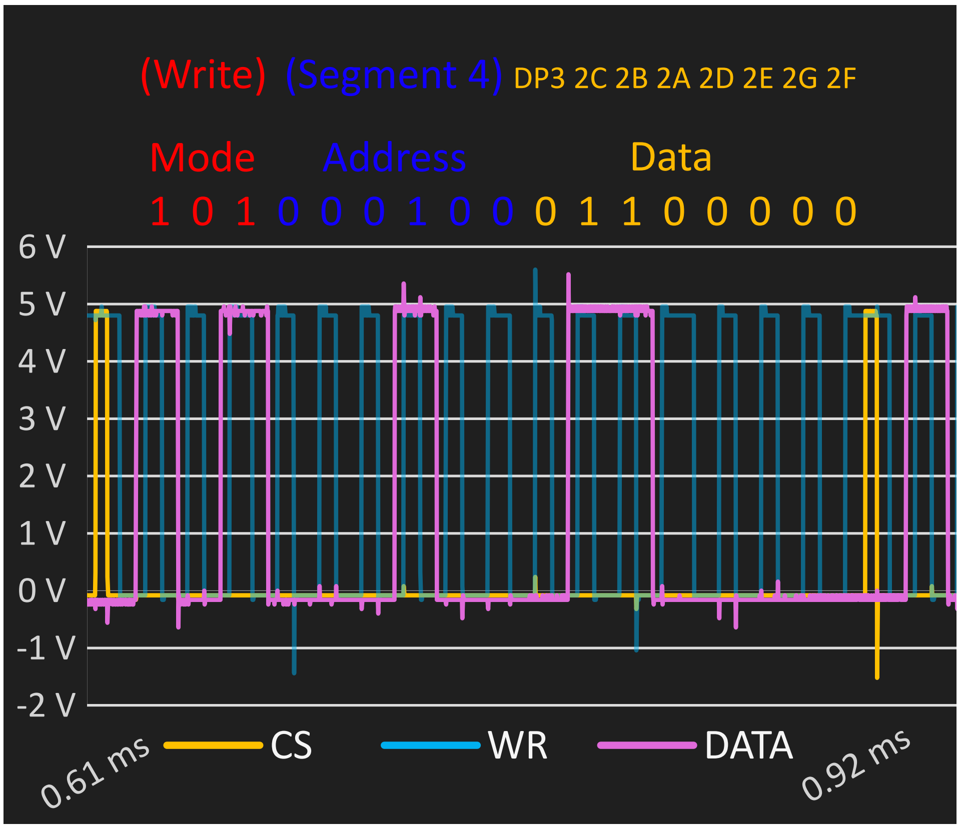

Figure 3.13: Writing to memory location 16 to 23 (Segment 4+5) on the HT1621 chip

The data is written to the fifth data position on the RAM memory and can be decoded to display the number 1.

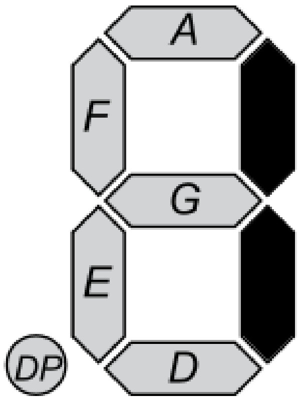

Figure 3.14: Second 8 bit decoded to 7-seg format

The next decoding is primarily only the RAM memory positions as no additional data is written but rather just cleared from the memory.

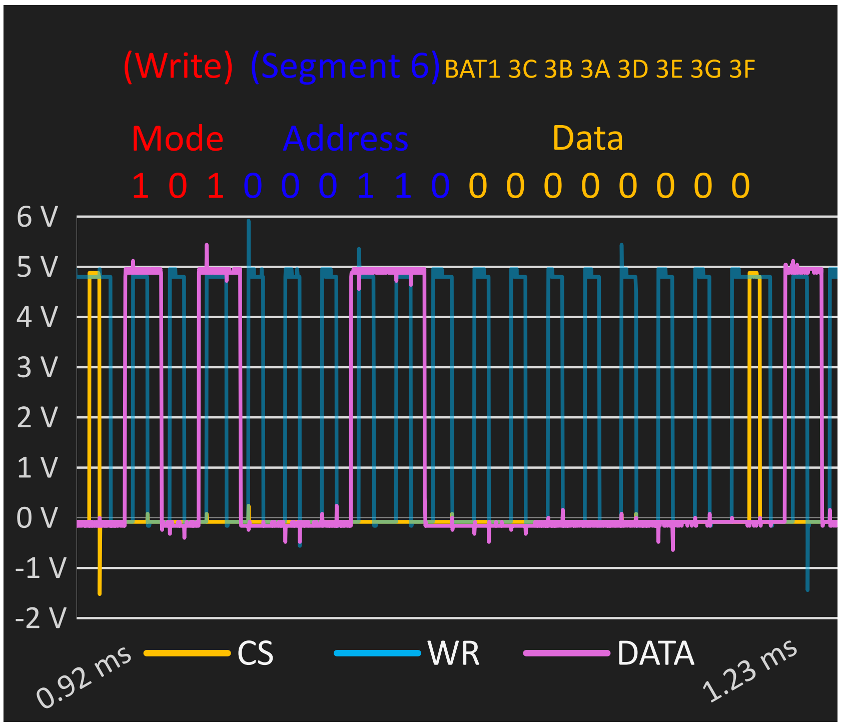

Figure 3.15: Writing to memory location 24 to 31 (Segment 6+7) on the HT1621 chip

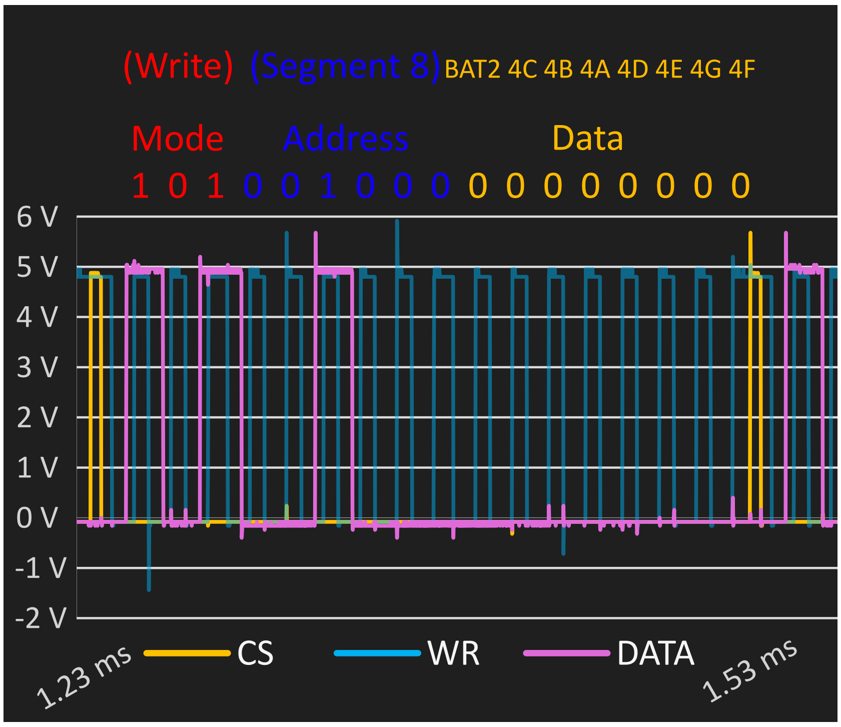

Figure 3.16: Writing to memory location 32 to 39 (Segment 8+9) on the HT1621 chip

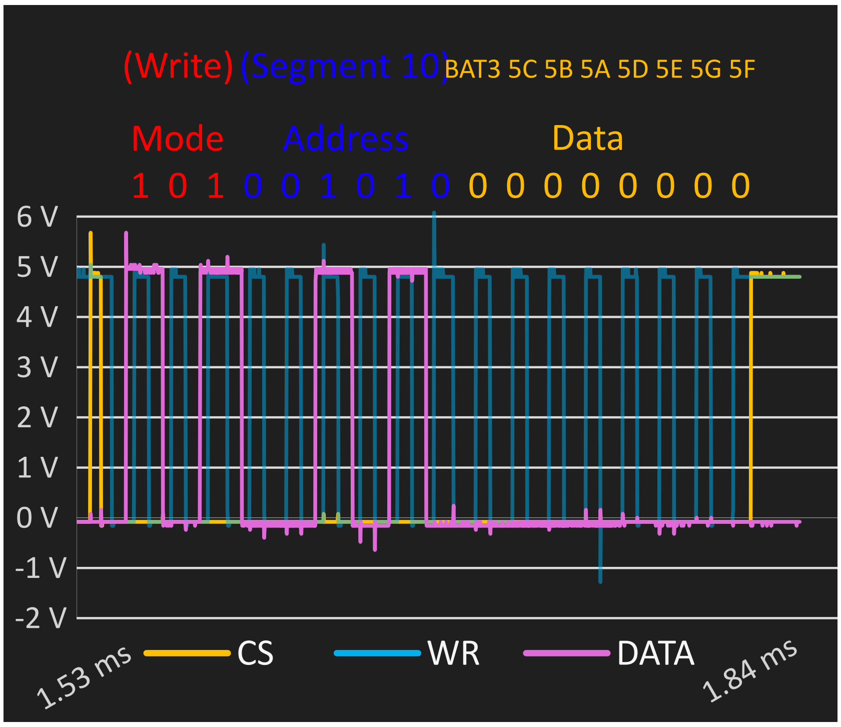

Figure 3.17: Writing to memory location 40 to 47 (Segment 10+11) on the HT1621 chip

Finally the chip will from memory print the data to the display until the memory is once again updated. The rest of the time, the chip itself will use the internal clock to continue updating the display by turning each segment on and off at the correct time while it cycles the COM lines.

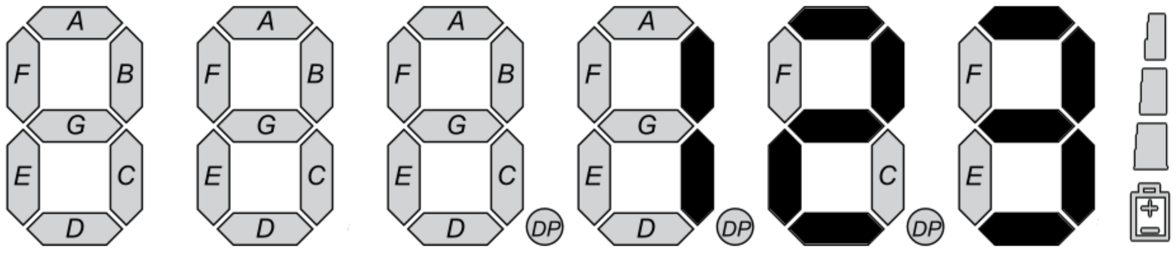

Figure 3.18: Visualization of what is in the RAM memory of the HT1621 chip (black is digital 1)

The Table 3.2 will be what is in the RAM memory matrix of the chip after having written the new data to the chip. X means the data has not changed and is the same as it was before. If it is by default set to 0 then the rest of the RAM memory would be empty.

COM / SEG	COM3	COM2	COM1	COM0	
SEG0	1	1	1	0	0
SEG1	0	1	0	1	1
SEG2	1	1	0	0	2
SEG3	0	1	1	1	3
SEG4	0	1	1	0	4
SEG5	0	0	0	0	5
SEG6	0	0	0	0	6
SEG7	0	0	0	0	7
SEG8	0	0	0	0	8
SEG9	0	0	0	0	9
SEG10	0	0	0	0	10
SEG11	0	0	0	0	11
⋮		⋮			⋮
SEG31	X	X	X	X	31
	D3	D2	D1	D0	ADDRESS / DATA

Table 3.2: The final memory configuration of the HT1621 chip

CHAPTER 4

Discussion

4.1 Test with CAN BUS and car

As a result of COVID, the PCB that acts as the connection between the CAN network and the Arduino, was delayed and meant it was not possible to test the steering wheel with the rest of the CAN network and due to everything on the car being delayed overall there would not be anything to send messages between.

4.2 Code

The code could be optimized, but for now its better to keep it simple instead of possible optimizing it and making it more convoluted. There also is not a problem with space as it is on the Mega as of writing.

For the code implementation of the displays, a more optimal design could be made that takes into account what is currently in the memory of each display so the same data is not written to the HT1621 chip. On top of this, considerations for what happens in case of communication failure should also be taken into account. Currently the display will only update once new data is received via the CAN bus. Therefore a system that checks when the last received data happened. Adding a feature that lets the driver know on the display could also be implemented. In addition to more code features, it would also be possible to not have to rely on the HT1621 library and instead send the exact data that is needed in the RAM memory. This would make it possible to write any symbol that is possible on a 7-segment display and not just numbers that are compatible with the HT1621 library.

4.3 Display

Due to COVID-19, the 6-digit displays did not arrive in time to perform a live demonstration. It was also not possible to buy any HT1621 display drivers on their own and have it delivered fast as neither Mouser nor RS had the component as part of their assortment. Due to not having the physical component an important thing to keep in mind when final assembly takes place is the viewing angle of the LCD displays. As the display modules did not arrive in time, a section around viewing angles is not included in our report.

4.4 Future improvements

Future improvements could be adding a sound module and small speaker to the Arduino, as an example making it possible to play a sound when the turn signal is on. This could be easily accomplished by adding two small speakers and a music maker mp3 shield [Ada]. This could also allow us to add other sounds such as voice saying low battery or other things.

An alternative easy solution could simply be a buzzer, that could be wired up to the turn signal indicators, so that it will sound as the LED/diodes are being turned on.

One improvement that is certain is making a PCB from the prototype.

CHAPTER 5

Conclusion

The original goal was to design a new steering wheel layout that would improve safety and energy consumption. On that front we achieved our goal, but the project was not only about finding a new more optimised design but also implement it and make it work. Due to the whole solar project being postponed due to COVID-19, the CAN implementation could not take place. The report of the previous steering wheel design aimed at simulating the CAN bus communication but was never fully implemented with the rest of the solar car. Thus construction of a CAN system or use of the previous simulation system would have been a necessity if this project was to have been tested as a full implementation. Finally the CAN bus pcb, display components and display driver did not arrive on time and appear to still be in transit. This meant we could not demonstrate the functionality of the display. On top of this, the display featured no datasheet and we would have liked to have a section in our report regarding the claims of the product. Overall we are satisfied with the project but wish more things had worked out so we could have made a more adequate implementation.

Appendices

APPENDIX A
Scripts and schematics

A.1 Steering wheel interface Script

```
#include <SPI.h>
#include <mcp2515.h>


//For the canbus
struct can_frame canMsg;

//Pin assignment for canbus connection
MCP2515 mcp2515(10);


//All buttons is normally open(NO)
//Pin assignment
const int HazardBPin       = 53;
const int WiperOffPin      = 52;
const int WiperAutoPin     = 51;
const int WiperSlowPin     = 50;
const int WiperNormalPin   = 49;
const int WiperQuickPin    = 48;
const int LightOffPin      = 47;
const int LightAutoPin     = 46;
const int LightLowPin      = 45;
const int LightHighPin     = 44;
const int LightParkPin     = 43;
const int LightFogPin      = 42;
const int FogPaddlePin     = 40;
const int HornPin          = 39;
const int RightLightPin    = 38;
const int LeftLightPin     = 37;
const int CruisePaddlePin  = 36;
const int CruiseUpPin      = 35;
const int CruiseDownPin    = 34;
const int VolumeDownPin    = 33;
const int VolumeUpPin      = 32;
const int ChannelUpPin     = 31;
const int ChannelDownPin   = 30;
const int ExtraB1Pin       = 29;
// 41 is broken on my board


//States
boolean HazardBState       = 0;
boolean HazardBLastState   = 0;

boolean WiperOffState      = 0;
boolean WiperAutoState     = 0;
boolean WiperSlowState     = 0;
boolean WiperNormalState   = 0;
boolean WiperQuickState    = 0;

boolean LightOffState      = 0;
boolean LightAutoState     = 0;
boolean LightLowState      = 0;
boolean LightHighState     = 0;
boolean LightParkState     = 0;
boolean LightFogState      = 0;

int LightState = 62;
int LightLastState = 62;
int WiperState = 31;
int WiperLastState = 31;

boolean RightLightState       = 0;
boolean RightLightLastState   = 0;
int     RightLightCounter     = 0;
boolean LeftLightState        = 0;
boolean LeftLightLastState    = 0;
int     LeftLightCounter      = 0;


boolean CruisePaddleState          = 0;
boolean CruisePaddleLastState      = 0;
int     CruisePaddleCounter        = 0;
boolean CruiseUpState              = 0;
boolean CruiseUpLastState          = 0;
boolean CruiseDownState            = 0;
boolean CruiseDownLastState        = 0;

boolean FogPaddleState       = 0;
boolean FogPaddleLastState   = 0;
boolean FogPaddleCounter     = 0;

boolean HornState            = 0;
boolean HornLastState        = 0;


boolean VolumeDownState       = 0;
boolean VolumeDownLastState   = 0;
boolean VolumeUpState         = 0;
boolean VolumeUpLastState     = 0;
boolean ChannelUpState        = 0;
boolean ChannelUpLastState    = 0;
boolean ChannelDownState      = 0;
boolean ChannelDownLastState = 0;

boolean ExtraB1State        = 0;
boolean ExtraB1LastState    = 0;

void setup() {

  pinMode(HazardBPin, INPUT);
  pinMode(WiperOffPin, INPUT_PULLUP);
  pinMode(WiperAutoPin, INPUT_PULLUP);
  pinMode(WiperSlowPin, INPUT_PULLUP);
  pinMode(WiperNormalPin, INPUT_PULLUP);
  pinMode(WiperQuickPin, INPUT_PULLUP);
  pinMode(LightOffPin, INPUT);
  pinMode(LightAutoPin, INPUT);
  pinMode(LightLowPin, INPUT);
  pinMode(LightHighPin, INPUT);
  pinMode(LightParkPin, INPUT);
  pinMode(LightFogPin, INPUT);

  Serial.begin(115200);
  SPI.begin();
  mcp2515.reset();
  mcp2515.setBitrate(CAN_125KBPS);
  mcp2515.setNormalMode();

}

```

```
void loop() {
  //Read buttons
  HazardBState = digitalRead(HazardBPin);
  WiperOffState = digitalRead(WiperOffPin);
  WiperAutoState = digitalRead(WiperAutoPin);
  WiperSlowState = digitalRead(WiperSlowPin);
  WiperNormalState = digitalRead(WiperNormalPin);
  WiperQuickState = digitalRead(WiperQuickPin);
  LightOffState = digitalRead(LightOffPin);
  LightAutoState = digitalRead(LightAutoPin);
  LightLowState = digitalRead(LightLowPin);
  LightHighState = digitalRead(LightHighPin);
  LightParkState = digitalRead(LightParkPin);
  LightFogState = digitalRead(LightFogPin);
  CruisePaddleState = digitalRead(CruisePaddlePin);
  FogPaddleState = digitalRead(FogPaddlePin);
  HornState = digitalRead(HornPin);
  RightLightState = digitalRead(RightLightPin);
  LeftLightState = digitalRead(LeftLightPin);
  CruiseUpState = digitalRead(CruiseUpPin);
  CruiseDownState = digitalRead(CruiseDownPin);
  VolumeDownState = digitalRead(VolumeDownPin);
  VolumeUpState = digitalRead(VolumeUpPin);
  ChannelUpState = digitalRead(ChannelUpPin);
  ChannelDownState = digitalRead(ChannelDownPin);
  ExtraB1State = digitalRead(ExtraB1Pin);

  delay(100);

  //bitshift for wiper
  bitWrite(WiperState, 0, WiperOffState);
  bitWrite(WiperState, 1, WiperAutoState);
  bitWrite(WiperState, 2, WiperSlowState);
  bitWrite(WiperState, 3, WiperNormalState);
  bitWrite(WiperState, 4, WiperQuickState);

  //ReadRotaryWiper
  if (WiperState != WiperLastState) {
    if (WiperState ==  0x1E) {
      //WiperOFF
CreateData(1, 0);
     mcp2515.sendMessage(&canMsg);
    } else if (WiperState == 0x1D) {
      //WiperAuto
```

```
CreateData(1, 1);
     mcp2515.sendMessage(&canMsg);
    } else if (WiperState == 0x1B) {
      //WipeSlow
CreateData(1, 2);
     mcp2515.sendMessage(&canMsg);
    } else if (WiperState == 0x17) {
      //WipeNormal
      CreateData(1, 3);
     mcp2515.sendMessage(&canMsg);
    } else if (WiperState == 0x0F) {
      //WipeQuick
      CreateData(1, 4);
      mcp2515.sendMessage(&canMsg);
    }
  }

  //bitshift for Light
  bitWrite(LightState, 0, LightOffState);
  bitWrite(LightState, 1, LightAutoState);
  bitWrite(LightState, 2, LightLowState);
  bitWrite(LightState, 3, LightHighState);
  bitWrite(LightState, 4, LightParkState);
  bitWrite(LightState, 5, LightFogState);


  //ReadRotaryLight
  if (LightState != LightLastState) {
    if (LightState == 0x3E) {
      //LightOFF
      CreateData(2, 0);
      mcp2515.sendMessage(&canMsg);
    } else if (LightState == 0x3D) {
      //LightAuto
      CreateData(2, 1);
      mcp2515.sendMessage(&canMsg);
    } else if (LightState == 0x3B) {
     //LightLow
      CreateData(2, 2);
      mcp2515.sendMessage(&canMsg);
    } else if (LightState == 0x37) {
      //LightHigh
      CreateData(2, 3);
      mcp2515.sendMessage(&canMsg);
    } else if (LightState == 0x2F) {
      //LightNormal
      CreateData(2, 4);
      mcp2515.sendMessage(&canMsg);
    } else if (LightState == 0x1F) {
      //LightQuick
      CreateData(2, 5);
      mcp2515.sendMessage(&canMsg);
    } else {
      delay(10);
    }
  }

  if (RightLightState == HIGH and RightLightLastState == 0) {
    if (RightLightCounter == 0) {
      RightLightCounter = 1;
      //RightLight ON
      CreateData(3, 1);
      mcp2515.sendMessage(&canMsg);
      if (LeftLightCounter != 0) {
        //LeftLight OFF
        LeftLightCounter = 0;
        CreateData(4, 0);
        mcp2515.sendMessage(&canMsg);
      }
    } else {
      RightLightCounter = 0;
      //RightLight OFF
      RightLightCounter = 0;
      CreateData(3, 0);
      mcp2515.sendMessage(&canMsg);
    }
    RightLightState = 1;
  }
  if ( RightLightState == 0 and RightLightLastState == 1) {
    RightLightLastState = 0;
  }


  if (LeftLightState == HIGH and LeftLightLastState == 0) {
    if (LeftLightCounter == 0) {
      LeftLightCounter = 1;
      //LeftLight ON
      CreateData(4, 1);
      mcp2515.sendMessage(&canMsg);
      if (RightLightCounter != 0) {
        //RightLight OFF
        RightLightCounter = 0;
        CreateData(3, 0);
        mcp2515.sendMessage(&canMsg);
      }
    } else {
      LeftLightCounter = 0;
      //LeftLight OFF
      LeftLightCounter = 0;
      CreateData(4, 0);
      mcp2515.sendMessage(&canMsg);
    }
    LeftLightState = 1;
  }
  if ( LeftLightState == 0 and LeftLightLastState == 1) {
    LeftLightLastState = 0;
  }


  if (FogPaddleState != FogPaddleLastState) {
    if (FogPaddleState == HIGH) {
      //FogPaddle ON
      CreateData(5, 1);
      mcp2515.sendMessage(&canMsg);
    } else {
      //FogPaddle OFF
      CreateData(5, 0);
      mcp2515.sendMessage(&canMsg);
    }
  }

  //HazardButton
  if (HazardBState != HazardBLastState) {
    if (HazardBState == HIGH) {
      //Hazard Button ON
      CreateData(6, 0);
      mcp2515.sendMessage(&canMsg);
    } else {
      //Hazard Button OFF
      CreateData(6, 1);
      mcp2515.sendMessage(&canMsg);
    }
  }

  if (HornState != HornLastState) {
    if (HornState == HIGH) {
      //Horn ON
      CreateData(7, 1);
      mcp2515.sendMessage(&canMsg);
    } else {
      //Horn OFF
      CreateData(7, 0);
      mcp2515.sendMessage(&canMsg);
    }
  }


  if (CruisePaddleState == HIGH and CruisePaddleLastState == 0) {
    if (CruisePaddleCounter == 0) {
      CruisePaddleCounter = 1;
      //CruisePaddlePaddle ON
      CreateData(8, 1);
      mcp2515.sendMessage(&canMsg);
    } else {
      CruisePaddleCounter = 0;
      //CruisePaddlePaddle OFF
      CreateData(8, 0);
      mcp2515.sendMessage(&canMsg);
    }
    CruisePaddleLastState = 1;
  }
  if ( CruisePaddleState == 0 and CruisePaddleLastState == 1) {
    CruisePaddleLastState = 0;
  }


  if (CruiseUpState != CruiseUpLastState and CruisePaddleCounter != 0) {
    if (CruiseUpState == HIGH) {
      //Cruise Up
      CreateData(9, 1);
      mcp2515.sendMessage(&canMsg);
    }
  } else if  (CruiseDownState != CruiseDownLastState and
 ↪  CruisePaddleCounter != 0) {
    if (CruiseDownState == HIGH) {
      //Cruise Down
      CreateData(9, 0);
      mcp2515.sendMessage(&canMsg);
    }
  }




  if (VolumeDownState != VolumeDownLastState) {
    if (VolumeDownState == HIGH) {
      //Volumne Down
      CreateData(10, 0);
      mcp2515.sendMessage(&canMsg);
    }
  }
  if (VolumeUpState != VolumeUpLastState) {
    if (VolumeUpState == HIGH) {
      //Volumne Up
      CreateData(10, 1);
      mcp2515.sendMessage(&canMsg);
    }
  }



  if (ChannelDownState != ChannelDownLastState) {
    if (ChannelDownState == HIGH) {
      //Channel Down
      CreateData(11, 0);
      mcp2515.sendMessage(&canMsg);
    }
  }

  if (ChannelUpState != ChannelUpLastState) {
    if (ChannelUpState == HIGH) {
      //Channel Up
      CreateData(11, 1);
      mcp2515.sendMessage(&canMsg);
    }
  }


    if (ExtraB1State != ExtraB1LastState) {
      if (ExtraB1State == HIGH) {
        //ExtraB1 ON
        CreateData(12, 1);
        mcp2515.sendMessage(&canMsg);
      } else {
        //ExtraB1 OFF
        CreateData(12, 0);
        mcp2515.sendMessage(&canMsg);
      }


  HazardBLastState = HazardBState;
  WiperLastState = WiperState;
  LightLastState = LightState;
  LeftLightLastState = LeftLightState;
  RightLightLastState = RightLightState;
  FogPaddleLastState = FogPaddleState;
  HornLastState = HornState;
  VolumeUpLastState = VolumeUpState;
  VolumeDownLastState = VolumeDownState;
  ChannelDownLastState = ChannelDownState;
  ChannelUpLastState = ChannelUpState;
  ExtraB1LastState = ExtraB1State;
  CruisePaddleLastState = CruisePaddleState;
  CruiseUpLastState = CruiseUpState;
  CruiseDownLastState = CruiseDownState;
}


  //sending the data to the CAN
  //CAN ints, for now i have just arbitarily numbered each button/dial
  ↪  and their states, since we have no overall set convention
  void CreateData(int ID, int STATE) {
    canMsg.can_id  = ID;          //The ID - for each component, the
    ↪  lower - the more cirtical/important
    canMsg.can_dlc = 8;          // Data Length Code
    canMsg.data[0] = STATE;      //State
    canMsg.data[1] = 0x00;        //Set rest to 0
    canMsg.data[2] = 0x00;
    canMsg.data[3] = 0x00;
  }
```

A.2 Display Script

```
#include <SPI.h>
#include <mcp2515.h>
#include <HT1621.h>

HT1621 currentSpeedLCD;
HT1621 optimalSpeedLCD;
bool backlightStatus = false;
long currentSpeed;
long optimalSpeed;

struct can_frame canMsg;
MCP2515 mcp2515(53);

void setup() {
  Serial.begin(115200);

  currentSpeedLCD.begin(13, 12, 11, 10); // (cs, wr, Data, backlight)
  optimalSpeedLCD.begin(9, 8, 7, 6); // (cs, wr, Data, backlight)

  currentSpeedLCD.clear();
  optimalSpeedLCD.clear();

  mcp2515.reset();
  mcp2515.setBitrate(CAN_125KBPS);
  mcp2515.setNormalMode();

  Serial.println("------- CAN Read ----------");
  Serial.println("ID DLC DATA");

  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) {
    Serial.print(canMsg.can_id, HEX); // print ID
    Serial.print(" ");
    Serial.print(canMsg.can_dlc, HEX); // print DLC
    Serial.print(" ");

    for (int i = 0; i<canMsg.can_dlc; i++) { // print the data
      Serial.print(canMsg.data[i],HEX);
      Serial.print(" ");
    }

    Serial.println();
  }
}

void loop() {
  if (canMsg.can_id == 0x0F1){
    currentSpeed = canMsg.data[1];
  }

  if (canMsg.can_id == 0x0F2){
    optimalSpeed = canMsg.data[1];
  }

  updateDisplays(currentSpeed, optimalSpeed);

  delay(10);
}

void updateDisplays(long currSpeed, long optimSpeed) {
    currentSpeedLCD.print(currSpeed);
    optimalSpeedLCD.print(optimSpeed);
}

void toggleBacklight() {
  if(backlightStatus) {
    currentSpeedLCD.noBacklight();
    optimalSpeedLCD.noBacklight();
  }
  else {
    currentSpeedLCD.backlight();
    optimalSpeedLCD.backlight();
  }
  backlightStatus != backlightStatus;
}
```

Bibliography

[Ada] Adafruit. *Musicmaker shield.* URL: https://www.adafruit.com/product/1788.

[Ard] Arduino. *arduinomega info.* URL: https://store.arduino.cc/arduino-mega-2560-rev3.

[aut68] TRW automotive. *Rain sensor.* 1968. URL: https://cdn.hswstatic.com/pdf/rain-sensor.pdf.

[Cha20] Bridgestone World Solar Challenge. *Regulations Bridgestone World Solar Challenge.* 2020. URL: https://worldsolarchallenge.org/the-challenge/regulations.

[Com21] Hobby Components. *HT1621 6 Digit 7 Segment LCD Module.* 2021. URL: https://hobbycomponents.com/newproducts/1001-ht1621-6-digit-7-segment-lcd-module.

[CS20] Lukas Fredrik Cronje and Louise Sternholdt-Sørensen. *Implementation of CAN bus in a concept solar car.* 2020. URL: https://drive.google.com/file/d/1uEm09sTHEzy_%20yrX0qxGZzbVgrMhvMZm4/view?usp=sharing.

[Fer17] Daniel Fernandes. *Wiring the 6 Digit 7 Segment 2.4 Inch HT1621 LCD Display Module on Microcontroller.* 2017. URL: https://www.14core.com/wiring-the-6-digit-7-segment-2-4-inch-ht1621-lcd-display-module-on-microcontroller/.

[Gee18] Geekboots. *LCD vs TFT vs LED.* 2018. URL: https://www.geekboots.com/story/lcd-vs-tft-vs-led.

[Hol19] Holtek. *HT1621 datasheet.* 2019. URL: https://www.holtek.com/documents/10179/116711/HT1621v321.pdf.

[Jen21] Henrik Kjær Jensen. *Display driver library for 7, 14 16 segment LED and LCD displays on Arduino.* 2021. URL: https://lygte-info.dk/project/DisplayDriver%20UK.html#HT1621_6_digit_LCD_display.

[KP13] Dagmar Kern and Bastian Pfleging. “Supporting interaction through haptic feedback in automotive user interfaces.” eng. In: *Interactions* 20.2 (2013), pages 16–21. ISSN: 15583449, 10725520. DOI: 10.1145/2427076.2427081.

[LR06] JAMES ERIC LLOYD Minister for Local Government Territories and Roads. *Vehicle Standard (Australian Design Rule 46/00-Headlamps).* 2006. URL: https://www.legislation.gov.au/Details/F2006L02294.

[Lun+10] Wei Lun et al. “Review of Researches in Controller Area Networks Evolution and Applications.” In: *Proceedings of the Asia-Pacific Advanced Network* 30 (August 2010). DOI: 10.7125/APAN.30.3.

[mag19] The News Wheel (car magazine). *All Your Car’s Headlight Symbols Are Finally Explained*. 2019. URL: https://thenewswheel.com/your-cars-headlight-symbols-are-finally-explained/.

[Nic21] Karim Nice/HowStuffWorks. *wiper controls - How Windshield Wipers Work*. 2021. URL: https://auto.howstuffworks.com/wiper3.htm.

[RBB12] Annie Rydström, Robert Broström, and Peter Bengtsson. “A comparison of two contemporary types of in-car multifunctional interfaces.” eng. In: *Applied Ergonomics* 43.3 (2012), pages 507–514. ISSN: 18729126, 00036870. DOI: 10.1016/j.apergo.2011.08.004.

[Sil17] Robert Silva. *The Truth About So-Called LED TVs*. 2017. URL: https://www.lifewire.com/truth-about-so-called-led-televisions-1847935.

[Spy10] The Car Spy. *2006 Range Rover Sport HST*. 2010. URL: https://commons.wikimedia.org/wiki/File:2006_Range_Rover_Sport_HST_-_Flickr_-_The_Car_Spy_(14).jpg.

[TD21] Time and Date. *Darwin, Northern Territory, Australia — Sunrise, Sunset, and Daylength, October 2023*. 2021. URL: https://www.timeanddate.com/sun/australia/darwin?month=10&year=2023.

[Teo] Teoriklar. *Signals, symbols and lighting requirements*. URL: https://www.teoriklar.eu/552/signale-tegn-og-lygtefoering.

[Uln18] Uln2003. *7 Segment Display with Labeled Segments*. 2018. URL: https://commons.wikimedia.org/wiki/File:7_Segment_Display_with_Labeled_Segments.svg.

[USA14] Micro Tips USA. *DIFFERENCE BETWEEN TFT AND LCD*. 2014. URL: https://www.microtipsusa.com/blog/difference-tft-lcd/.

[va21] valerionew and anxzhu. *HT1621 Arduino Library*. 2021. URL: https://www.arduino.cc/reference/en/libraries/ht1621/.

[VW06] Jan B.F. Van Erp and Peter Werkhoven. “Validation of principles for tactile navigation displays.” eng. In: *Proceedings of the Human Factors and Ergonomics Society* (2006), pages 1687–1691. ISSN: 21695067, 10711813.

[www] www.sparkfun.com. *hazardbutton datasheet*. URL: http://cdn.sparkfun.com/datasheets/Components/Switches/com-11966.pdf.

[YLC13] Jeong Il Yu, Youngjae Lim, and David Chung. “A study regarding comparisons of driver performance according to touch display locations (X, Y, Z axes).” eng. In: *Sae Technical Papers* 1 (2013).