**Cruise Control HUD: Team 1**



|  |  |  |
| --- | --- | --- |
| *Team Members (left-to-right on picture, above)* | *Class No.* | *Lab Div* |
| Patrick May | 5059-M | 7 |
| William Pierce | 6079-P | 3 |
| Tyler Carter | 6876-C | 6 |
|  |  |  |

|  |  |
| --- | --- |
| *Report/Functionality Grading Criteria* | *Points* |
| Originality, creativity, level of project difficulty | 20 |
| Technical content, succinctness of report | 10 |
| Writing style, professionalism, references/citations | 10 |
| Project functionality demonstration | 20 |
| Overall quality/integration of finished product | 10 |
| Effective utilization of microcontroller resources | 10 |
| Significance of individual contributions\* | 20 |
| *Bonus Credit Opportunities* | *Bonus* |
| Early completion | 0.5% |
| PCB for interface logic | 2% |
| Poster (required for Design Showcase participation) | 1% |
| Demo video (required for Design Showcase participation) | 1% |
| Design Showcase participation (attendance required)\* | 1% |

##### \**scores assigned to individual team members may vary*

|  |  |
| --- | --- |
| *Grading Rubric for all Criteria (Including Bonus)* | *Multiplier* |
| *Excellent* – among the very best projects/reports completed this semester | 1.0 - 1.1 |
| *Good* – all requirements were amply satisfied | 0.8 - 0.9 |
| *Average* – some areas for improvement, but all basic requirements were satisfied | 0.6 - 0.7 |
| *Below average* – some basic requirements were not satisfied | 0.4 - 0.5 |
| *Poor* – very few of the project requirements were satisfied | 0.1 - 0.3 |

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| 1.0 Introduction | 1 |
| 2.0 Interface Design | 2 |
| 3.0 Peripheral Utilization | 3 |
| 4.0 Software Narrative | 4 |
| 5.0 Packaging Design | 5 |
| 6.0 Summary and Conclusions | 6 |
| 7.0 References | 7 |
| Appendix A: Individual Contributions and Activity Logs | 8 |
| Appendix B: Interface Schematic and PCB Layout Design | 17 |
| Appendix C: Software Flowcharts | 19 |
| Appendix D: Packaging Design | 21 |

Source and additional documentation may be found at

**Common Acronyms:**

Engine Control Unit (ECU)

On Board Diagnostics (OBD)

Light Detection and Ranging (LIDAR)

Computer Aided Design (CAD)

Best Friend From Another Major (BFFAM)

3-Dimensional (3D)

<https://github.com/wgpierce/Cruise-Control-HUD>

Video explanation and demonstration may be found at

<https://www.youtube.com/watch?v=xGuQt2sF4p8>

1. **Introduction**

The purpose of this project was to create an adaptive cruise control HUD (heads-up display) to be used in an automobile that would display its current speed as well as distance relative to whatever object happens to be in front it. For this project, the HC9S12C microcontroller had two main functions: to interface with the automobile’s ECU (engine control unit) to get its speedometer reading, and to communicate with a LIDAR sensor to determine the distance between the vehicle the device was mounted on (referred to as the “ego vehicle”) and whatever happens to be directly in front of it.

Interfacing with a car’s ECU allowed the connected microcontroller to access a wealth of diagnostic information. Though it certainly would have been possible to try and measure the car’s velocity through other means, interfacing with the ECU also provides power directly from the car. To retrieve information, such as current speed, a request must be sent and then a response received via the chip’s onboard SCI peripheral.

The LIDAR sensor measured distance from the ego vehicle to any object in line with the sensor. These measurements were then used to approximate relative velocity by taking the difference between each reading. The velocity of the ego car could then be increased or decreased so that the relative velocity was zero. Then, the cruise control could be set indefinitely. To assist with this, three LEDs were included to indicate if the user should speed up, slow down, or maintain velocity to match the car in front of them.

Patrick oversaw designing the PCB in Eagle. He wrote the SCI drivers,

and designed the additional hardware required for the serial communication. He also

had a part in creating the 3-D printed case and lid for the project.

Will wrote a large chunk of the software, including the main loop and the preliminary version of the LIDAR driver. He also designed and built the LIDAR enclosure and created the poster board for the SPARK challenge, as well as creating the YouTube video that was put online.

Tyler programmed the three PWM LEDs and helped write the final version of the LIDAR driver. He and Pat got the voltage regulator working, and he also assisted in creating the case as well as largely designing the lid and building a prototype out of paper.

1. **Interface Design**

The user interface consisted of two four-by-seven segment displays and three “direction arrow” LEDs. A LIDAR was used to determine the distance of the current car from the car in front of it. An OBD-to-UART board was used to provide power to the project as well as determine the car’s speed. An external switch was used to connect or disconnect power to the external boards. Two eight-bit shift registers were daisy chained together to form a 16-bit shift register and connected to the LED displays.

The 16-bit shift register controlled the individual segments of the display. The 4x7 segment displays were multiplexed, so the shift registers could only specify a single digit at a time. By changing whether the common cathode for each digit was high or low, which digit was currently being displayed could be changed. Thus, if the shift register was changed and the corresponding common cathode was held high, a single digit of the four could be illuminated. By illuminating multiple segments in rapid succession, all four digits would look like they were illuminated at the same time due to persistence of vision [5].

The LIDAR had both an I2C and a PWM interface [2]. As the 9S12C does not have an I2C interface, the PWM interface was used. The PWM interface required a mode control switch and an input pin. Both were connected to the mode control line; however, a resistor was connected to the mode control line to prevent bus contention. In order to request a reading from the sensor, the mode control line was driven low. The LIDAR would then drive the line high proportional to the distance measured [2].

The Sparkfun OBD-to-UART board provided both power for the project as well as the vehicle’s speed. The OBD board used 0-5v logic levels, so a MAX232 IC mounted on an external board facilitated communication between the OBD board and the 9S12C. A linear regulator connected to the car battery voltage breakout on the OBD-to-UART provides 5V power to the board [1].

The OBD board received and sent data as a series of ASCII characters. Commands sent to the microcontroller on the OBD board are prefixed by “at”. The two relevant commands are “atz”, which initializes the OBD communications interface, and “atsp0”, which automatically determined the OBD protocol standards the car uses.

Commands not prefixed with an “at” were interpreted to be OBD commands. Any value that was not a string of ASCII hexadecimal (e.g. “0”-“9”, “A”-“F” or “a”-“f”) characters was rejected by the board. By default, the OBD-to-UART board echoed characters sent to it. When it was prepared for a new command, it would send a greater than sign (“>”) to the microcontroller. Commands sent to the board were delimited by a carriage return character.

The OBD protocol accepts two-byte commands. The first byte was the instruction group. Group “01” contained general vehicle information, and was the only group of commands used by this project. The second byte was the command instruction. The microcontroller only needed the speed requested, which was command “0D” in the “01” subgroup. The car ECU responded with the byte “41 0D” to acknowledge that a speed request was sent, then two postbytes containing the speed in hexadecimal. Since the OBD-to-UART board changed these bytes into their string equivalent, the response received was in the form “41 0D XX” where XX was the speed in hexadecimal.

# **Microcontroller Resource Utilization**

The microcontroller utilized the ATD, SCI, SPI, TIM, and PWM modules. The TIM, PWM, and SPI modules were used for displaying information to the user, and the TIM, ATD and SCI were used to gather information from the sensors.

The SPI module was used to control a shift register connected to a multiplexed 4x7 segment display. This was implemented by using two eight-bit shift registers which were daisy chained to create a single 16-bit shift register. [4] The SPI clock was connected to the input clock of the shift registers.

As mentioned in the paragraph prior, the 4x7 segment displays were multiplexed. Multiplexing in the context of a multi-digit display means that all of the identical anode segments for the numbers were wired together (e.g. the “A” anode for the first digit was connected to the “A” anode of the other three digits). [5] The common cathode for each digit was connected to separate GPIO pins. In order for all four digits to be visible the controller needed to update the display within strict time requirements. It was experimentally determined that any update rate less than 50 Hz would cause a visible “scanning” effect. The timer module was employed in an interrupt-driven mode using TIM7 output compare to ensure that the refresh rate for each of the seven-segment displays remained consistent at approximately 50 Hz.

The PWM module was used to modulate the brightness of the speed up/slow down LEDs. The PWM duty cycle was then changed proportional to the difference between the current LIDAR reading and the previous reading. The PWM clock was set to operate at the bus clock speed and the period was set to 255 cycles.

The SCI module operated in an interrupt driven mode with circular buffers for both input and output. Any time a transmission was required, the data to transmit was placed in the output buffer and output ready interrupts were enabled. When the output buffer was empty, the transmit ready interrupt was disabled. The data received interrupt was always enabled, and received data took priority over transmitted data. This approach to serial communication allowed for more lenient timing compared to program-driven operation, as the main loop did not need to run at a certain speed to ensure that data is received. An interrupt driven module could also avoid being preempted by a lower priority interrupt. This ensured that every bit received was placed into memory. During initialization, the SCI sent out a string to wake up the OBD-to-UART board, and then a second string was sent to enable automatic detection of the OBD protocol. [1]

The ATD module was used in a similar manner to a pulse accumulator. The LIDAR interfaced with the microcontroller using a PWM-based protocol. The LIDAR would then set the line high for 10 microseconds for every centimeter of distance detected. [2] As it was determined that the aperture was too small to average the PWM reading without modifying the bus clock (and slowing down other peripherals) [6], an alternative method was used. The ATD was set to behave in an interrupt driven fashion every time a measurement was requested. The aperture was set to generate an interrupt every 10 microseconds. For every time the ATD reported a 3.3 V signal (the logic level of the sensor), a counter was accumulated. When the line was driven low again, the counter stopped and the mode control line was set high and ATD interrupts disabled. Since the mode control needed to be set high immediately after the reading finished to avoid picking up multiple measurements, the ATD was used rather than the pulse accumulator.

1. **Software Narrative**

This program utilized both interrupt driven and a continuous polling main loop. The software consisted of 3 main sections: the main loop which controls requesting speed via the OBD and updating status LEDs, the timer interrupt which controls LED update refresh rate and LIDAR measurements.

First, a driver was written to control the shift registers and 7-segments via SPI. This driver was then utilized to create a function which would shift out numbers one at a time. Because of the hardware configuration, two bytes were shifted out at a time. These functions were called within the timer 7 interrupt to update the display at 50Hz. Each time the function was called, a global variable for which digits should be displayed was incremented modulo 4. The output was held until the next time the function was called within this interrupt. This method performed better than its first iteration, which involved waiting 9000 CPU cycles for each 7-segment. The second iteration allowed for other tasks to be performed during that waiting period.

The next driver developed controlled communication with the ECU in the car via the OBD port. This included an external board as described in the interface design section. This driver utilized two buffers which handle incoming and outgoing communications. The contents of the buffers are shifted out and in using SCI interrupts with circular buffer logic. This SCI driver was utilized to send and receive commands via the OBD board. The OBD board was initialized by sending the strings “atz\r” to wake the board and “atsp0\r” to tell the board which OBD line to listen on [1]. The OBD was then continuously polled to ask for car’s speed with the request "010D\r", and then the program waits until the response is received. Once the whole message is received, it is parsed.

The driver to control the LIDAR measurement was then written. It was entirely handled within the ATD interrupt service routine. Every 13 TIM interrupts, an ATD measurement was initiated, which consisted of turning on ATD interrupts and initiating a conversion. Initially, it was assumed that the LIDAR had a constant PWM period. This assumption was later proved false [2]. In response, the code was changed to accumulate the number of times the LIDAR held the pin high. After measurement is done, distance is calculated using known constants, and the ATD interrupt is turned off.

Finally, the logic for the status LEDs was written. The up, down, and dash arrow logic was evaluated in the main loop. This logic includes calculating the relative speed between the ego car and the target car. If the absolute difference was less than 1km / hr, then the dash arrow was displayed at a constant brightness. If the ego car was traveling between 1 and 25 km faster or slower than the target car, the up or down arrow was lit with a brightness proportional to the relative speed of the target car. If the velocity measured was greater than 25 kph, then maximum brightness was used.

1. **Packaging Design**

All of the electronic components used in the design, except for the LIDAR sensor, were contained in a single 3-D printed plastic enclosure that was 5.25 by 5.5 by 1.5 inches in size. All 3-D printed components were printed by a PrintrBot Simple Metal with black polylactic acid filament. The PCBs, including the main one with the HC9S12C on it, the Sparkfun OBD UART board, and the MAX232 PCB, were held in place by standoffs with a raised core that was small enough to fit through the corner holes in each PCB. A dot of hot glue on the core secured the PCB to the bottom of the standoff. Hot glue was also used elsewhere inside the enclosure to provide strain relief for wires and to prevent any loose components from shifting. There was a hole in the front of the plastic enclosure for the serial port of the Sparkfun OBD UART board to stick out of. There was another hole near the OBD UART for the three wires going to the LIDAR sensor. There were two strips of hot glue on the bottom to prevent sliding.

The enclosure also had a 3-D printed plastic cover. This cover was intended to protect all the electronics in the enclosure while allowing for wide viewing angles on the LEDs. The three PWM LEDs are walled off to separate them from each other, and covered on top with a symbol cut out so that a specific shape illuminates when the LEDs are turned on. The three symbols are an up arrow, a down arrow, and a dash. A single layer of a tissue was placed directly under each of these symbols to diffuse the light from the LEDs. The cover also has a slot for the serial port on the UART board and an opening for debugging the microcontroller via USBDM cable.

To connect this enclosure and the LIDAR sensor, there were three wires that were about five feet in length each. These wires are passed through a nylon sheath that was held in place using black electrical tape. The wires themselves are braided to keep them close to each other throughout the entire length of this cable. The three wires leading to the LIDAR sensor were able to be connected and disconnected from the wires coming out of the main enclosure for convenience when transporting.

The LIDAR enclosure was a smaller box that was also 3-D printed using the same material as the main enclosure. The LIDAR enclosure had indents on the bottom to hold the neodymium magnets used to mount the enclosure to the car. The back wall includes pegs to fit into the LIDAR’s mount holes [3], as well as an opening to put wires through. This opening was placed so that rain cannot infiltrate the enclosure easily. The middle of the enclosure included a support wall to hold the LIDAR up, should the pegs break. The enclosure was designed to be aerodynamic so that the LIDAR would not be blown off the car or misaligned. To assemble the enclosure, the magnets were placed in the bottom, the LIDAR was slotted into the pegs, and the lid was placed on and taped down cleanly using electrical tape.

1. **Summary and Conclusions**

The hardware aspect of the project was a learning experience for everyone, especially Will and Tyler who had less exposure to hardware design. Using Eagle to design a PCB was something that only Pat had done previously, and he learned some additional tricks, such as putting an image on a PCB. We all had some degree of experience soldering prior to this project, but not with parts as close together as the ICs that we used (namely the contacts on shift registers), and not with using soldering paste. Using soldering paste was actually very different from using a soldering iron and solder, though it does make things remarkably easier. We all ended up learning the difference between RS232 and TTL voltage levels, since we had to end up including a separate perfboard after forgetting about that detail in the initial design. Similarly, we had to learn about using voltage regulators to step down the 12V OBD Vcc to the 5V power we wanted for out HC9S12C. We all got experience with 3-D printing.

Software-wise, we gained some experience using the on-chip peripherals, particularly the SCI and ATD. There were many iterations on the PWM driver, as the LIDAR outputted a PWM signal of variable frequency, so the pulse accumulator wouldn't work. We then had to use the ATD to simulate a variation of the pulse accumulator. Interfacing the two main peripherals used in this project (OBD and LIDAR) included a fair bit of research, as both involved using new protocols.

To improve the design, we would integrate more things onto the main PCB. We didn't capture everything that we were going to need in our preliminary PCB design, which resulted in a bulkier final product. All of these could be integrated onto a single board to make a more compact product. The UART board could have been integrated onto the main circuit board using the schematic provided by Sparkfun. Another improvement that could be made is that the LIDAR sensor would have to be replaced with a sensor that has substantially more range. The LIDAR used in this project claimed to have an operational range of 40 meters, however in practice it detected up to 25 meters. This is at most two or three car lengths. At such a distance, the HUD does not show anything the driver does not already know.

Overall, the project was a valuable learning tool, and has taught the team much about the design process in the real world.

1. **References**
2. *STN1110 Multiprotocol OBD to UART Interpreter Datasheet*, Rev. B, OBD Solutions, Phoenix, AZ, 2012, http://cdn.sparkfun.com/datasheets/Widgets/stn1110-ds.pdf
3. *Lidar Lite v3 Operation Manual and Technical Specifications*, Garmin Ltd., Olathe, KS, 2016, <http://static.garmin.com/pumac/LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf>
4. *LIDAR-Lite v2 Overview*, PulsedLight, LLC., Bend, Or, 2014, https://cdn.sparkfun.com/datasheets/Sensors/Proximity/lidarlite2DS.pdf
5. *74HC595; 74HCT595*, Rev. 8, NXP Semiconductors, NV, 2016, <http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf>
6. *YSD-439AY2B-35,* China Young Sun LED Technology CO., LTD., <https://www.sparkfun.com/datasheets/Components/LED/7-Segment/YSD-439AY2B-35.pdf>
7. *ATD\_10B8C Block User Guide,* Motorola Inc., <https://engineering.purdue.edu/ece362/Refs/9S12C_Refs/S12ATD10B8CV2.pdf>

**Appendix A:**

**Individual Contributions**

**and**

**Activity Logs**

**Activity Log for:** Patrick May **Role:** Team Leader

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Activity*** | ***Date*** | ***Start Time*** | ***End Time*** | ***Time Spent*** |
| Preliminary PCB design, picked parts for project | 11/2 | 16:30 | 18:30 | 2:00 |
| Revised PCB design | 11/3 | 12:30 | 14:30 | 2:00 |
| Some parts on board obsolete; removed 7-segment decoder, replaced with Sparkfun parts | 11/8 | 20:30 | 24:00 | 3:30 |
| Routed board | 11/10 | 13:30 | 14:30 | 1:00 |
| Sent order for board | 11/11 | 14:30 | 16:30 | 2:00 |
| Worked on debugging SCI, went to lab to find out it was RS232 standard | 11/19 | 13:30 | 17:30 | 4:30 |
| Built MAX232 circuit on breadboard | 11/21 | 12:30 | 13:00 | 30 |
| Wrote preliminary SCI drivers | 11/22 | 16:30 | 18:30 | 2:00 |
| Explored OBD-to-UART interface | 11/22 | 14:30 | 16:30 | 2:00 |
| Finished OBD-to-UART interface | 11/24 | 13:30 | 14:00 | 30 |
| Ordered components for PCB | 11/27 | 14:30 | 15:00 | 30 |
| Soldered MAX232 circuit | 11/25 | 13:00 | 14:00 | 1:00 |
| Soldered main board | 12/1 | 20:30 | 22:30 | 2:00 |
| Systems integration test | 12/2 | 16:00 | 20:00 | 4:00 |
| Worked on CADing external packaging | 12/3 | 14:30 | 21:00 | 7:30 |
| Worked on systems integration, external packaging | 12/4 | 14:30 | 20:30 | 6:00 |
| Fixed LIDAR driver, demonstrated project | 12/5 | 12:30 | 4:00 | 3:30 |
| Made prototype lid, added sheath to cable | 12/6 | 15:30 | 17:30 | 2:30 |
| Designed and printed lid | 12/7 | 16:30 | 18:30 | 2:00 |
| Worked on documentation | 12/8 | 16:30 | 22:00 | 6:30 |
| Spark Challenge | 12/9 | 12:00 | 5:00 | 5:00 |

**Written Summary of Technical Contributions:** Patrick May

I was the first person to gather a group together. In October I got the team together and we brainstormed an idea. Over the course of that weekend I started creating a preliminary design and began choosing all the parts. Since I included parts that are now obsolete, I had re-do a substantial portion of the PCB design. This redesign proved for the better, though, as the finalized PCB was much cleaner and more compact. I submitted the finalized schematic for review by the other team members and in early November ordered the PCB normal shipping from Oshpark. Later I would call in on some favors to secure a heating iron to reflow solder the SMD components to the board. With the other team members, I tested and ensured that every component was connected properly.

While the PCB was shipping, I delegated some tasks to the other members of the team. In addition, I started working on developing the SCI module. Although the SCI code seemed to work with the connections in the lab, the SCI connection to my computer didn’t seem to work properly. After some debugging in the lab, I found that the SCI module only worked with an RS-232. I asked my TA about how to fix this and he suggested a MAX232, which I then purchased. Using the datasheet for help, I wired the MAX232 to convert the microcontroller logic to 0-5v logic. I would later put this on a perfboard to make the conversion portable.

For the SCI drivers, I implemented two circular buffers and developed the code to send and receive data. After taking a look at some OBD communications protocol examples, I created a rudimentary system to request a speed. I would later refine this algorithm to only request speed when the OBD board was prepared to handle an input, as the OBD would halt operation if too many commands were received. With Tyler, we went to his car and logged example call-and-responses from the OBD-to-UART converter.

I worked with the other team members on packaging the product. I used my 3d printer to print designs made by Will and Tyler. After our demonstration, I would design the lid to the box to make the project look “nicer” for the Spark Challenge.

Finally, I assisted other teammates with their drivers. I helped Will figure out the issue with the LIDAR drivers. I also provided suggestions for how to construct the 7-segment display driver.

I also handled any paperwork with the Spark Challenge and registering our team within the 362 team interface.

**Activity Log for:** William Pierce **Role:** Software Lead / LIDAR integration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Activity*** | ***Date*** | ***Start Time*** | ***End Time*** | ***Time Spent*** |
| Created Git repository, added files | 11/11 | 20:00 | 20:10 | 0:10 |
| Wrote Shift register and LED driver | 11/11 | 20:00 | 23:00 | 3:00 |
| Finished multidigit operation and adjusted timing parameters for LED driver, wrote register initializations | 11/12 | 20:00 | 22:00 | 2:00 |
| Changed LED driver to not use delay, wrote rudimentary LIDAR driver | 11/18 | 17:00 | 19:00 | 2:00 |
| (PCD now soldered) Fixed port declarations, tested LEDs, disable MISO in SPI, double buffer LEDs, reverse display order, invert polarities | 11/30 | 13:00 | 16:00 | 3:00 |
| Systems integration test | 12/2 | 18:00 | 20:00 | 2:00 |
| Rewrote LIDAR driver using ATD interrupts | 12/2 | 13:00 | 14:00 | 1:00 |
| Changed LIDAR to only take measurement when we read measurement | 12/3 | 20:00 | 21:00 | 1:00 |
| Drew up LIDAR enclosure in CAD | 12/3 | 22:00 | 24:00 | 2:00 |
| Test project with actual car | 12/4 | 18:30 | 20:30 | 1:30 |
| Helped fix LIDAR driver, demonstrated project | 12/5 | 12:30 | 4:00 | 2:30 |
| After lid printed, go out and record documentation video | 12/7 | 16:30 | 18:30 | 2:00 |
| Edit video together in Adobe Premier | 12/7 | 20:00 |  |  |
| Wrote PPT poster, submitted to be printed | 12/8 | 00:00 | 02:00 | 2:00 |
| Documentation | 12/8 | 15:00 | 22:00 | 7:00 |
| Spark Challenge | 12/9 | 12:30 | 6:00 | 5:00 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Written Summary of Technical Contributions:** William Pierce

Pat recruited me as a team member, and I obliged, as it seemed a worthwhile endeavor. Being a computer scientist, I took on the role of software leader. I also wrote code, and ensured correct initializations, and contributed other meta work to the project.

The role of software leader involved creating and maintaining the Git repository, setting up the CodeWarrior programming environment, and maintaining coding standards within the program. To maintain the Git repository, I also made sure everyone knew what files to include and what files not to include. The coding standard involved using snake\_case, using tabs for indentation, suffix active low variables with \_N, and make #defines UPPERCASE.

The code I wrote included drivers to the shift registers and 4x7-segments, values for shifting out to the shift registers, a preliminary LIDAR driver, and all the correct register initializations. To write the driver to the shift register, I made sure all correct signals to the shift registers were correct per [4]. The LIDAR driver required pulling a line low for the LIDAR to initiate a measurement per [2]. These initializations required calculating timing parameters for TIM interrupts to refresh the 4x7-segment LEDs fast enough that the refresh rate is unnoticeable to the human eye, and ATD interrupts to match the LIDAR PWM output speed.

I also created a LIDAR enclosure using Autodesk Fusion. I created a design that sufficiently protected the LIDAR from the elements, allowed the LIDAR to be placed inside fairly easily, allowed neodymium mounting magnets to be placed inside, and was at least somewhat aerodynamic. To this end, I combined these specifications and dimension measurements and dimensions from [3] into a CAD file and then had Pat 3-D print them on his 3-D printer.

For the extra credit portions, I shot and edited a video of the other two members explaining the project and demonstrating the project. Editing the video took knowledge of Adobe Premier to edit volume levels, splice videos, and add titles and music tracks. The video was uploaded to YouTube.

I also took charge of creating and printing a poster which includes summary information about out project and representative pictures. The poster was printer at the Burke Morgan Center for Entrepreneurship at Purdue.

**Activity Log for:** Tyler Carter **Role:** Documentation Lead

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Activity*** | ***Date*** | ***Start Time*** | ***End Time*** | ***Time Spent*** |
| Explored OBD to UART interface | 11/22 | 2:30 | 4:20 | 2:00 |
| Soldering main board | 12/1 | 20:30 | 22:30 | 2:00 |
| Systems integration test | 12/2 | 16:00 | 20:00 | 4:00 |
| Worked on CAD external packaging, added voltage regulator | 12/3 | 14:30 | 21:00 | 6:30 |
| Systems integration, external packaging, programmed LEDs | 12/4 | 14:30 | 20:30 | 6:00 |
| Fixed LIDAR driver, demo project | 12/5 | 12:30 | 16:00 | 3:30 |
| Made prototype lid, added sheath to cable | 12/6 | 15:30 | 18:00 | 2:30 |
| Designed and printed lid | 12/7 | 16:30 | 18:30 | 2:00 |
| Worked on documentation | 12/8 | 12:00 | 22:00 | 10:00 |
| Spark challenge | 12/9 | 13:00 | 17:00 | 4:00 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Written Summary of Technical Contributions:** Tyler Carter

On November 22, Pat and I determined how to interface with the OBD UART board. We hadn't programmed the HC9S12 yet, so we used a computer terminal. After several failed attempts, the two of us determined that a hex code had to be sent as two or four ASCII alphanumeric characters followed by a carriage return in order to receive a response. This laid the groundwork for the SCI routines written in our code. After Thanksgiving, on the 1st, we as a group soldered the necessary components to the main board. After that, on December 2, we wanted to perform an integration test, but we had to put it off because we didn't have a voltage regulator. Instead, I spent time testing the LIDAR software. The following day, we got a voltage regulator from RadoShack, which I eventually got to work. It was more difficult than it should have been because we ended up with a variable voltage regulator rather than a standard five volt one. I also designed the 3-D printed main enclosure, though we didn't make the lid yet. On Sunday, I programmed the LEDs to indicate whether to speed up or slow down. Since I used the PWM outputs, they have variable brightness depending on relative velocity. We finally got to test out our entire setup, which worked, though I suspected that the LIDAR was not quite working, as it only seemed to work within two meters. On Monday, we determined that the LIDAR driver was, in fact, non-functional, and so I went back to the documentation and determined that we had initially misinterpreted the LIDAR PWM output. Rather than giving a recurring wave with a set duty cycle, the LIDAR only really gave one period, which varied based on how long the signal was high. As such, we reprogrammed the LIDAR to wait until the signal was high, count how many microseconds it was high for, and then convert that to distance. On December 6, in anticipation of the SPARK challenge, we decided that we should make our project more visually appealing. We got a sheath for the cables connecting the LIDAR to the main board, and I designed a prototype cover for our main enclosure out of paper. The next day, we created the lid in CAD and printed it. Finally, on Thursday the 8th, I worked on documentation for far longer than any sane human being would want to.

**Appendix B:**

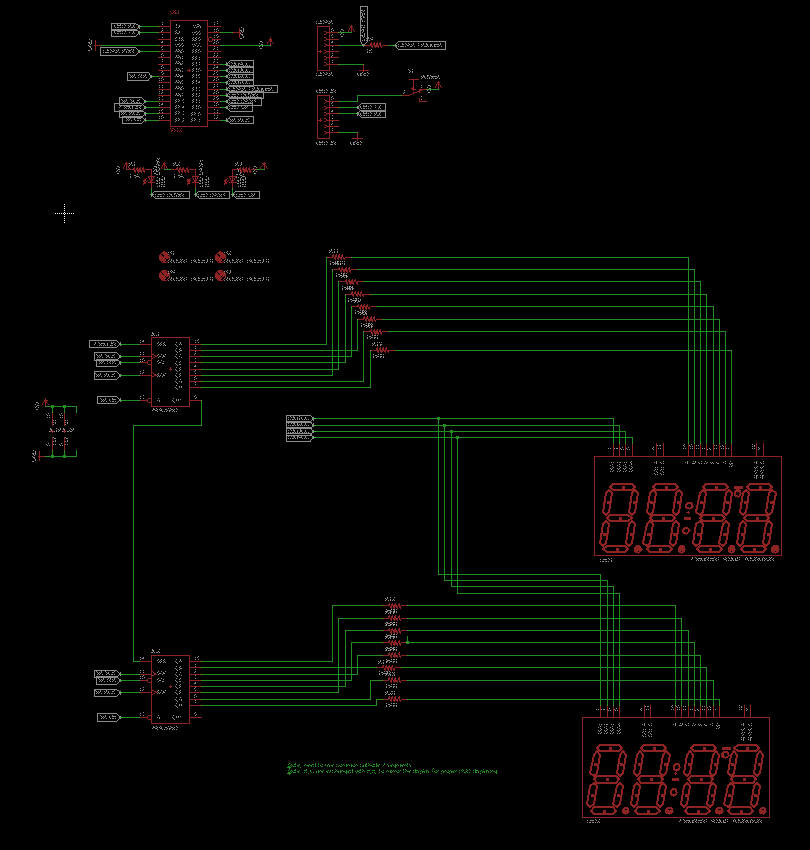
**Interface Schematic**

**and**

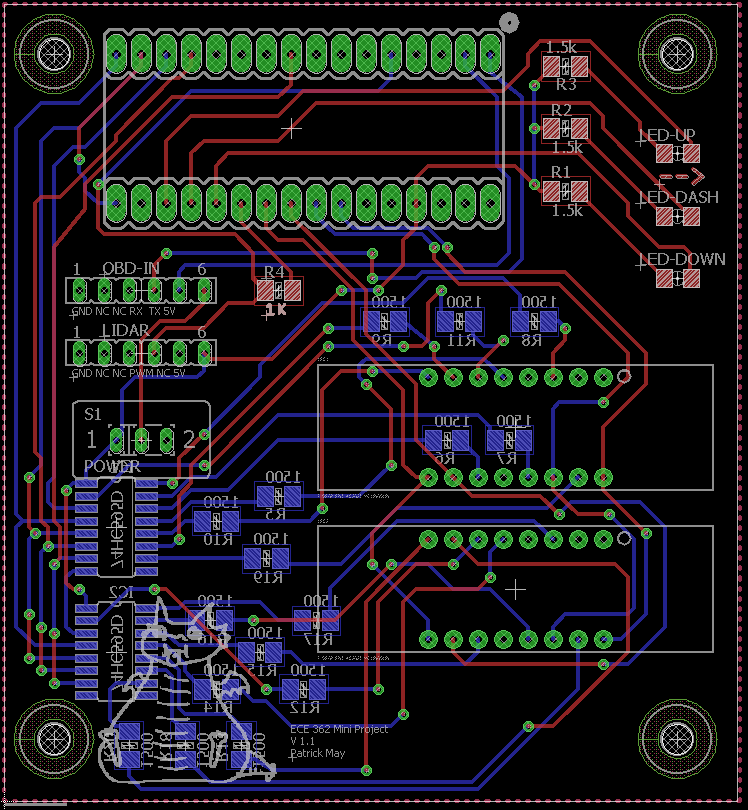
**PCB Layout Design**

Eagle files produced by Patrick May. Documentation produced by William Pierce. Full Eagle files may be found on the Git repository.

**Schematic**



**PCB Layout**

**

**Appendix C:**

**Software Flowcharts**

Software flowchart produced by William Pierce



**Appendix D:**

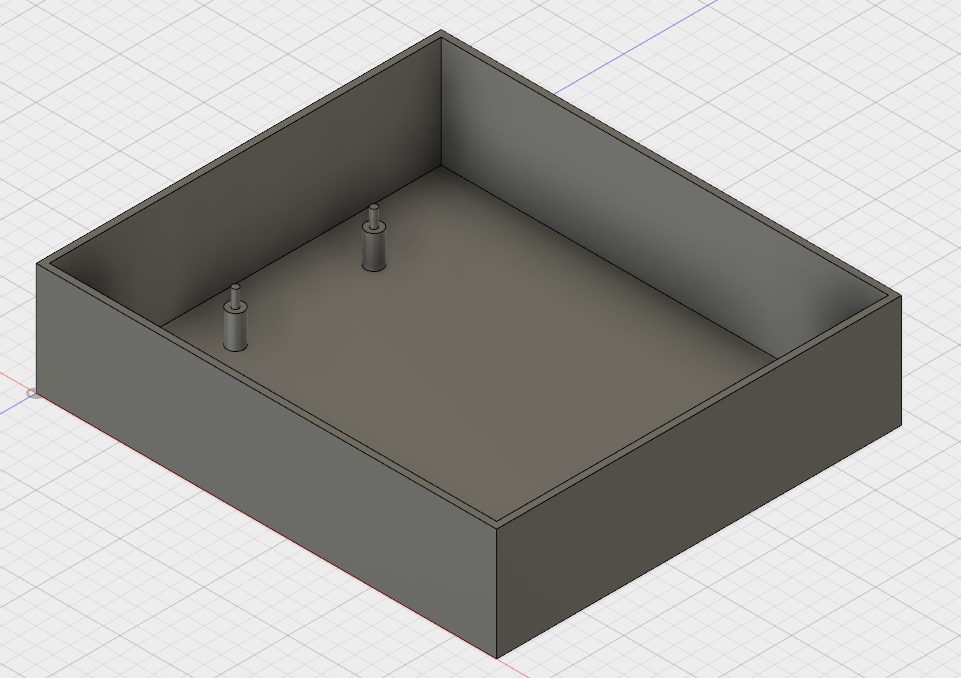
**Packaging Design**

**Main enclosure**

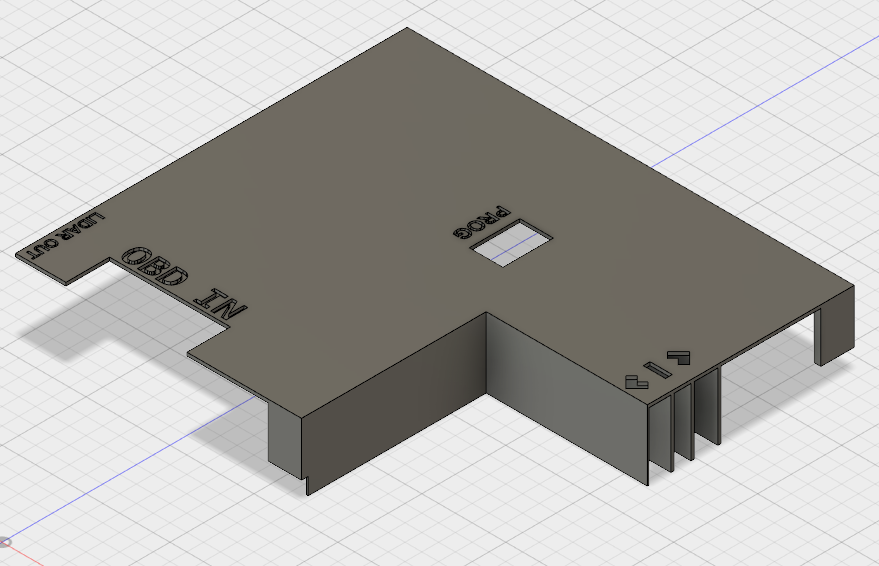
Enclosure produced by Pat, measured by Tyler

Documentation produced by Will

**Main box**

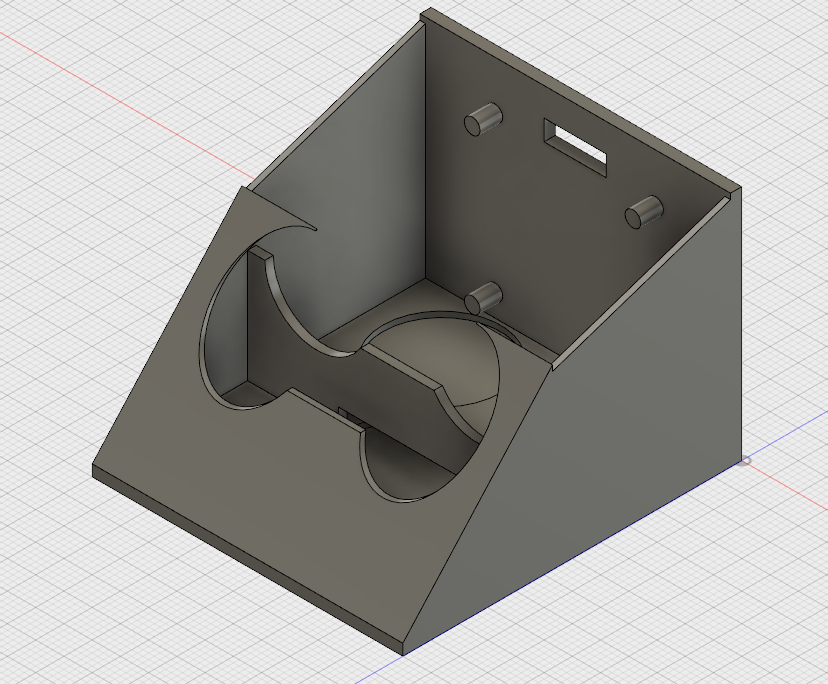


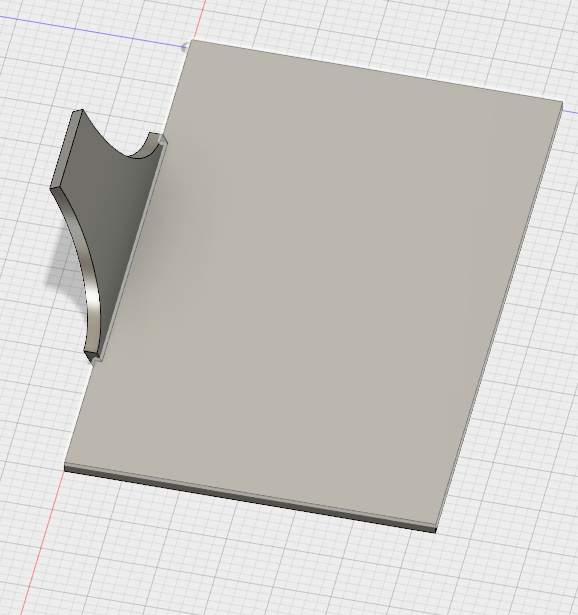
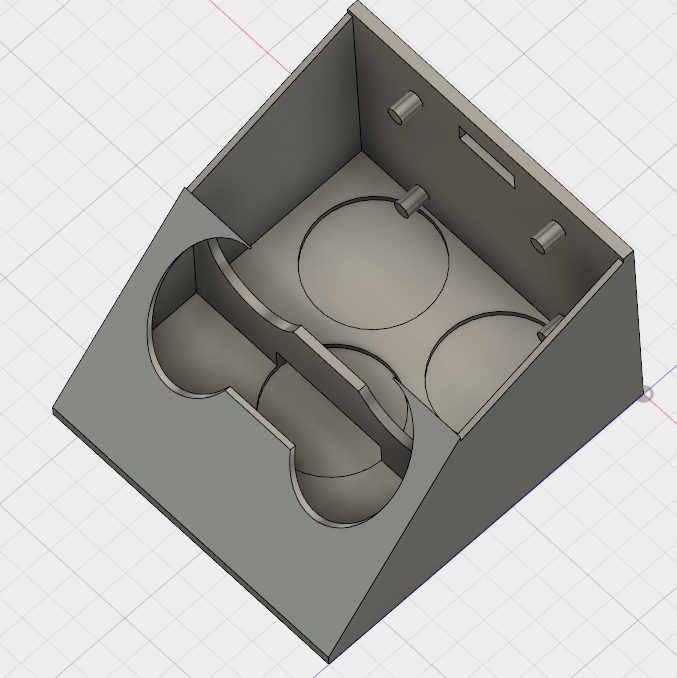
**Lid**



LIDAR Enclosure created by Will. Documentation Produced by Will

**LIDAR Enclosure**





**LIDAR Enclosure Lid**