

MICHIGAN STATE UNIVERSITY COLLEGE OF ENGINEERING

Final Design Report: CAN Lighting System

ECE 480 Team 12

Raymond Cashen, Maxton Coral, Kyle Granger, Eleazar Gutierrez & Stephanie Stolsky

4/29/2015

Facilitator: B. Wang



Sponsor: MSU Solar Car Team

Executive Summary

Team 12 is to design a high efficiency Controller Area Network (CAN) blinker system for Michigan State's Solar Car Team. The goal is to design fully functional printed circuit boards with the necessary integrated circuitry that will act as nodes or modules to control each of the 11 signaling lights. These modules can be daisy chained together and will be connected to the CAN bus. A node at the user interface will send a signal in the form of a CAN frame over the CAN bus. When the correct message for the specific module is received, the microcontroller will send a signal turning on the LED driver, thus illuminating the required light/lights on the solar car.

Acknowledgments

Throughout the spring semester of 2015, Team 12 has had help from various Michigan State professors and industry experts. Their help is greatly appreciated. The team wishes to thank Dr. Wierzba and Dr. Bingsen Wang for allowing the team to use their lecture notes, schematics, and photos on linear regulators and switch mode power supplies. This material was very helpful in preparing and presenting a technical presentation about linear regulators and switch mode power supplies. The team wishes to thank Dr. Bingsen Wang for being the faculty advisor. He managed the team and kept everyone on track throughout the design project. Specifically, he wanted to promote efficiency by suggesting that everybody on the team have their own task with minimal overlapping responsibilities. The team wishes to thank Bryant Barnes, at the Facility for Rare Isotope Beams at MSU, who led the team to looking into microcontrollers that operated with CAN libraries. Finally, the team would like to thank Michigan State's Solar Car Team, and our Solar Car Team sponsor Ian Grosh for the creation of the CAN blinker project and the time he dedicated to helping the team along the way.

Table of Contents

Chapter 1	
Introduction	1
Background	4
Chapter 2	
Exploring the Solution Space, Selecting a Specific Approach	8
Chapter 3	
Technical Description of Hardware Design Efforts	18
Technical Description of Software Design Efforts	24
Chapter 4	
Design Test Data	34
Proof of Functional design	37
Chapter 5	
Final Cost and Schedule	41
Conclusion and Final Summary	42
Appendix 1	
Technical Roles and Responsibilities	43
Work Accomplished	43
Appendix 2	
Literature and Reference Material	48
Appendix 3	
Detailed Technical Attachments	50

Chapter 1:

This chapter will be an overview of the project, design process, difficulties encountered, redesign process and background information about CAN systems. The project will be explained in further detail in subsequent chapters

Introduction

The goal of team 12's senior design project was to replace the MSU Solar Car Team's existing central microcontroller lighting system with a CAN node system, while maintaining control of all 11 signaling lights. The previous system was easy to implement, but had excessive amounts of wires connecting each signaling light to the central microcontroller, as seen in Figures 1 and 2. One can note that in Figure 2, there is a redundancy of wire to serve different signal light locations. In order to reduce the weight of wire used in the solar car, the implementation of the CAN system was decided as the best solution. This CAN system will be connected via a daisy chained node system, where each printed circuit board (PCB) node includes a microcontroller, a CAN controller, CAN transceiver, power regulator, and a LED driver. The design elements of this project were to construct a PCB layout of the lighting nodes, the code that controls the CAN system, the Ethernet CAN bus, a LED driver to supply continuous current to the LEDs, and be rated for an input voltage of 12V to power the microcontroller and CAN controller.

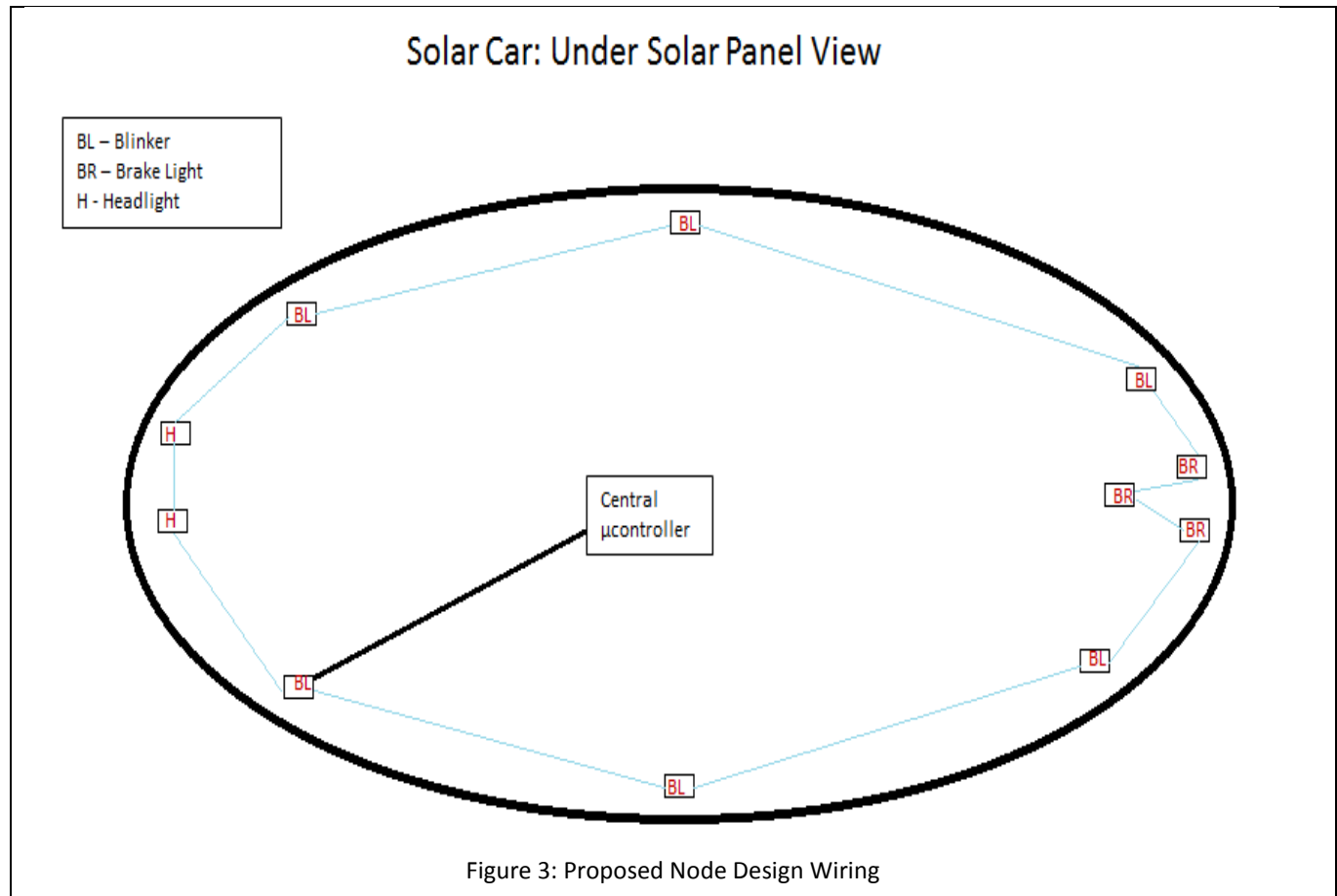


Figure 1: Current Solar Car Wire Usage



Figure 2: Current Solar Car Wire Usage

The reason for this project was to reduce the amount of wiring in the solar car in order to reduce the weight of the solar car. Prior to this project each light in the solar car had two wires connecting it to the central microcontroller for power, as seen in Figures 1 and 2, and four smaller wires for the CAN bus. With 11 lighting nodes in the solar car (6 blinkers, 2 headlights and 3 brake lights) the original design had 22 large wires, for supplying functionality of the signaling system. In a race car with limited power, that is a significant weight investment. The design chosen to reduce the wire weight was a daisy-chained node system, where the CAN bus was made using Ethernet wire. In Figure 3, the blue line represents the Ethernet wire.



A second consideration in the design was the power usage of the components, since the solar car has limited power available. This power limitation is based on the fact that the solar car does not have an alternator, like a standard car, to produce power once the battery has started the car. The only power sources are the solar energy converted by the solar panels and excess power from the electric motor.

The requirements of this project were that the system operate at automotive voltages of 5v - 35V, capable of transmission/reception of at least 250kbps, high efficiency voltage regulators for the LEDs and using RJ45 connectors for the CAN bus. The project specification document also listed the components currently used in the system. These components were not requirements but preferences of the sponsor. These components included the MSP430G2553 microcontroller, the MCP2551 CAN transceiver and the MCP2515 CAN controller.

In many ways this project is combination of two designs. The basic design of a CAN node system remained the same, but how it was implemented changed drastically in week 13. Prior to week 13, the design was using a microcontroller that required the CAN protocol to be programmed from scratch and communicate with a CAN controller through a serial peripheral interface (SPI). This design resulted in the CAN transceiver and CAN controller not communicating. This issue was investigated for two weeks with no forward progress and then new information was discovered about using a CAN library. This information was found during a conversation with a co-worker, Bryant Barnes, who had previous experience working with CAN systems.

With this new information more research was done and a new design was developed using a PIC18F258 microcontroller. This microcontroller has an integrated CAN controller along with an established CAN library written in C. The PIC microcontroller was also recommended to be used with a Microchip transceiver that the team had samples of. With this new information the second design was started. This new information and the extensive CAN library led the team to redesigning the code and node hardware layout in week 13. A functional prototype was produced within 5 days of the redesign.

Background

Fundamentally, the CAN system has multiple nodes that are able to transmit and receive messages via a single CAN bus. The CAN bus consists of four wires - CAN high, CAN low, power, and ground. Each node is hooked up in parallel to the CAN bus, so that it receives every message sent on the CAN bus, as seen in Figure 4. Each node must be able to transmit and receive messages on the CAN bus for a fully-functional CAN system.

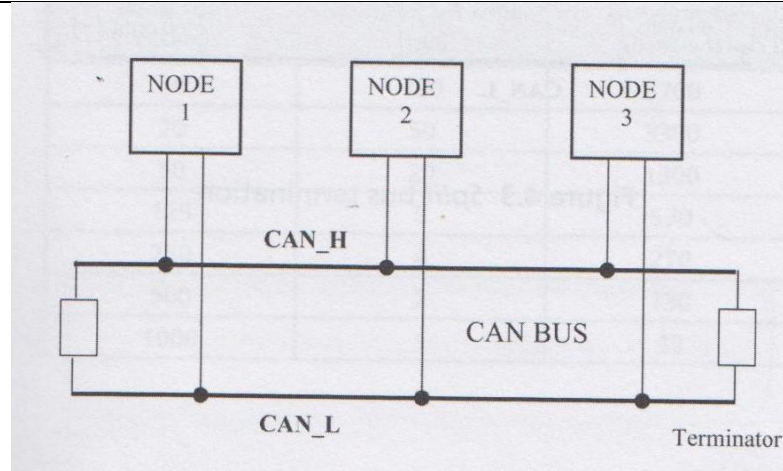


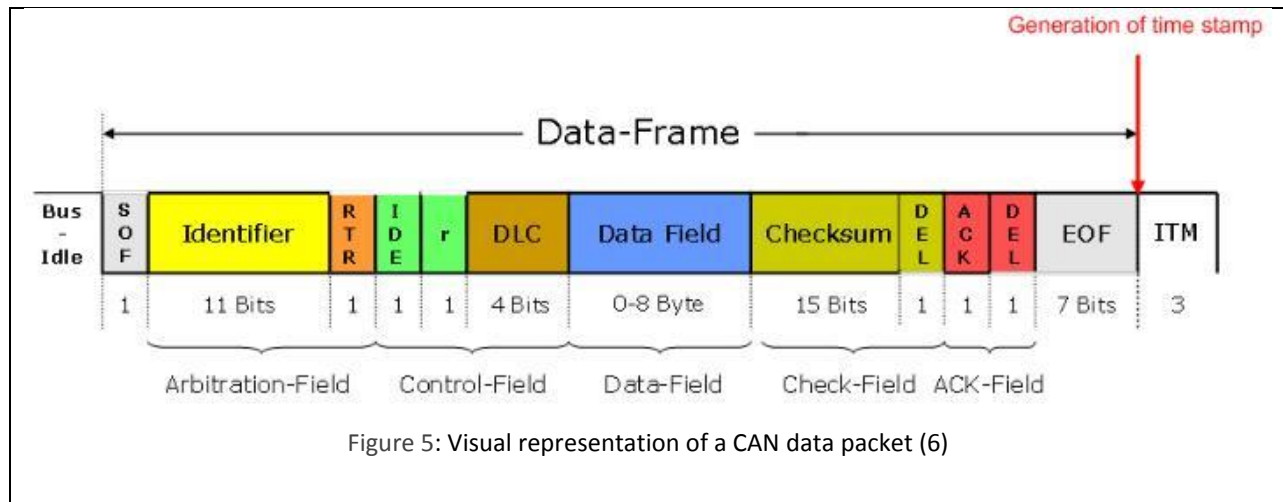
Figure 4 - Each node receives the same data due to its identical connection to the CAN bus [17]

CAN was developed by Bosch as a low-cost alternative to connecting all nodes directly to a central node. This was designed for the purpose of simplifying complex wiring that is a result of the increasing electronic systems in today's vehicles. For example, given a six node system, a direct connection configuration would require 36 connections. A CAN bus requires one circular bus and six connections. This means that for multiple-device systems, like the electrical systems in cars, the amount of wire and connections can be greatly reduced. In this project, 11 devices must be connected via a CAN bus.

The CAN protocol is a serial communication protocol. This protocol simplifies communication for the end user by automatically performing necessary overhead processes. Information is sent to the CAN bus by a node. Every node on the bus is able to sense the signal being sent. To send to a specific node, the transmitting node includes the arbitration ID and the data that it is trying to transmit. The priority of a signal in the network is decided by the arbitration ID. The arbitration ID is an 11 bit identifier in the CAN frame. On the receiving side, the CAN transceiver senses the incoming signal from the CAN bus. The transceiver then translates the incoming signal to a form that is able to be read by the CAN controller and processed by the microcontroller (3).

The CAN data packet is referred to as a CAN frame and is structured like many other data packets. The frame begins with the start of frame bit and ends with the end of frame bit. For error checking and signal reception confirmation, the CAN frame contains a cyclic

redundancy check (CRC) and acknowledgment, ACK packet. The CAN protocol uses arbitration ID's to resolve data priority. This arbitration ID can be 11 or 29 bits. The data corresponding to the signal is in the CAN signal segment of the frame. See Figure 5 for a visual representation of a CAN data packet (3).



The format for the data signal in the CAN protocol is a bit-wise designation (3). For example, the signal data for the blinkers will all begin with a 0 bit. Turning the front right blinker on will have a designation of 0x01 going high, while turning on the right center blinker will have a designation of 0x03 going high. Turning a light off will be a corresponding bit going low, Figure 6 is the proposed CAN frame.

CAN Frame:	
ARB ID	0x701
Length	4 Bits
Blinker Bit 0	0x00 = Front Right Blinker Toggle On = 1, Toggle Off = 0 0x01 = Front Right Blinker ON = 1, Off = 0 0x02 = Front Left Blinker Toggle On = 1, Toggle Off = 0

	0x03 = Front Left Blinker ON = 1, Off = 0 0x04 = Side Right Blinker Toggle On = 1, Toggle Off = 0 0x05 = Side Right Blinker ON = 1, Off = 0 0x06 = Side Left Blinker Toggle On = 1, Toggle Off = 0 0x07 = Side Left Blinker ON = 1, Off = 0
Blinker Bit 1	0x10 = Rear Right Blinker Toggle On = 1, Toggle Off = 0 0x11 = Rear Right Blinker ON = 1, Off = 0 0x12 = Rear Left Blinker Toggle On = 1, Toggle Off = 0 0x13 = Rear Left Blinker ON = 1, Off = 0
Break Bit 2	0x20 = Right Brake Toggle On = 1, Toggle Off = 0 0x21 = Right Brake On =1, Off = 0 0x22 = Left Brake Toggle On = 1, Toggle Off = 0 0x23 = Left Brake On =1, Off = 0 0x24 = Center Brake Toggle On = 1, Toggle Off = 0 0x25 = Center Brake On =1, Off = 0
Headlight Bit 3	0x30 = Right Headlight Toggle On = 1, Toggle Off = 0 0x31 = Right Headlight On =1, Off = 0 0x32 = Left Headlight Toggle On = 1, Toggle Off = 0 0x33 = Left Headlight On =1, Off = 0

Figure 6: Proposed CAN frame

There are many variations on the CAN protocol such as low speed CAN and single wire CAN (3). There are also international standards for the CAN protocol used in industry, such as ISO-11898 (4). The project will be implemented using the high-speed CAN protocol since it can be implemented with two wires and reaches the required minimum speed of 250 kbps (3). This project will not be adhering to international standards since it is not required.

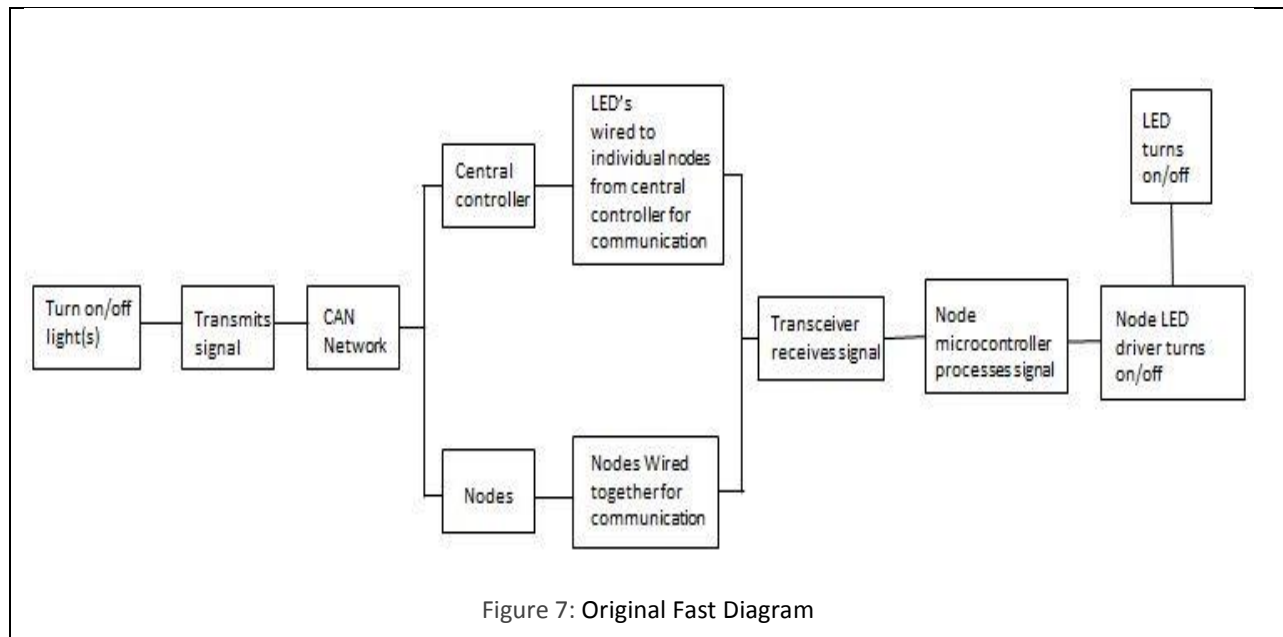
Chapter 2:

This chapter will address the initial design process, the mistakes made during the initial research process and the final design choices made in week 13. This chapter will conclude with a discussion of the budget and what an approximate budget would be if this CAN system were mass produced.

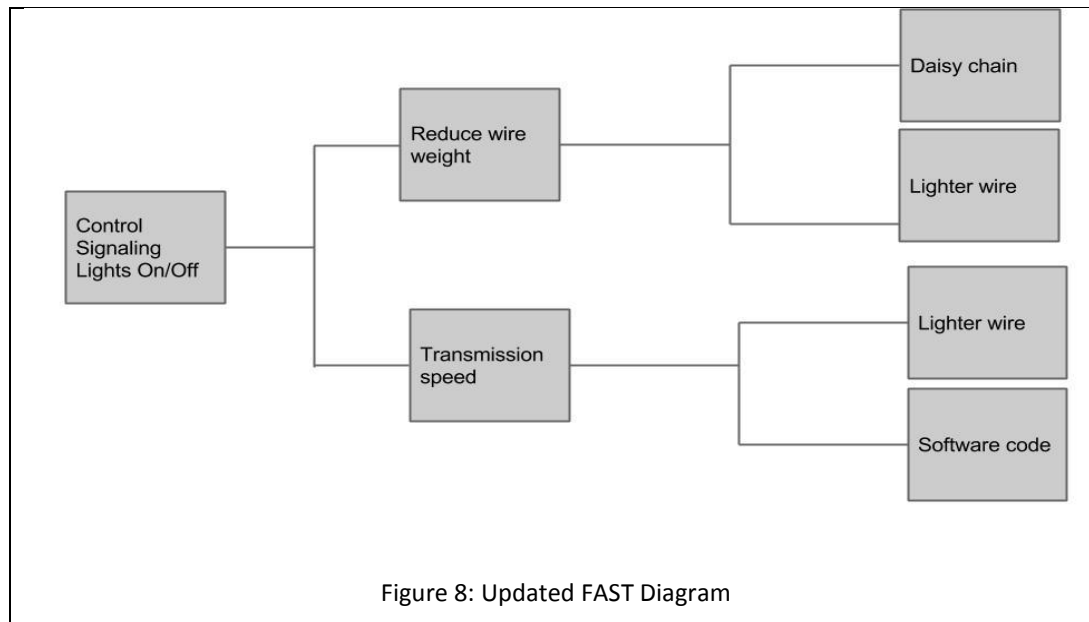
The subsystems of the CAN project were the node hardware design, node software design, CAN bus material choice, CAN bus fabrication, LED driver design, hardware selection, software/hardware integration, testing/troubleshooting, PCB design and final PCB fabrication. During the research process for the project, Team 12 made the critical errors of not looking at alternatives to the sponsor's suggestions or finding a person with first-hand CAN experience to interview and thus missed key elements used in the design selection process.

The errors team 12 made in the project began when researching for the design process. The team was so focused on the sponsor's suggestions that alternative options and levels of functions were not seen. There was a lack of understanding of how to make or use the FAST diagram as can be seen in the original FAST diagram (Figure 7). As seen in Figure 7, the FAST diagram was not levels of function, but instead a diagram of how the system works. If the FAST diagram were to be remade, the primary function would remain the same. The second level would be reduced wire weight and transmission speed. The third level, for reduced wire weight,

would be daisy chain and lighter signal wire. While, the third level for the transmission speed would be lighter signal wire, CAN transceiver, microcontroller and software code.



This alternate FAST diagram could have allowed the team to see the priorities of the project more clearly and make some better choices, like the CAT5 Ethernet cable and connectors, at an earlier stage of the project. That could have resulted in earlier ordering of the connectors and more time to integrate the Ethernet element into the system. The updated FAST diagram can be seen in Figure 8.



The deficits in research were also a problem in creating an accurate solution matrix, see Figure 9. The format was followed and honestly done, but the lack of understanding about the implementation of the CAN network lead to omitting a possible solution and under valuing the impact of a different solution. The value for the combined microcontroller/CAN controller ease of use was too low, since the combination of controllers removes the need for a SPI interface. Also, there should have been an entry for using a microcontroller that had a CAN library. This entry would have been vital to the implementation of the design. However, at that time of creating the solution matrix team 12 was unaware of CAN libraries. With those two pieces of information in the solution matrix, the best design would have been a combined microcontroller/CAN controller with a CAN library, not the separate microcontroller and CAN controller where programing the CAN protocol via SPI connection was necessary. This would have resulted in avoiding a last minute redesign and more time to optimize the project.

Engineering Criteria	Importance	Possible Solutions						
		Efficient switch mode buck converter to power the circuit	Use separate microcontroller and CAN controller	Use a combined microcontroller/CAN controller	Use microchip transceiver (currently used)	Use Texas Instruments CAN Transceiver	Daisy chain entire car in one loop	Daisy chain the front half of the car. Daisy chain the back half of the car in the other direction
Must function on automotive input voltages (5V - 35V)	5	9	9	9	3	9	3	3
Capable of transmitting and receiving @ at least 250kbs	5		9	9	9	9	9	9
Enter/exit ~2Hz toggle state over CAN	5		9	9	9	9	1	1
Reduce the amount of wire in the system	3	1					9	9
Connectors for LEDs must be easy to change	3						9	9
Manual ON/OFF control	2		9	9	3	3		
Ease of use (user interface)	1		9	1	9	9	9	3
Program in C	2		9	9	9	9		
Totals		48	162	154	120	150	128	122

Figure 9: Solution Matrix

The First Design:

The original FAST diagram and solution matrix lead to the decision to design the nodes using separate CAN controllers and microcontrollers. It was also decided to use a Texas Instruments CAN transceiver, SN65HVD233 that operated at 3.3V instead of 5V. This decision was made to avoid having two power supplies per node, to supply both the 3.3V and 5V voltages.

This design met the sponsor's requirements and suggestions in regards to maintaining use of the MSP430 microcontroller, so the sponsor would not have to learn a new microcontroller. The new aspects of the design were the node design and the daisy chain connections. The node layout was developed based on a logical progression of signal movement through the CAN system, as seen in Figure 10.

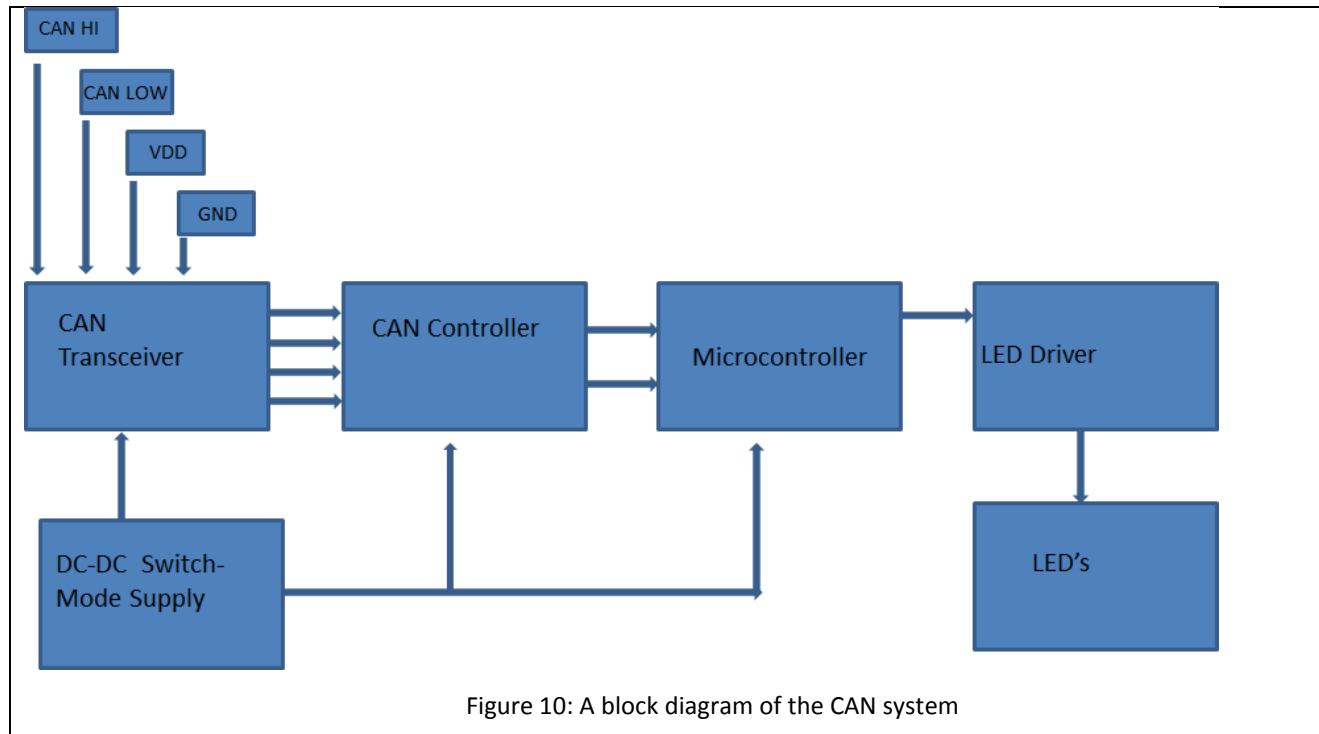
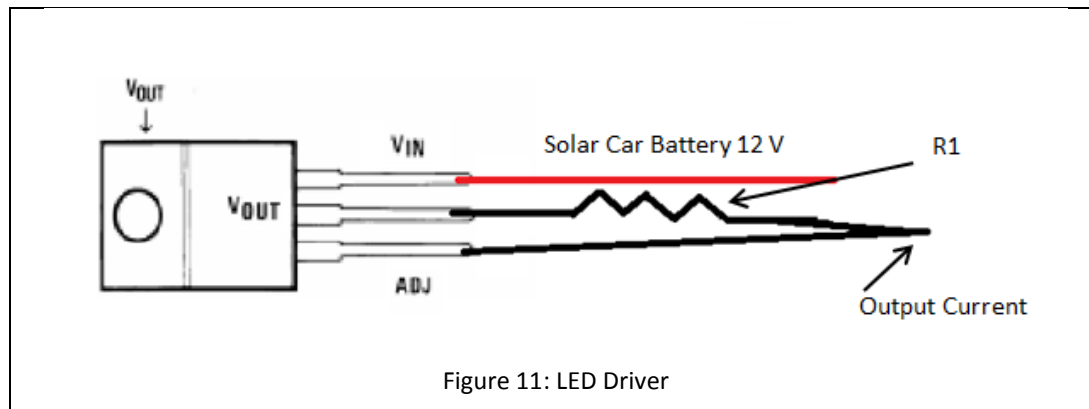


Figure 10: A block diagram of the CAN system

The continuous current LED driver was also designed at this time. The driver was basically a linear power supply. The driver was a voltage regulator with the voltage output and adjustment pins soldered together with a resistor to produce the needed current. The Vin pin would be connected to the power supply of the solar car, as seen in Figure 11. This system would have losses and require a heat sink for the voltage regulator. The customer requirement stated an efficient voltage regulator for the LEDs, but a constant current is required for the LEDs to perform optimally. Therefore, an efficient voltage regulator was chosen to be the basis of the design and limited losses were deemed acceptable. This decision was made due to the simplicity of the linear regulator and the complexity of making a control system for a switch mode constant current LED driver.

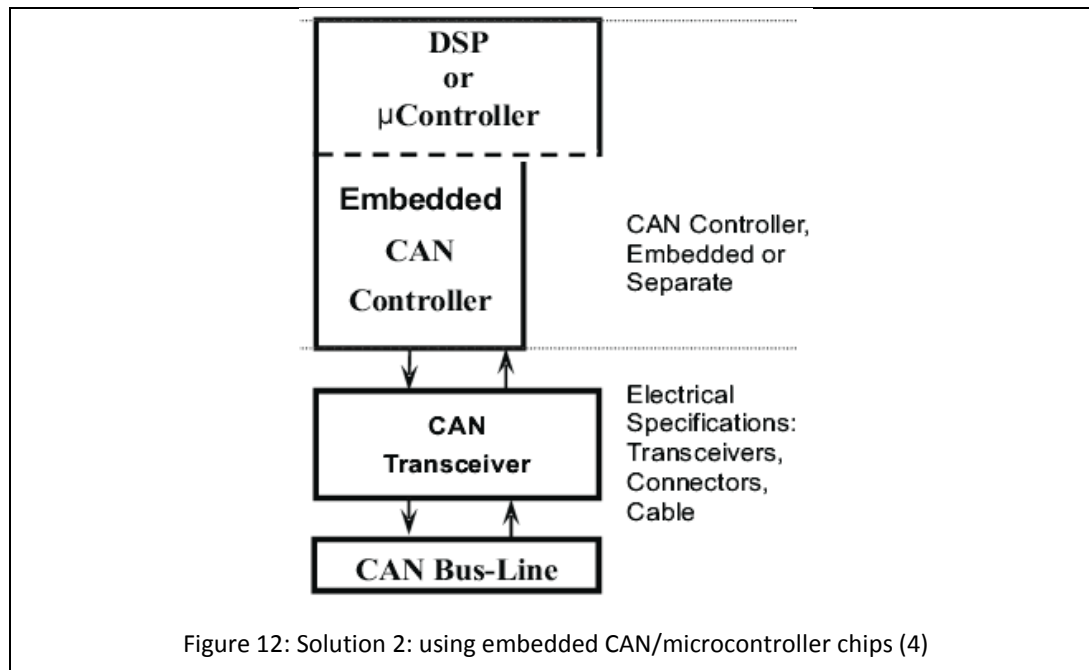


This design seemed like it would result in the successful implementation of the customers' requirements and suggestions since it used the existing microcontroller, the code implemented a state machine, and the node circuitry was an efficient integrated circuit layout. That was not the case when everything was constructed and programed. For an unknown reason the CAN transceiver would not engage and weeks were spent trying to troubleshoot the problem.

The Second Design:

After two weeks of troubleshooting with no new insights into the problem, one of the team members spoke to a coworker (Bryant Barnes) at the Facility for Rare Isotope Beams (FRIB) who had passed experience with CAN systems. When the problem was explained to him, Bryant asked what CAN library the team was using. This question lead to a new information stream and after days of research the decision was made to change designs.

The new CAN/microcontroller with a CAN library design was chosen because it would result in the most basic customer requirement; a functional CAN system. A simplified circuit connection of this design can be seen in Figure 12. The decision to redesign was made in week 13 of the semester, so it left very little time for the design process.



When doing research into CAN libraries the new components were selected since a CAN library is specific to a microcontroller. The selection of the PIC18F258 microcontroller was made for two reasons; it has an extensive CAN library and it is the recommended microcontroller to use with the MCP2551 CAN transceiver. The recommendation came from the Microchip webpage (CITE). Work immediately began on the new circuit layout, powering the new circuit, and the code.

The node now required a 5V power supply. The existing switch mode IC could provide 5V with changes to the discrete components in the switch mode circuit. The layout was similar to the original layout; however the CAN controller was integrated into the microcontroller.

With the use of the new programming libraries, the software was able to be built in a much more efficient fashion. The code was able to set up the registers and the CAN protocol with simple pre-defined functions. As stated below, a very methodical approach was required to test and verify functionality. The CAN bus was also constructed at this time. It was constructed from CAT5 Ethernet cable and terminated with 120Ω resistors.

Once the software and hardware were prototyped, they were integrated by programming a microcontroller and connecting the CAN bus to two nodes on two protoboards. There were a few corrections to the code, but within three days of receiving the new

microcontrollers there was a semi-functional CAN system. It was semi-functional since from time to time it would not work. The next test was using two LED nodes; the LEDs only lit when the corresponding input was selected. This was proof of concept and led to making the PCBs.

Once the pin connections were determined with the PIC18F258's datasheet and tests were conducted to confirm function, the new PCBs were designed. This process took longer than anticipated due to the lack of a footprint for the RJ45 connectors. When the boards were cut it took approximately four hours to cut the activation node. This led to the realization that on design day the PCB's would not be completed and the nodes would have to be constructed on breadboard or solder boards.

Schedule:

The initial schedule, Gantt Chart 1 in Appendix 3, was an ambitious timeline, but seemed achievable. The hardware prototyping was given 33 days and included calculating the values of capacitors, inductors, and resistors, ordering parts, designing a test PCB, and breadboard trials. The software programming was given 29 days and included figuring out how to implement the CAN frame, programming the microcontroller, testing the programmed microcontroller and troubleshooting the code. Final construction of the PCB's was given 11 day and optimization of the final project was given 10 days. This schedule would have resulted in having the project finished around April 16th.

The schedule was being adhered to until the CAN transceiver in the first design failed to communicate. Once that problem occurred breakout boards were made in an effort to see if communication could be established with the transceiver. There was no activity from the transceiver and a solution was not determined, the team fell behind the schedule and it was determined that a functional CAN system was not likely to be completed in time for May 1st.

Once the design was changed a second Gantt chart was made, see Gantt chart 2 in Appendix 3. This Gantt chart had an abbreviated timeline, since three weeks remained before design day. There were seven days allocated to hardware and software design. This design time was simultaneous so all team members were working on some element of the design. Six days

were called out for integration and testing, although some of this testing was done during the first week of software development. Ten days was allotted to the final design and this design segment began on the same day as integration and testing. This allowed team members who were not working on the physical prototype to work on the final report, design day poster, the design day table demonstration and the design day presentation. The team has followed this timeline, with the exception of the PCB production time. The Gantt chart was created before the amount of time to cut the boards was determined.

Budget:

Figure 13 shows the cost associated with manufacturing one board versus mass production. However, the budget does not reflect the true amount of money that was invested in developing and testing the multiple designs. One of the purchases that turned out to be extraneous was the LED's used as a standard for the American Solar Challenge race. The purchased LEDs were thought to be necessary in order to ensure the LED's the solar car team was using were bright enough compared to the standard. The cost of the LED's was \$45.21.

Another purchase that went unused was the car battery. The team wanted to power the design day demonstration off of a car battery. Unfortunately, the redesign prevented adequate time to interface the system to the battery. The cost of the battery was \$75.42. The team also bought discrete components for the switch mode power supply to produce an output voltage of 3.3V. The cost of the components was \$39.69. The total cost of these three orders was \$160.32. These expenses are what lead to the team going over budget.

The remaining expenses were for the second design. The PIC microcontrollers and transceivers were \$135.87, the programming interface for the microcontrollers was \$73.94, the discrete components for switch mode power supply (to produce 5V) was \$41.74, a research book was \$26.99, RJ45 connector tools were \$35.56 and solder boards and PCB mountable RJ45 were \$45.76.

Part	Cost	Production Cost
PIC18F2585	\$5.90	\$5.62
16 MHz oscillator	\$0.33	\$0.13
10 pF capacitor	\$0.20	\$0.07
CAN Transceiver	\$1.02	\$0.78
Resistors	\$0.24	\$0.07
Printed Circuit boards	\$3.00	\$2.19
Total	\$10.69	\$8.86

Figure 13: Project Production Budget

Chapter 3:

This chapter will focus on the technical design work that the team implemented for the CAN blinker system, focusing on the hardware and software design efforts of the project.

Hardware

To create a functional CAN blinker system under a variety of different input conditions required the implementation of certain hardware components into their design. Since the project description requires high efficiency, lightweight, and small package size components, the team began researching different integrated circuits (IC's) that would work in their design. From the beginning of the project, team 12 understood that a microcontroller, CAN controller, CAN transceiver, switch mode power supply, discrete switch mode power supply components, and discrete CAN system components were going to be required in the design of the CAN blinker system.

First, the team began viewing IC component manufacturer's websites to see what would be available in term of hardware components and the specifications that went along with the devices. This included efficiency, packages, voltage ratings, current ratings, and power ratings. The initial hardware components that met the project's requirements were:

MSP430G2553 - Chosen Microcontroller

MCP2515 - Chosen CAN Controller

MCP2551 - Chosen CAN Transceiver

LMR14206 - Chosen Switch Mode Power Supply

LM350 - Chosen LED Driver

The components listed above were all chosen for the reasons stated below.

The MSP430G2553 was an acceptable choice because the team had previous experience coding the microcontroller along with the fact that Texas Instruments offers helpful tools that work with the MSP430. The TI Launchpad development board was used to interface with the computer. Code Composer studio was the tool used to program the MSP430. The IC has low power consumption, operates at a low input voltage, and offers a variety of packages with surface mount options (7).

The MCP2515 was an acceptable choice because it implements CAN V2.0B at a data rate of 1 Mb/s (cite). The IC also has low power consumption, operates at a low input voltage, and comes in a variety of small packages with surface mount options (9).

The MCP2551 was an acceptable choice because it supports 1 Mb/s operation, allows up to 112 nodes to be connected to a single CAN bus, operates on a low input voltage, and comes in a variety of small packages with surface mount options (10).

The LMR14206 was an acceptable switch mode power because it holds a tight accuracy for powering digital IC's, wide range of operating voltages, adjustable range of output voltages, short circuit protection and comes in a tiny surface mount package (8).

The LM350 was an acceptable choice for LED regulator because it outputs 3A of current over a wide range of voltages, has internal thermal overload protection, and internal short circuit current limit (14).

Second, the team began to prototype with the free samples that were initially chosen. The IC's that were sampled were all surface mount packages which lead to an issue when prototyping the design. The surface mount packages are designed for computer aided manufactured printed circuit boards (PCB's). These packages have a small footprint to reduce cost when they are implemented in a mass produced design. The surface mount packages have small leads that are close together to reduce stray inductance therefore improve device performance. Since the team was still in the prototyping phase of the first design, it was very

difficult to continuously have to re-design a PCB and have it milled by the ECE shop every time a new wiring of components needed to be tested. This inability to quickly change around the wiring of components was an issue that needed to be solved. Because the leads to the surface mount components were very close together it made it very difficult to hand solder, and gave the team issues when trying to solder extension wires to the device.

The purpose of soldering wire to the package was to gain the ability to wire the components via breadboards versus having a PCB cut. This idea would let the team change around the wiring of components quickly. This idea in theory works but the vigorous and constant movement of the wires causes the nimble leads to break off rendering the IC useless. The team then cut small PCB breakout boards for every IC, making it easy to solder wires to the board and eliminating the stress on the leads while still maintaining the versatility of quick breadboard prototyping. The breakout board solution worked well, and the team could load different lines of code to the microcontroller as well as change the wiring of the various components in a very short time.

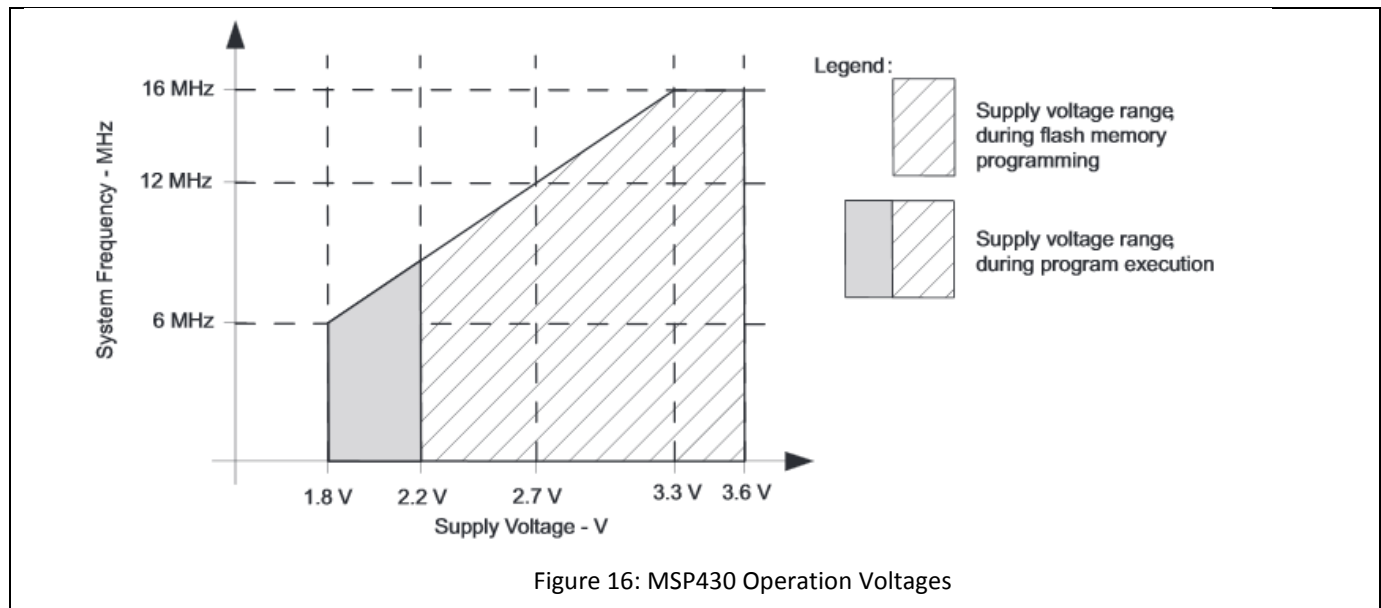
A second issue observed while prototyping on the breadboards was the need to use two bench power supplies. Multiple powers supplies were needed since the devices had various required input voltage ranges. To implement this design on a PCB it would demand two different switch mode power supplies for every node of the CAN system. Having two operating voltages would therefore increase the number of discrete components, increasing the complexity and size of the PCB, which results in an increase cost. An increase in the cost of the design was deemed unacceptable, so a different CAN transceiver was selected.

VDD	VRS	TXD	CANH	CANL	Bus State ⁽¹⁾	RxD ⁽¹⁾
4.5V ≤ VDD ≤ 5.5V	VRS < 0.75 VDD	0	HIGH	LOW	Dominant	0
		1 or floating	Not Driven	Not Driven	Recessive	1
	VRS > 0.75 VDD	X	Not Driven	Not Driven	Recessive	1

Figure 14: MCP2551 Operation Voltages

DC Characteristics			Industrial (I): T _{AMB} = -40°C to +85°C V _{DD} = 2.7V to 5.5V Extended (E): T _{AMB} = -40°C to +125°C V _{DD} = 4.5V to 5.5V			
Param. No.	Sym	Characteristic	Min	Max	Units	Conditions
	V _{DD}	Supply Voltage	2.7	5.5	V	

Figure 15:MCP2515 Operation Voltages



One can see in the three attached input voltage datasheet Figures 14, 15 and 16 that the MCP2551 CAN transceiver has a minimum operation voltage of 4.5V, and the MSP430 microcontroller has a maximum operation voltage of 3.6V, while the MSP2515 has a range from 2.7V to 5.5V. This shows that the MSP430 cannot operate on the same power supply as the MCP2551. The team decided that the best solution to this problem was to replace the CAN transceiver. The MSP2515 CAN controller and the MSP430 both operated at 3.3V so it was that it would be expedient to find a CAN transceiver that operated at 3.3V (7)(9)(10).

After searching on another manufacturer's website, Texas Instruments (TI), the team found a transceiver that would be able to function under the same voltage constraints as the other devices. The change from the previous Microchip's MCP2551 to Texas Instruments' SN65HVD233 allowed the node to function on one switch mode power supply.

The changing of transceivers would play an insufficient role because the team realized while testing with the MSP430, that trying to build the CAN protocol from scratch would be too time consuming. This will be explained in more depth in the software section of the report.

Due to the amount of time it would take to code the CAN protocol from scratch, it was determined that a redesign was necessary. After researching microcontrollers and CAN libraries, the first change made to the design was to select a different microcontroller. The microcontroller of choice was the PIC18F258. This microcontroller has extensive documentation regarding the implementation of the CAN protocol, along with pre-made CAN libraries. With the new microcontroller the team began to test the CAN system on breadboards. The final hardware selection was:

PIC18F258 - Chosen Microcontroller

MCP2515 - Chosen CAN Controller

MCP2551 - Chosen CAN Transceiver

LMR14206 - Chosen Switch Mode Power Supply

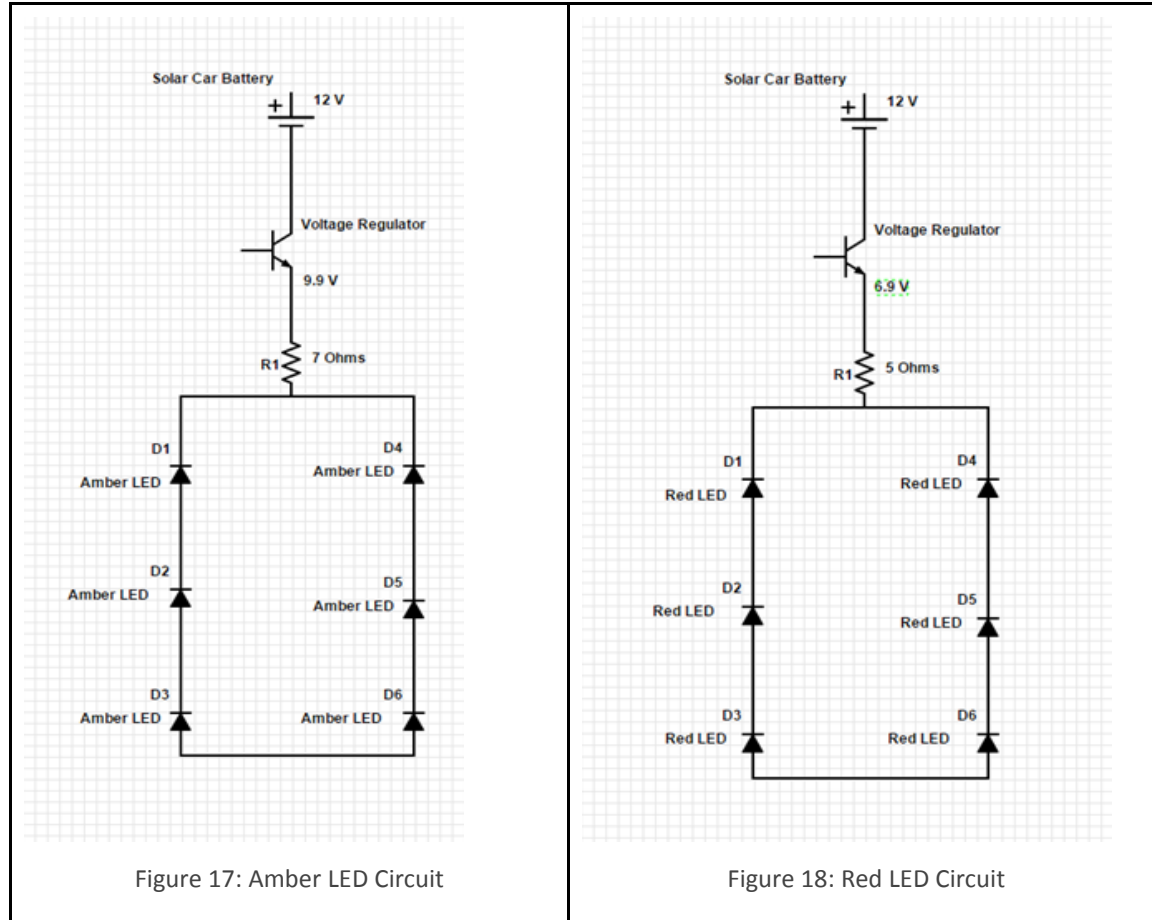
LM350 - Chosen LED Driver

Changing the microcontroller late into the semester was a risky move, but was determined necessary to have a functional project in the allotted time. With the CAN example code in "Controller Area Network Project," team 12 was able to achieve a two-node communication via the CAN bus (11).

The LEDs used in the LED boards for the blinkers and brake lights are Nichia LEDs. The amber LEDs (blinkers) have a voltage range of 3.3 – 3.8V, a maximum forward current of 700mA and a maximum junction temperature of 125 °C. The red LEDs (brake lights) have a voltage range of 2.3 – 2.6V, a maximum forward current of 700mA and maximum junction temperature of 120°C.

The LED boards are two parallel sets of three series LEDs, as seen in Figures 17(Amber) and 18 (Red). For the amber LED boards this leads to $3.3V * 3 = 9.9V$, and $(700 * 10^{-3}) * 2 = 1.4A$. The red LED boards need a voltage of $2.3 * 3 = 6.9V$ and 1.4A. To achieve this for amber LED(s), Ohm's Law is applied to get $9.9V = R * 1.4A \rightarrow R = 9.9 / 1.4 = 7.07\Omega$. The available resistor closest

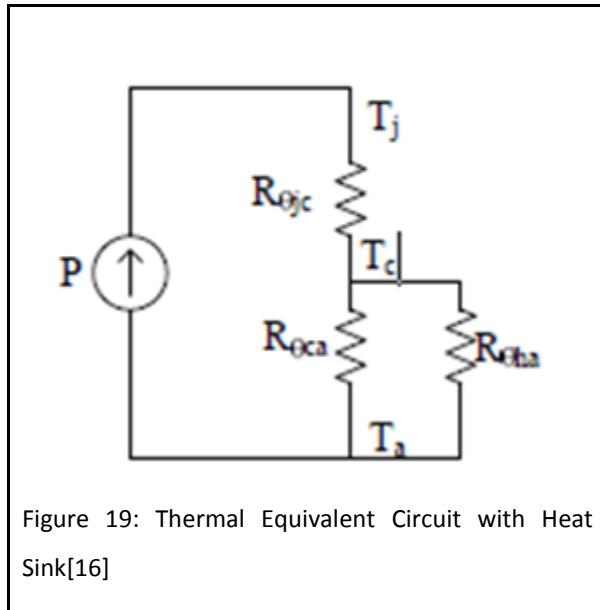
to this value is 7Ω . For the red LED(s), $6.9V/1.4A = 4.93\Omega$. The closest resistor value to this is 5Ω . Due to the power dissipated in the regulator the resistor will need to dissipate 2W of power.



The voltage regulator needed to supply 1.4A of output current, 9.9V or 6.9V output voltage with 12V as an input voltage. The LM317 was used in lab and can provide 1.5A of output current. This would technically meet the specifications. However, there is no margin for error and since the current is almost at the maximum the regulator can supply the regulator would get extremely hot. In order to have some margin, the regulator selected was the LM350. This voltage regulator has the appropriate voltage range and can provide 3A of output current.

To determine if a heat sink was necessary a thermal equivalent circuit was used (Figure 19). The equation from this circuit was $T_j = T_a + P_{loss} * (R_{\theta jc} + R_{\theta ca} * R_{\theta ha} \backslash (R_{\theta ca} + R_{\theta ha}))$, [16] where T_j is $125^\circ C$, $T_a = 40^\circ C$, $R_{\theta jc} = 3^\circ C/W$, $R_{\theta ca} = 50^\circ C/W$ and $P_{loss} = 2.94W$ (Amber) or $7.14 W$ (Red).

The thermal resistance values were from the LM350's data sheet [15]. This lead to the thermal resistance of the heat sink for the amber LED being $R_{\theta_{ha}} = [(T_j - T_a) / P_{loss}] - R_{\theta_{jc}} \approx 22.51 \text{ } ^\circ\text{C/W}$. The red LED needed a heat sink with a thermal resistance of about $11.7 \text{ } ^\circ\text{C/W}$.



Software

Messages sent along the CAN bus are referred to as frames. An example of a CAN frame can be seen in Figure 20. This frame contains the start-of-frame (SOF) and end-of-frame (EOF) bit to signal when the data frame starts and ends for correct transmission [17]. The frame contains the arbitration ID, which has a priority assignment that determines which node the transmitting microcontroller communicates with. The largest field in the frame is the data field. This field varies in size depending on the size of the data that needs to be sent. For this project, one bit in the data field corresponded to one device, and all of the signaling light nodes shared the same arbitration ID. Since the nodes shared the same arbitration ID, they had the same priority. This should allow the entire CAN system to be integrated into the car without any conflicts following design day. Every node receives every message, and responds only to the arbitration ID that it is programmed to recognize.

The other fields in the frame correspond to checking the frame - making sure that it is ready to be sent or received correctly. The checksum is an error check that determines if a

message has been received correctly. If the checksum is not accurate, the transmission would be resent. There are several other flags that need to be cleared in order for a message to be sent. One of these flags is the busy flag, which indicates whether or not the CAN bus is available for transmission. If multiple frames are being transmitted by multiple microcontrollers, the frame with the lowest arbitration ID (highest priority) will “win”, and the other nodes will stop transmitting. That is how CAN systems handle multiple signals on the bus.

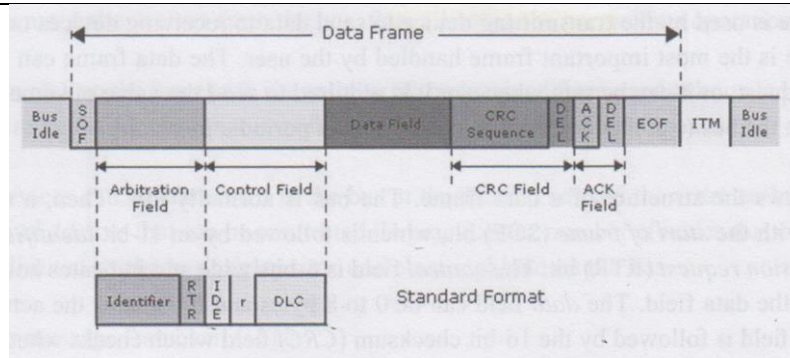


Figure 20 - An example of the standard CAN frame. Each device must use this protocol when transmitting or receiving a message on the CAN bus. [17]

In the first design for this project, each node consists of three primary devices: a CAN transceiver, a CAN controller, and a microcontroller. The transceiver interprets messages from the CAN bus into a format that the CAN controller can process. The transceiver converts the serial messages on the CAN bus into peripheral, digital logic-level signals. After loading the serial message into a buffer, it is sent to the CAN controller. The CAN controller passes the message through built in filters that determines whether or not the microcontroller will respond to that message. The above concept was the same in the second design, except the CAN controller and microcontroller were in one unit.

Each node in the blinker system has the same arbitration ID, and hence, the same priority. Since the nodes at the signaling lights respond to an identical arbitration ID, the code for the microcontroller would have to check the data portion of the CAN frame for instruction on functionality. Line 284 of the final code of the Front Right Blinker shows the process of checking for the proper arbitration ID. Lines 277, 278 and 279 of the final code for the Front Right Blinker checks the bits of the data portion of the CAN frame. The microcontroller,

assuming it would respond to the message, would drive the hardware to operate the blinkers/headlights, according to what the message required. The logic behind this will be further detailed later in the report.

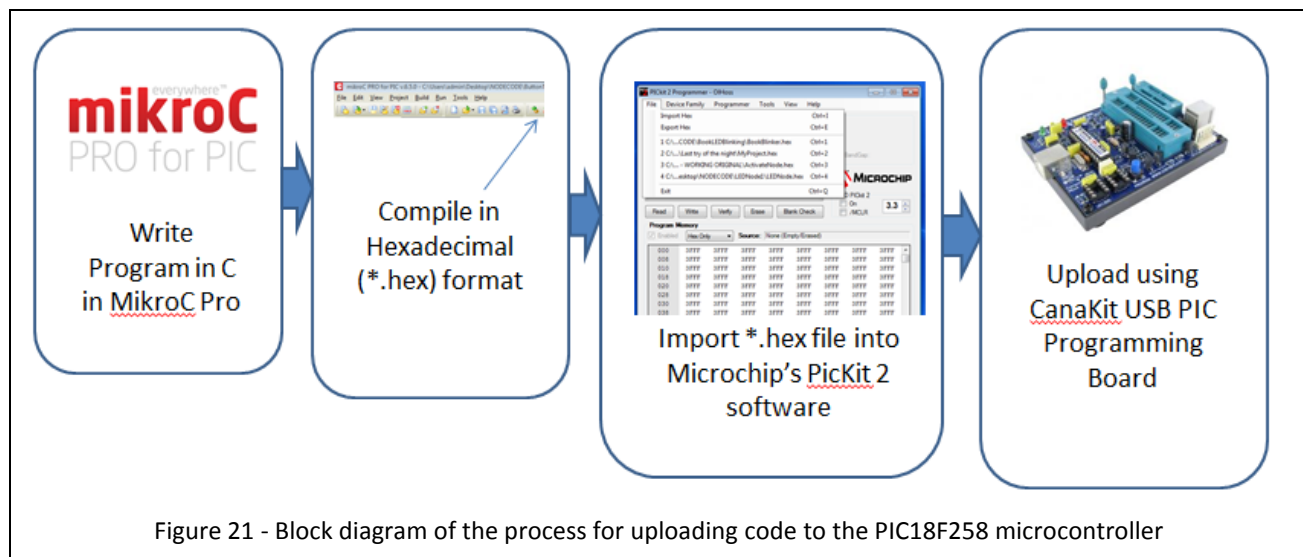
The initial software design tried to write the I/O relationship for the CAN protocol from scratch using the MSP430G2553 microcontroller, which was used in the ECE-480 lab portion. This task proved to be quite difficult – the number of functions required is large, and requires a very deep understanding of both the CAN protocol and microcontroller configuration for functionality. This led to a much more complicated usage of the MSP430 than was ever reached in class. Designing these functions required a large amount of research into individual registers of the microcontroller and CAN controller, and was have been very difficult to debug. The only available debugging tool was a logic analyzer that was quite old, and had little documentation describing how to operate the device. This meant that the only way of confirming a functional command node was to check that the registers in the MSP430 were set correctly. However, that does not indicate whether or not those registers were sent out to the CAN controller, and there was not a convenient way to debug that process.

Instead, the team found an online library of CAN functions designed for the Microchip PIC microcontroller family. The library used, made by MikroElektronika, required the use of MikroC Pro programming software. For assistance utilizing this library, the group found a book on Amazon titled “Controller Area Network Projects”[17]. This book detailed the creation of a CAN network with the use of the MikroElektronika CAN library and all of its configurations. Learning from this book gave the group the understanding to create a functioning CAN system. The book, however, highlighted use of development boards that were too expensive for the budget.

The CanaKit USB PIC Programmer was a good cost-friendly alternative to the programming boards referenced in “Controller Area Network Projects.” The CanaKit USB PIC Programmer was exclusively compatible with Microchip software, which the team did not have access to. Since the libraries used utilized the MikroElektronika software [13], the team had to first compile the code to a more universal hexadecimal format. The Microchip software could

then upload the code using the CanaKit USB PIC Programmer. Figure 21 shows a box diagram of the process.

Much trial and error was required, even after having working code. Changing the wiring between protoboards or breakout boards affected the performance of the bus, and caused a lot of time to be spent debugging. Most issues were that something was not configured correctly, something was not wired correctly, something wasn't grounded, etc. This resulted in the destruction of two of our PIC microcontrollers.



Creating a functional CAN system was only achievable by following a strict and slow process. Changing too many variables at one time made debugging impossible. Using the new code and the PIC18F258 microcontroller, the first goal was to create a “Hello, World!” program. This meant creating a program that was very basic, using only one I/O channel, with very basic logic and outputting a visible verification of the functionality of the microcontroller. The program created did not use the CAN bus or even set up the CAN protocol yet - it simply made an LED blink on a specific port. This code can be seen in Figure 22. After output was confirmed, input functionality was to be verified. To do this, a program was written to illuminate an LED when the compression of a tact switch was sensed. This can be seen in Figure 23. After full functionality of the input and outputs of the microcontroller were confirmed, the team was able to proceed.

```

• void main() {
•   TRISC = 0; //Set all PIN C to output
•   for(;;)    //Infinte loop
•   {
10      PORTC.F0 = 1;      //Turn on LED
•      Delay_Ms(1000);    //Delay 1 second
•      PORTC.F0 = 0;      //Turn off LED
•      Delay_Ms(1000);    //Delay 1 second and return to top of loop
•   }
• }

```

Figure 22 - Code for "Hello World!" program. This simply makes the LED blink.

```

• void main() {
•   TRISB = 0;      //Set all PORTB to output
•   TRISC = 0x01;   //Set PORTC.0 to input from switch
4   for(;;)         //Infinite Loop
•   {
•   if(PORTC.F0 != 0) //Button not pressed
•   {
•   PORTB.F2 = 0;    //Turn off LED on PORTB.F2
•   }
10  else             //Button Pressed
•   {
•   PORTB.F2 = 1;    //Turn on LED on PORTB.F2
•   }
•   Delay_Ms(100);  //Delay 1/10 seconds
•   }
• }

```

Figure 23 - Verification of input as well as output

The next goal was to create a code that would transmit from one node, across a small CAN bus and received by another node. The idea is similar to that of Figure 23. In this case, one node would sense the input; send data via the CAN bus to another node, which would output the illumination of an LED. The transmitting node was to send a CAN frame upon sensing an input on one of its ports. This meant the code had to output a CAN message and that message had to be converted into a serial communication on the CAN bus. Then, that signal had to be converted back to a peripheral signal and interpreted by the CAN controller on the receiving

node. Finally, the microcontroller must decide whether or not it would turn on the LED based on that information. Conceptually, this process seemed simple, but there are many variables which could go wrong. After some trial and error, specifically modifying configuration bits and confirming wiring setups, there was success in a two-node CAN system. Figure 24 shows a block diagram of the two node system.

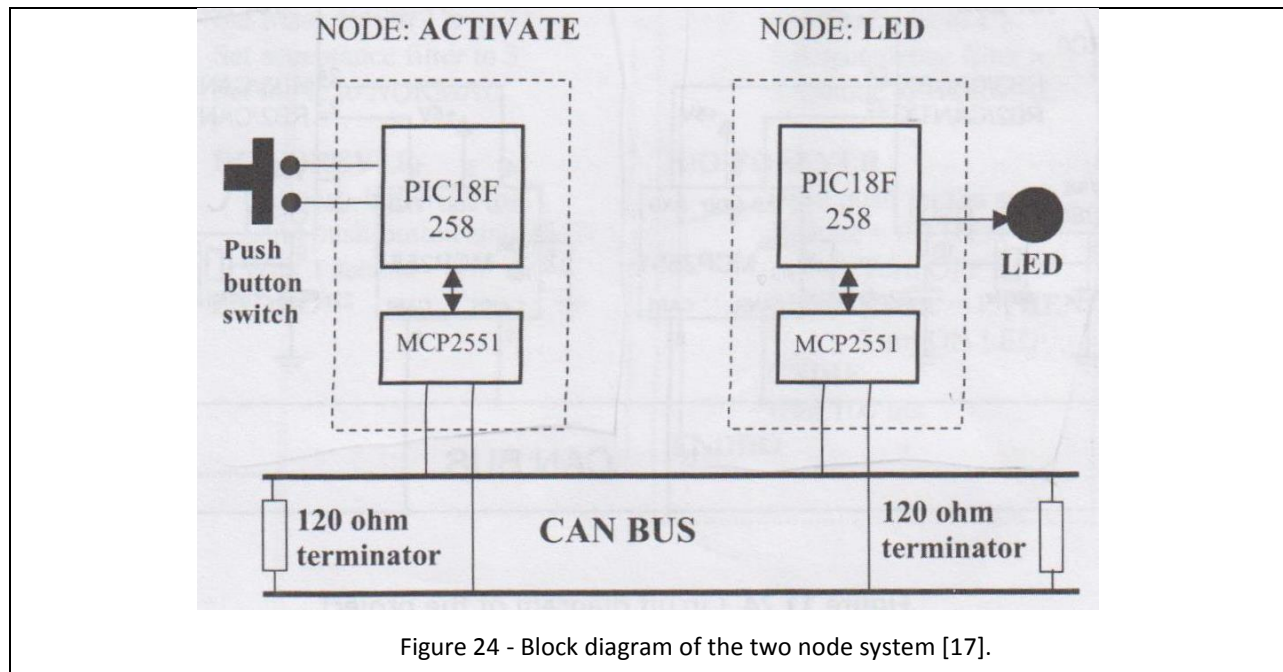


Figure 24 - Block diagram of the two node system [17].

In the above Figure 24, the team was able to press a button on NODE: ACTIVATE and see the output of the illuminating LED on NODE: LED above. The CAN library used can be found at <http://www.mikroe.com/mikroc/pic/libraries/>. Both the setup of the CAN system on the microcontroller and the use of the CAN functions iterated in the code for the final system. One can reference the final code in the appendix for the setup of the CAN protocol. Figure 25 shows an example of the ACTIVATE node. It is important to note that this is just the functionality of the ACTIVATE node and omits the setup portion. The setup can be found in the final code.

```
for(;;)
{
    push_button1 = PORTC.F0;    //Get button state

    if(push_button1 != 0)
    {
        sdata[0] = '0';
    }
    //If not pressed
    else
    {
        sdata[0] = '1';
    }
    //If pressed

    //Send button state in data frame over the CAN bus
    CANWrite(LED_ID, sdata, 3, send_flag); //Send button state
    Delay_Ms(100);                       //Delay 1/2 second
}
```

Figure 25- Logic of NODE: ACTIVATE

The next goal was to create a 4-node system with 1 master node and 3 slave nodes. The CAN bus was set up with 1.5 feet of wire between each node. Since there was no easy way to connect the nodes to a bus, team 12 had to solder the bus wires directly to the transceivers leads. This soldering was problematic since the leads were very delicate and broke off, which rendered the transceivers nonfunctional. This breakage delayed debugging the code. The solution to this problem was to make breakout boards for the transceivers in Eagle. These breakout boards were manufactured in the MSU ECE shop.

After previous issues were resolved, the team made another iteration of the prototype. This iteration allowed connections to the CAN bus using a protoboard and no soldering was required. Initially, this prototype was not functioning properly either. The LED nodes were responding unpredictably and no clear pattern was established. Sometimes the LED would respond to the correct button, and other times it would not. Each microcontroller was cleared and reprogrammed, and all wiring was verified to be correct. Yet the issue persisted. The “Hello World!” program was then reinstalled on the microcontroller and the LED would not blink on the ACTIVATE node. Ultimately, moving the master node off of a protoboard and onto a new protoboard fixed the issue. This suggested that the first protoboard used was defective. The

protoboard problem example showed how multiple variables affected the design, and illustrated that often the problem could be something unexpected. However, with work, a functional four-node system was ready to demonstrate to Professor Udpa and Professor Grotjohn on June 22nd.

The code originally used for the purpose of testing CAN communication was heavily based on the code in “Controller Area Network Projects” [17]. However, this code would not work for all 11 nodes. The data portion of the CAN frame could only be a maximum of 8 bytes. The original code used in the book used an entire byte of data for sending one function. The data frame (sdata[0] in the code from Figure 25- Logic of ACTIVATE node) was defined as an unsigned char array, with 1 byte per position in the array. To send a function to another node in the book, the code “sdata[0] = ‘1’ ” was used. This code would have limited the system to sending a maximum of 8 functions (1 function per byte). Since this project required that 16 different functions be sent (1 signal for controlling each of the 11 individual signal light lights plus one signal for each of the five functions), the team would have been unable to meet requirements using an entire byte per function. Bitwise manipulation of the data frame was the chosen solution to this issue. With the ability to set a single bit in the data frame being sent (line 91 of the code for the ACTIVATE node), the user could send all 16 signals using just 2 bytes of data. The required data frame was sufficiently small for sending data via a CAN frame. Table 1 shows the bits sent and the location of the bits when a button is pressed.

Table 1

Location	sdata Array Position	Name of Variable	Port on Node
Front Right Blinker	sdata[0], bit 0	RB_F	PORTC.F0
Front Left Blinker	sdata[0], bit 1	LB_F	PORTC.F1
Side Right Blinker	sdata[0], bit 2	RB_S	PORTC.F2
Side Left Blinker	sdata[0], bit 3	LB_S	PORTC.F3
Rear Right Blinker	sdata[0], bit 4	RB_R	PORTC.F4
Rear Left Blinker	sdata[0], bit 5	LB_R	PORTC.F5
Right Brake	sdata[0], bit 6	Br_R	PORTC.F6
Left Brake	sdata[0], bit 7	Br_L	PORTC.F7
Center Brake	sdata[1], bit 0	Br_C	PORTB.F0
Right Headlight	sdata[1], bit 1	HI_R	PORTB.F1
Left Headlight	sdata[1], bit 2	HI_L	PORTB.F4
Function			
Brake	sdata[1], bit 3	BRAKE	PORTB.F7
Left Blinker	sdata[1], bit 4	L_BLINKER	PORTB.F6
Right Blinker	sdata[1], bit 5	R_BLINKER	PORTB.F5
Emergency	sdata[1], bit 6	EMERGENCY	PORTA.F4
Headlight	sdata[1], bit 7	HEADLIGHTS	PORTA.F6

For the purpose of checking the individual bits received by the nodes of the signaling lights, another line in the code was required. Line 277 of the Front Right Blinker code shows how to check a single bit. To check a different bit, the value of x at the beginning of the Front Right Blinker code would be changed. y and z on the code indicate which bit to check for the

input of the “right blinker” and “emergency flashers” functionality. For the Front Right Blinker node, sdata[0], bit 0 should turn on the light, until the button is released. The node must also respond to the Right Blinker (sdata[1], bit 5) input and the Emergency input (sdata[1], bit 6). If, for some reason, multiple inputs are sensed, the node must determine how to function. Lines 286, 293, 297 and 299 of the Front Right Blinker node’s code display this logic. Figure 26 shows a diagram of the logic implemented. In the case of the front right blinker, the blink function takes priority if multiple inputs were sensed.

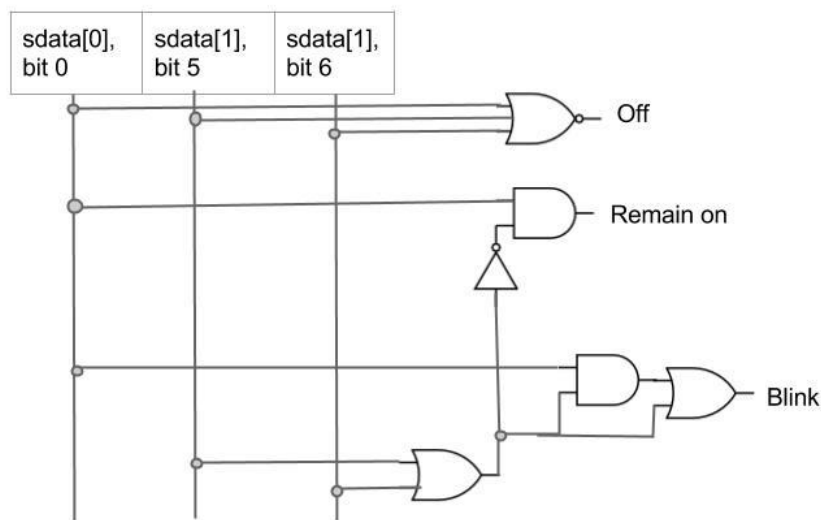


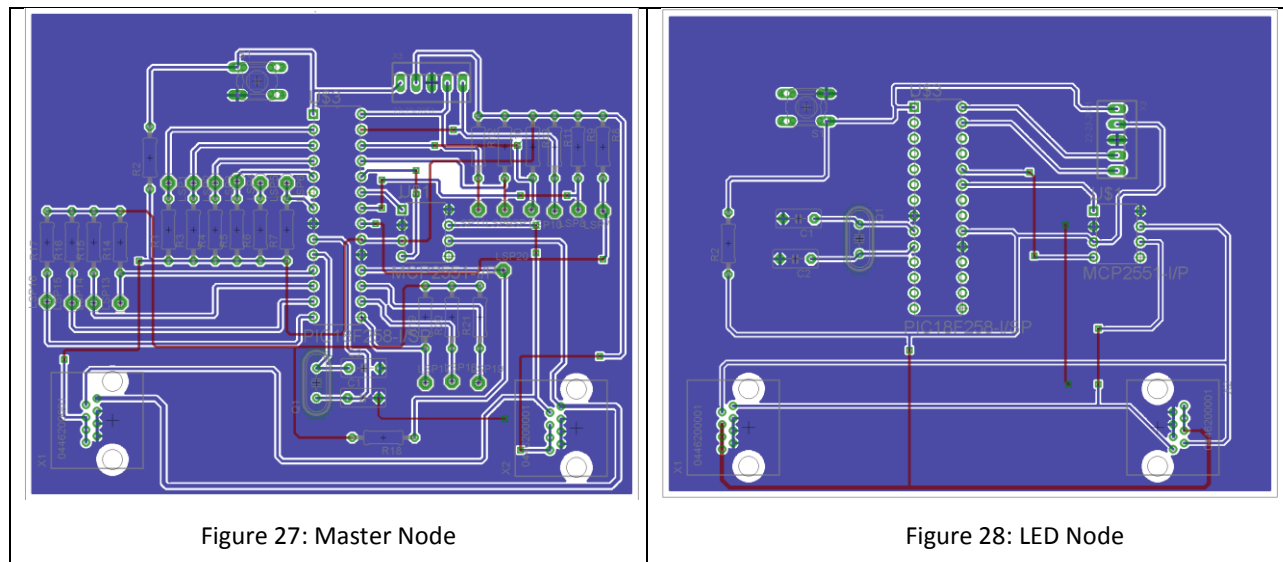
Figure 26 - Functionality of the Front Right Blinker node

Each node had different functionality based on the buttons pressed. The code for each node is in the appendix of this document. Functionality of the code proved to be successful and the team could focus on the production of the physical nodes

Chapter 4

This chapter will focus on how team 12 tested the different design iterations.

As previously mentioned in the hardware section of chapter three, team 12 went through many different hardware components that were all tested at various points in the design. Testing started at a protoboard level and then moved to a solder board. Two printed circuit boards were designed using EAGLE CAD. One of the boards served as a transmitting node and the other was a receiving node. However, due to the redesign there was insufficient time to produce 12 PCB by May 1st. Figure 27 and Figure 28 shows the PCB layouts. The progression of the testing plan can be seen in the following figures below.



The first design involved the use of surface mount components and breakout boards. Figure 29 shows a picture of a surface mount CAN Controller on a breakout board. The surface mount components were used because the sponsor suggested them. This initial design had a separate CAN controller and microcontroller. This design proved to be difficult to implement because communication between the microcontroller and the CAN controller could not be established.

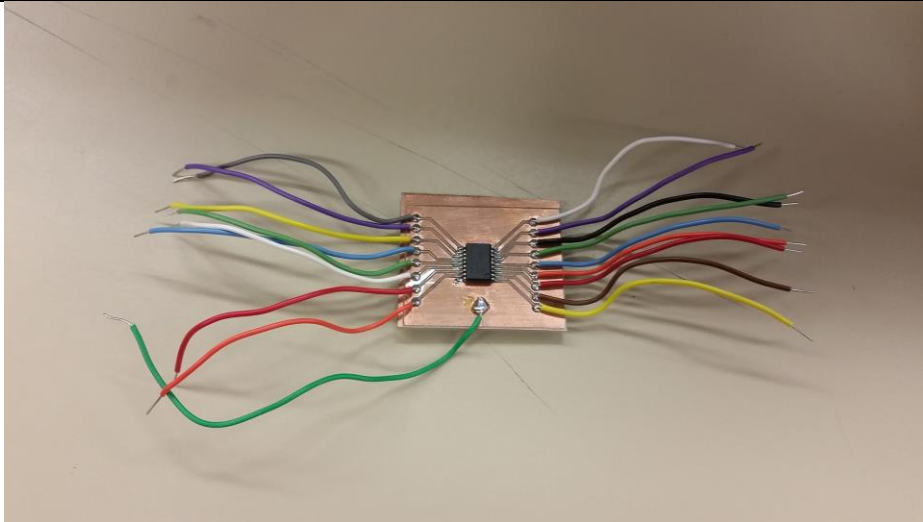


Figure 29: Surface mount CAN controller on breakout board

After the design of CAN nodes changed and the MSP430 microcontroller was replaced with the PIC18F, it was decided that for prototyping purposes dual in line packages would be a better choice. The change to dual in line packages allowed prototyping without making or using breakout boards. The “CANakit” kit was used to program the initial prototype. Figure 30 show the first design iteration using the CANAKit; the first functioning prototype produced. The main challenge in this design lay in the software since the Canakit PIC USB programmer is not directly compatible with MikroElektronika’s MikroC Pro for PIC software. The main problem with this design was the interference from the transients that existed in the protoboard due to the capacitance of the holes. This caused the LED’s to malfunction by turning on and off when it was not necessary.

It was decided that the best way to deal with the transients was to move to solder boards as shown in Figure 31. This decision allowed the transients to be worked out of the design. However, the solder boards made the wiring of the component more complicated. At this point it was decided that the best way to deal with this was to design a printed circuit board that were previously presented.

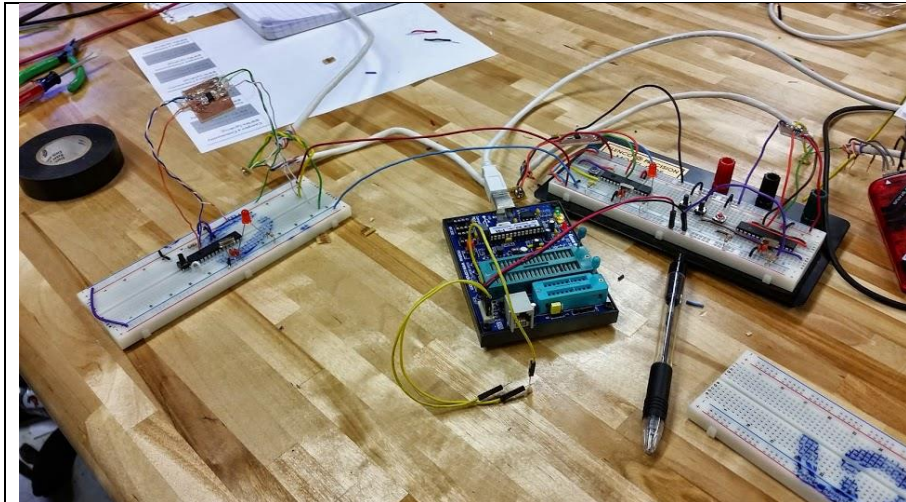


Figure 30: Initial CAN network using CANakit

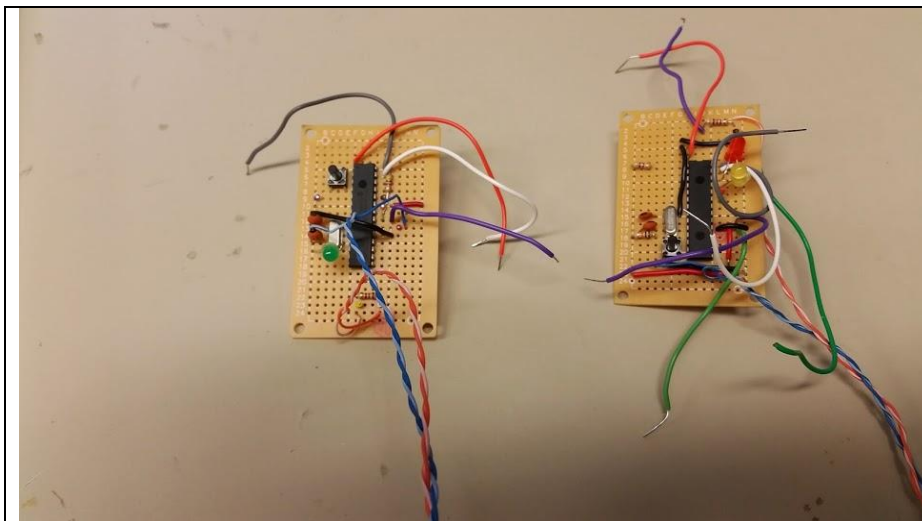


Figure 31: Solder Board

Figure 32 shows a PCB that was cut in the MSU ECE shop located on the third floor of the engineering building. It takes a considerable amount to make and solder each board so it was decided that this was not the best way to present the project at design day.

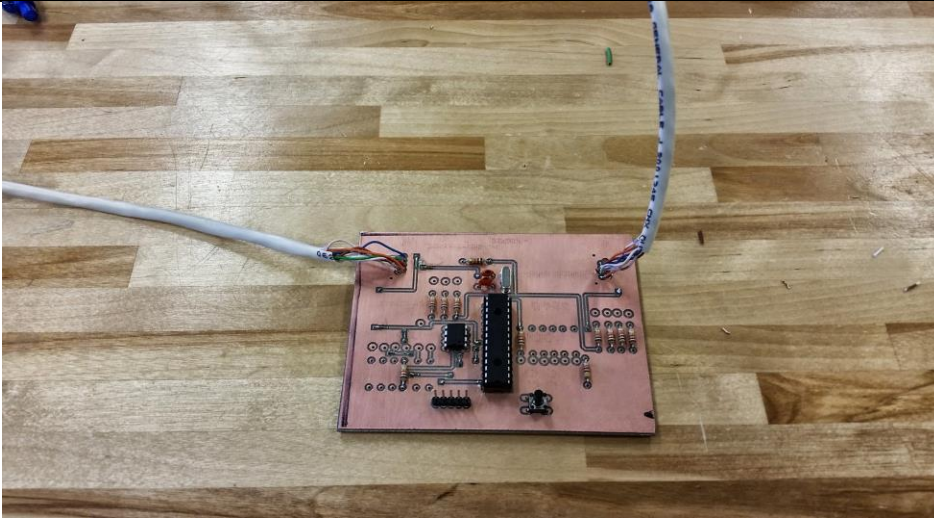


Figure 32: Printed Circuit boards with placed components

Various tests were done to ensure the project was functioning correctly. The first tests involved ensuring that the integrated circuits had the correct voltage applied to them to perform properly. Afterward, a master node was built on a breadboard to develop the “Hello World” program discussed earlier. After the “Hello World” program was developed the team went ahead and developed a CAN bus that was able to transmit data to two different nodes. Using a push-button the master node was able to send signals to specific nodes. When the corresponding node received the message it output a voltage on one of its pins and light an LED that is connected to ground.

At this point measurements of the signals sent onto the CAN bus were taken. The next two figures, Figures 33 and 34, are oscilloscope captures of the different data-frames that the transmitting node send out on the bus depending on which push-button was pushed.



Figure 33: Node One Data Signal



Figure 34: Node Two Data Signal

When zoomed out on the horizontal axis on the oscilloscope to $166\mu\text{s}$, one can see the data-frame being repeatedly sent on the bus every 100ms. When zoomed in on one of the spikes, a CAN data-frame just like the ones in Figures 34 and 35 can be seen.

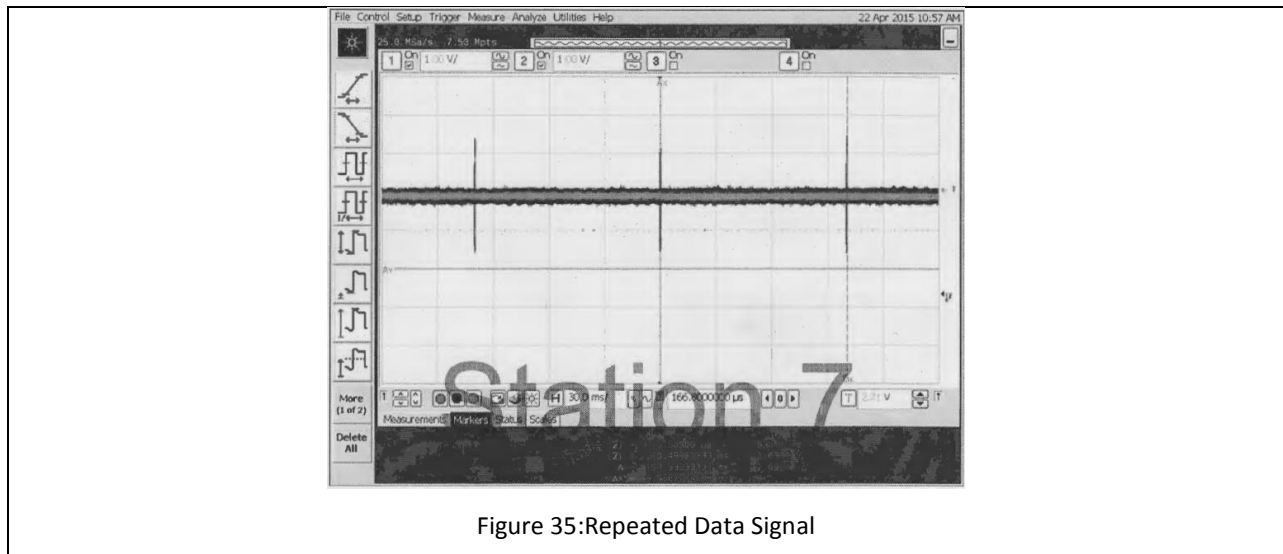


Figure 35: Repeated Data Signal

To power the sensitive devices a regulated 5V supply was necessary. The solar car operates off a standard 12V car battery, so a buck converter was built and tested for functionality. The team wanted to test the switch mode supply under worst case scenario conditions or double battery voltage. The buck converter is rated for a maximum of 42V

operation input voltage (Figure 38); therefore testing at 24V is well within the devices range of voltages. The switch mode was built on a solder board, as seen in Figure 36. The board was then taken to the ECE480 lab where it was connected to a bench power supply outputting 24V and an oscilloscope to ensure the converter was indeed outputting the expected voltage. The oscilloscope capture seen in Figure 37, shows the bench supply waveform outputting 24V with a peak-to-peak ripple voltage 222.6mV on top and the switch mode outputting 5V with a peak-to-peak ripple voltage of 145.2mV on bottom. To simulate the converter under voltage to get the worst case measurement, a 10 Watt, 10 ohm resistor was used (8).

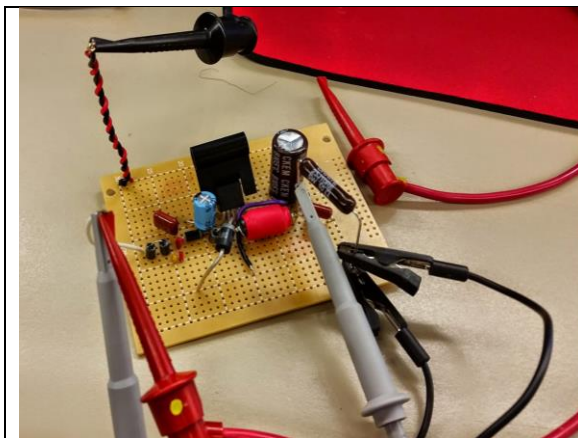


Figure 36: Switch Mode Power Supply

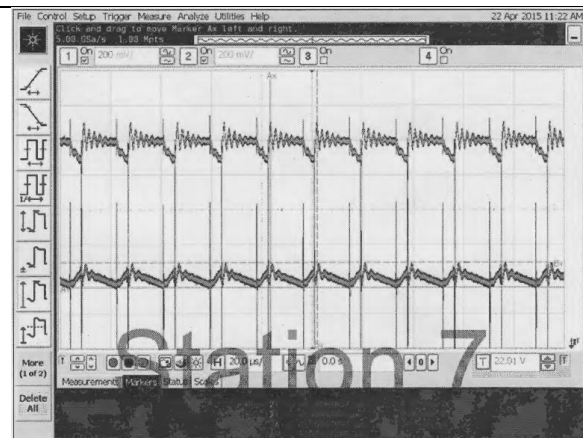


Figure 37: Power Out Ripple Voltage

Operating Junction Temperature Range ⁽¹⁾	-40°C to +125°C
Storage Temperature	-65°C to +150°C
Input Voltage V_{IN}	4.5V to 42V
SW Voltage	Up to 42V

Figure 38: LMR14206 Operational Voltage

Chapter 5:

This chapter has single production cost, mass production cost, team member tasks, future work considerations and a final overview conclusion of the design and experience.

Each group member had specific tasks to complete to ensure success of the design project. The team created a personal task table at the beginning of the semester to make clear what every member had to do to complete a functional product see Table 2 below. The largest success of the project was the fact that after two designs and extremely hard work, a functional CAN lighting system was produced.

The largest failure of the project was the inability to complete the first design. This failure came about from a lack of understanding on the teams part about implementing the CAN protocol from scratch. This lack of understanding was a result of not researching a diverse knowledge base or finding someone experienced working with CAN.

Team 12 learned many different lessons while designing and building the CAN blinker system that can be passed on to future groups. Future teams should know to get a working prototype of the current system before ordering all of the parts. Assuming that the system will work with the ordered parts is a mistake and will result in the loss of money. If a future team decides to implement a different microcontroller, transceiver or configuration, it is a good idea

to get a simple prototype working in a lab environment and then order parts.

Another important note for future teams is to find existing tools available to them regarding the CAN controller. In regards to research; use more than just the internet. Google is a wonderful tool, but it does not know everything. Another lesson was finding a person with first-hand experience in the area of your project is invaluable. Their knowledge of what works and what does not can save an enormous amount of time and frustration.

Team 12 found that creating a CAN system without pre-existing libraries is very challenging and tedious. There is no need for future teams to work on something that already exists and can be used to develop functioning prototypes. The team also found it useful to use existing part libraries when developing the printed circuit boards. Almost all parts have a PCB footprint that somebody has developed.

The cost of building a single circuit is \$10.69. This design is a prototype system and the Solar Car Team only needed one transmitting node and 11 LED nodes. If hypothetically this system was going to be implemented in a mass production scheme for many vehicles the cost would be \$8.86. This is a savings of \$1.83. If one million nodes were manufactured, the company would have a savings of \$1,830,000.

The final budget for team 12 was \$520.18. Going over budget was the result of ordering parts for the first design that were not used and ordering a few extraneous items (the car battery and test lights). If the project was repeated, ordering of parts would not take place until a fully functional prototype was completed.

This experience has been a valuable learning experience for team 12. The largest lessons learned have been in working as a team, clearly communicating ideas, compromising, giving and taking constructive criticism and to look for existing resources. These lessons are important as team 12 enters the workforce as electrical engineers.

Table 2:

Task	Person Responsible
Design of the CAN node circuit, and CAN bus	Eleazar, Kyle, Maxx, Ray, Stephanie
Selection of a switch mode buck power supply	Maxx

Selection of the CAN transceiver and controller	Eleazar
Selection of the LED driver	Stephanie
Selecting resistors and inductors to balance the power usage of the node	Eleazar, Maxx, Stephanie
Wire selection	Eleazar, Stephanie
Negating any inductance produced by the daisy chaining of the nodes	Eleazar, Maxx
Define the CAN frame	Ray, Kyle
Program the microcontroller	Ray, Kyle
Debug and optimize the microcontroller program	Ray, Kyle
Interface the CAN transceiver/CAN controller/microcontroller	Ray, Kyle
Create the hazard light formation	Ray, Kyle
Create the parade light formation	Ray, Kyle

Appendix 1:



Stephanie Stolsky (Electrical Engineering) – The technical elements I was involved with for the solar car project were a hand in the design of the CAN nodes, soldering/constructing nodes and the design and prototyping of the LED driver. The design of the CAN nodes was shared by the entire team. This design process was very interesting since all members of the team were bringing different perspectives into the

discussion. My suggestion for the node design was a switch mode power supply. This suggestion was made by multiple team members. The soldering was a nice change of pace and it improved my soldering skills.

I designed the LED driver and the following was the process. I researched LED's and found that for optimum operation it is the current through the LED that must be constant. This led to research on currently commercially produced constant current drivers and the discovery that those drivers were out of our budget. Next, I considered the easiest way to produce a constant current and the most efficient way to produce a constant current. The most basic way to produce a constant current is to put a resistor in series with voltage source. The most efficient way I could think of producing a constant current, was a pulse width modulated switch mode power supply. The switch mode supply would have required the design and construction of a control system to carry out the pulse width modulation in the presence of any voltage and given the amount of work in this class and on the rest of the project, the control system would take too long. This left the easy approach. I remembered using the LM3317 voltage regulator in class and started researching it. The LM317 could produce 1.5A of current and that was very close to the 1.4A needed by the LED's, so I found a voltage regulator that could produce 3A of current. This regulator was the TI LM350. The design was the input voltage would be connected to the input pin, while the output pin and adjustment pin would be soldered together with a resistor to produce the constant current.



Kyle Grager (Electrical Engineer) - Technical portions of the project were assigned to each member of the group. At first, my technical contribution included just the software portion of the project. In order to properly program the microcontroller, I first had to research the idea behind the CAN protocol. Expanding my knowledge of the entire CAN system and the communication from one node to another on a shared bus was a lot to learn. Having little experience with microcontrollers, just learning how to setup registers on the MSP430G2553 microcontroller was a tough task. I was able to setup the communication, but the team and I decided to switch to a more efficient alternative. We decided to switch to a different microcontroller, which in turn, resulted in different software used.

Once the new microcontroller arrived and we were able to get to work, I immediately started writing code. I first figured out how to upload code to the microcontroller using all of our new tools. The new tools included MikroElektronika MikroC Plus for PIC software, CanaKit USB Programmer for PIC development board, and the book, "Controller Area Network Projects." After uploading code, I verified functionality with some test programs.

With working test programs and microcontrollers, I then built a small prototype and tested it. To do so, I programmed a couple of CAN nodes and verified they could communicate with one another, as mentioned earlier in the report. For the prototype, too, they had to be built. I designed and built a prototype on protoboards.

Once communication via the CAN system was performing, I was able to design the program for each of the nodes. This required a lot of work, as a lot of troubleshooting had to be done. Sometimes the nodes would freeze and require a frequent reset. To resolve this, I included a "break" line within a loop. I also came up with the logic to perform when multiple buttons were pressed.

Finally, I helped solder the individual nodes for the display that will be shown on design day. If all goes well, the final design will look and function wonderfully. Overall, I feel the team worked together well and fulfilled their roles.



Eleazar Gutierrez (Electrical Engineering) - As a member of this team I was extensively involved in the development of the hardware. To begin I looked through data sheets to find components that would meet the system requirements. I was also responsible for ordering discrete components needed to make circuits work.

In the first design I was responsible for soldering wires to the integrated circuits to put them on breadboards. After realizing that the pins were too fragile and would break when prototyping I designed the breakout boards using EAGLE CAD. I responsible for soldering the surface mount components onto the breakout boards.

The team could not get the separated microcontroller, CAN controller and CAN transceiver to communicate and it was decided to switch to a microcontroller with an integrated CAN controller. The same transceiver was kept in the new design. Using a book that was bought by a team member I was tasked with putting together the circuit for the first design. In this first design I was responsible for building a master node that would send signals and a receiver that receives the messages from the master node. If the correct message is received by the receiving node the microcontroller lights up an LED. After this worked properly I was tasked with designing and building the first CAN bus. Signals for the CAN, power for the circuitry and ground were all attached to this bus.

The decision to move from breadboards to solder boards required that I help solder the components and troubleshoot the circuits. After this worked properly I was tasked with designing the printed circuit boards using EAGLE CAD. I was once again tasked with soldering components onto the printed circuit boards once they were manufactured by the MSU ECE shop. Ultimately, it was decided that the boards would take too long to manufacture and it was decided to not go ahead with this plan. Overall, the team had team members that were able to handle all of the major technical challenges. The team worked well and fulfilled their technical duties to a satisfactory extent.



In this project, Raymond Charles Cashen was tasked initially with understanding and researching CAN protocol and systems. I researched into the MSP430G2553 microcontroller, used in the 480 lab portion, in order to use it with a CAN system. This included in-depth research into the registers of the microcontroller in order to set up a SPI connection between the microcontroller and the can controller, and interfacing with the CAN transceiver and the CAN bus. This approach led

to many problems, which in turn led to more research and trial an error.

I contributed to conceptual designs of the code required to operate the CAN system, including structure of the CAN frame and structure of the individual microprocessor code that interperets the CAN frame. There is not specific way to program a frame, so there are many ways to achieve the same goal. This means that every design must be tested or planned out to see if it is the most efficient and effective way to accomplish the design goals. The process of trial and error takes some time.

I helped to design and build the demonstration of the CAN system used during design day, so that it would accurately and completely describe what our design does – more specifically, why it is different than directly connecting each microcontroller to each node. This requires the ends of the CAN bus to be visible, that way it is apparent that it is not a direct-connection system. Every functionality that we require must be testable, so 16 buttons must be accessible to demonstrate each action.

I was tasked with writing the software portion of the final report. This report had to be descriptive enough to accurately describe the project in its entirety, on the software side. This includes a complete description of the process by which a final design was created, which is a lengthy process. It must be written in such a way that it is understandable by anyone, which can be difficult given the nature of this project.



When the project was assigned to team 12, the members clearly saw there was going to be a hardware and software section of the project. Thus we split into groups to attack each issue. Maxx Coral was in the hardware group tasked with the selection of a CAN transceiver, CAN controller, switch mode power supply, and a LED driver for the design project. To make a section of a component that can work in the project required a lot of hours on integrated circuit manufactures websites, looking at the numerous different options they produce and comparing them to the specifications that were given to team 12.

Many datasheets were viewed, and after a couple iterations of selecting parts and ordering free samples, we found chips that would all function correctly together. Teamwork in this selection of components helped two main ways. First, by having second pair of eyes looking over a possible parts helped catch issues early off that work affect functionality.

Having built and tested switch mode power supplies in previous MSU classes, I explained the fundamental theory of operation of a buck-boost switch mode power supply in a technical presentation. I have also built and tested these in lab and which give me real world hands on experience with switch modes. With my background experience it only made sense that I would focus my technical efforts on the hardware section of team 12's design project. I performed efficiency measurements at worse case scenario's for linear and switch mode power supplies. Even though the team already knew from classes and online research that linear regulators were much worse in term of efficiency compared to switch mode supplies, the data still needed to be collected. This data proved that linear regulators were indeed much worse, and the team went on to design a switch mode power supply.

The team classified the hardware tasks as researching hardware components (i.e. CAN controller, CAN transceiver, and switch mode power supply), buying the needed devices and building the breadboard circuits. This were some of the technical tasks I achieved while working on the CAN blinker system.

Appendix 2:

References

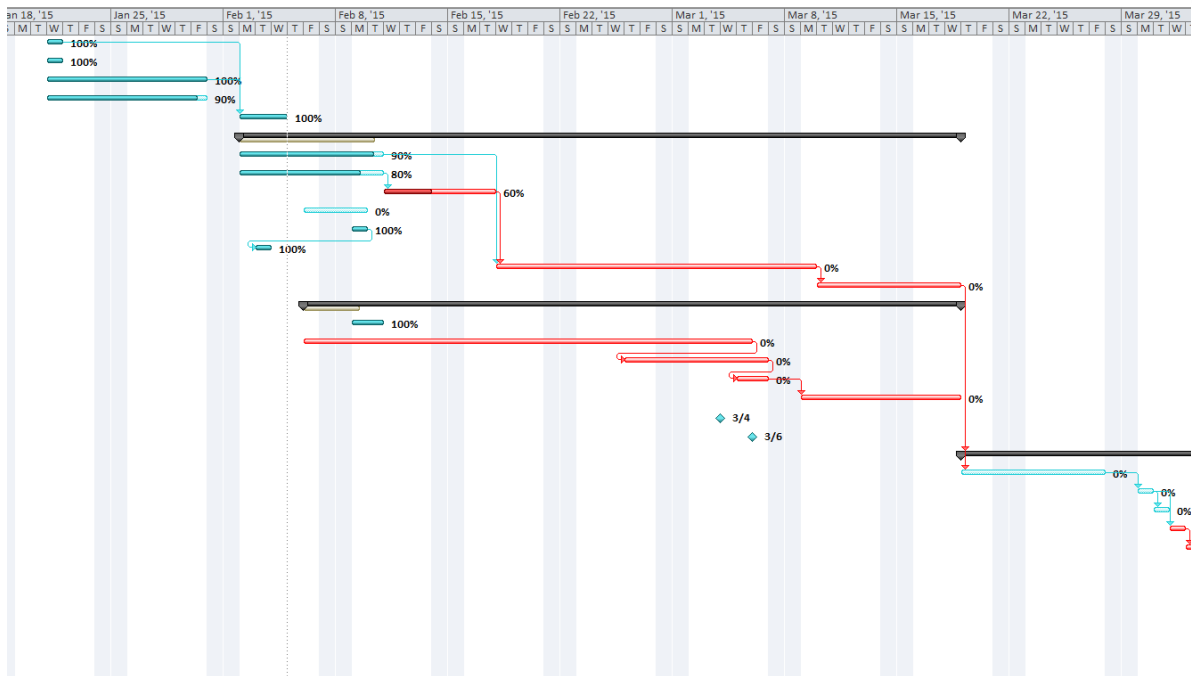
1. "6.2 Lighting." *Formula Sun Revision A*. Formula Sun, 14 Dec. 2014. Web. 4 Feb. 2015. <<http://americansolarchallenge.org/ASC/wp-content/uploads/2014/12/FSGP-2015-External-Revision-A.pdf>>.
2. "Appendix C: Reference Standard for Lighting." *Formula Sun Revision A*. Formula Sun, 14 Dec. 2014. Web. 4 Feb. 2015. <<http://americansolarchallenge.org/ASC/wp-content/uploads/2014/12/FSGP-2015-External-Revision-A.pdf>>.
3. "Controller Area Network (CAN) Overview." *National Instruments*. National Instruments, 1 Aug. 2014. Web. 3 Feb. 2015. <<http://www.ni.com/white-paper/2732/en/>>.
4. "Introduction to the Controller Area Network (CAN)." *Texas Instruments Literature*. Texas Instruments. Web. 4 Feb. 2015. <<http://www.ti.com/lit/an/sloa101a/sloa101a.pdf>>.
5. "PS560200 4.5-V to 17-V Input, 500-mA Synchronous Step-Down SWIFT™ Converter With Advanced Eco-Mode™." *PS560200 4.5-V to 17-V Input, 500-mA Synchronous Step-Down SWIFT™ Converter With Advanced Eco-Mode™*. Texas Instruments, 1 Jan. 2015. Web. 4 Feb. 2015. <<http://www.ti.com/lit/ds/symlink/tps560200.pdf>>.
6. "Time Stamp Generation." *Vector Knowledge Base*. Vector, 28 Jan. 2015. Web. 4 Feb. 2015. <<https://vector.com/kbp/entry/47/>>.
7. <http://www.ti.com/lit/ug/slau144j/slau144j.pdf>
8. <http://www.ti.com/lit/ds/symlink/lmr14206.pdf>
9. <http://ww1.microchip.com/downloads/en/DeviceDoc/21801G.pdf>
10. <http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf>
11. Controller Area Network Projects - Dogan Ibrahim
12. <https://www.fairchildsemi.com/datasheets/LM/LM350.pdf>
13. <http://www.mikroe.com/>
14. <https://www.fairchildsemi.com/datasheets/LM/LM350.pdf>

- [15] Texas Instruments (2013, March). *LM150/LM350A/LM350 3-Amp Adjustable Regulators Datasheet* [online]. Available: <http://www.ti.com/lit/ds/symlink/lm350-n.pdf>
- [16] Wang, Bingsen, "Lec4 : Keep Semiconductors Cool," unpublished.
- [17] D. Ibrahim, *Controller Area Network projects*. United Kingdom: Elektor Int Media Bv, 2011.

Appendix 3

	Duration	Start	Finish	Predecessors	Resource Names
1	1 day	Wed 1/21/15	Wed 1/21/15		S,E,M,R,K
2	1 day	Wed 1/21/15	Wed 1/21/15		E,M
3	8 days	Wed 1/21/15	Fri 1/30/15		E,M
4	8 days	Wed 1/21/15	Fri 1/30/15		S,K,R
5	3 days	Mon 2/2/15	Wed 2/4/15	1,3	S,M,E
6	33 days	Mon 2/2/15	Wed 3/18/15		E,K,M,R,S
7	7 days	Mon 2/2/15	Tue 2/10/15		E,M
8	7 days	Mon 2/2/15	Tue 2/10/15		S,E,M
9	5 days	Wed 2/11/15	Tue 2/17/15	8	M
10	2 days	Fri 2/6/15	Mon 2/9/15		E
11	1 day	Mon 2/9/15	Mon 2/9/15		S
12	1 day	Tue 2/3/15	Tue 2/3/15	11	S
13	14 days	Wed 2/18/15	Mon 3/9/15	7,9	E,K,M,R,S
14	7 days	Tue 3/10/15	Wed 3/18/15	13	S,E,M
15	29 days	Fri 2/6/15	Wed 3/18/15		K,R
16	2 days	Mon 2/9/15	Tue 2/10/15		
17	20 days	Fri 2/6/15	Thu 3/5/15		R,K
18	7 days	Thu 2/26/15	Fri 3/6/15	17	R,K
19	2 days	Thu 3/5/15	Fri 3/6/15	18	R,K
20	8 days	Mon 3/9/15	Wed 3/18/15	19	R,K
21	0 days	Wed 3/4/15	Wed 3/4/15		
22	0 days	Fri 3/6/15	Fri 3/6/15		
23	11 days	Thu 3/19/15	Thu 4/2/15	14	E,K,M,R,S
24	7 days	Thu 3/19/15	Fri 3/27/15	14	E
25	1 day	Mon 3/30/15	Mon 3/30/15	24	S
26	1 day	Tue 3/31/15	Tue 3/31/15	25	M
27	1 day	Wed 4/1/15	Wed 4/1/15	25	K
28	1 day	Thu 4/2/15	Thu 4/2/15	27	R
29	10 days	Fri 4/3/15	Thu 4/16/15	28	E,K,M,R,S
30	1 day	Fri 4/3/15	Fri 4/3/15	28	K
31	5 days	Fri 4/3/15	Thu 4/9/15	28	R
32	4 days	Fri 4/3/15	Wed 4/8/15	28	S
33	5 days	Fri 4/10/15	Thu 4/16/15	30,31,32	E
34	1 day	Fri 4/17/15	Fri 4/17/15	33	E,K,M,R,S

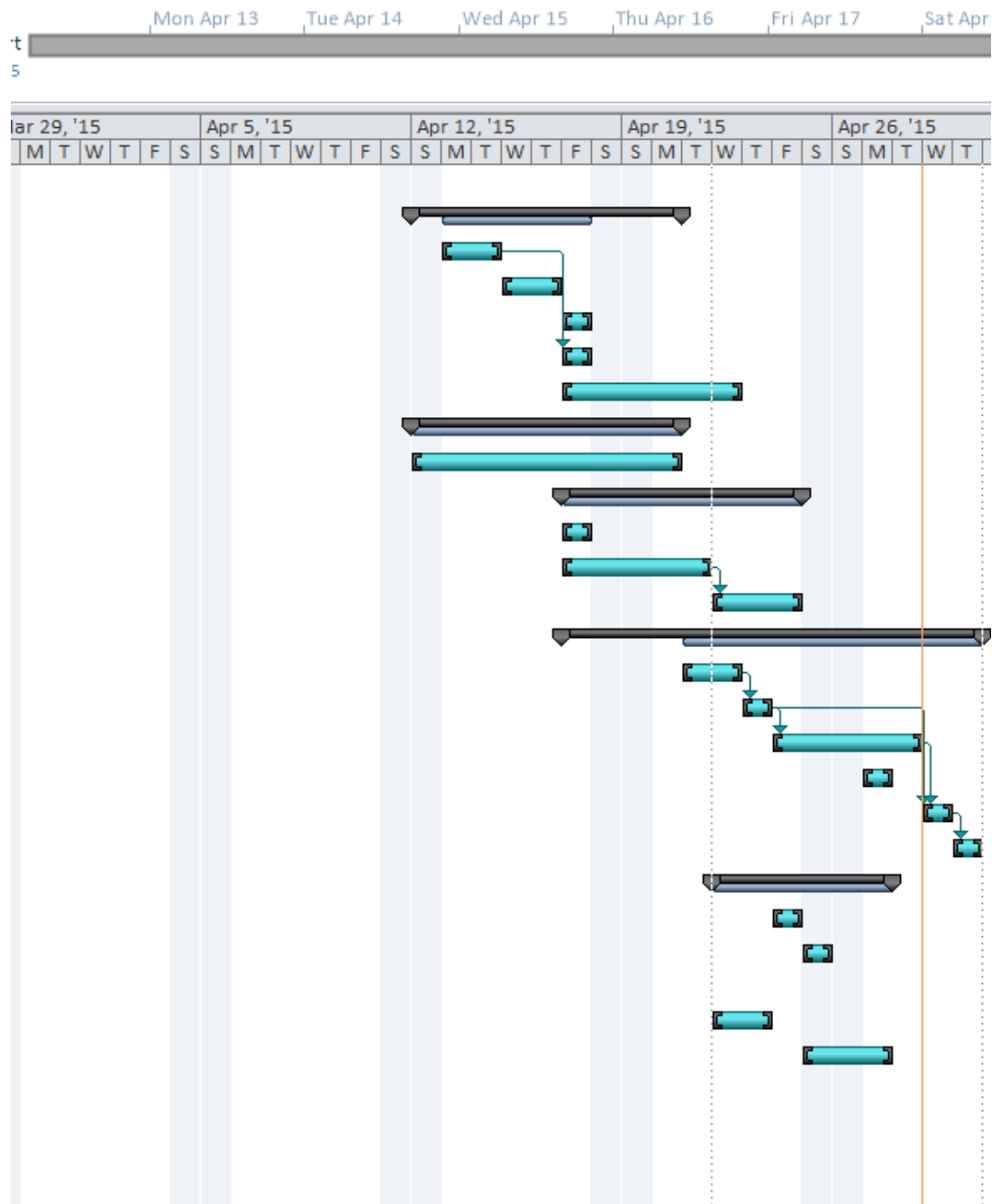
Gantt Chart 1: Schedule



Gantt Chart 1: Graph

		Task Mode ▾	Task Name ▾	Duration ▾	Start ▾	Finish ▾
1						
2			Hardware Prototyping	7 days	Sun 4/12/15	Mon 4/20/15
3			Ordering New Mirocontrollers	2 days	Mon 4/13/15	Tue 4/14/15
4			Ordering More CAN Transceivers	2 days	Wed 4/15/15	Thu 4/16/15
5			Make the CAN Bus	1 day	Fri 4/17/15	Fri 4/17/15
6			Build the breadboard	1 day	Fri 4/17/15	Fri 4/17/15
7			Ordering RJ45 Connectors	4 days	Fri 4/17/15	Wed 4/22/15
8			Software Programming	7 days	Sun 4/12/15	Mon 4/20/15
9			Code/Debug the CAN Nodes	7 days	Sun 4/12/15	Mon 4/20/15
10			Intergration and Testing	6 days	Fri 4/17/15	Fri 4/24/15
11			Program the microcontroller	1 day	Fri 4/17/15	Fri 4/17/15
12			Test/Troubleshoot the Nodes	3 days	Fri 4/17/15	Tue 4/21/15
13			Test/Troubleshoot the System	3 days	Wed 4/22/15	Fri 4/24/15
14			Final Design	10 days	Fri 4/17/15	Thu 4/30/15
15			Make the PCB layouts	2 days	Tue 4/21/15	Wed 4/22/15
16			Cut the PCB's	1 day	Thu 4/23/15	Thu 4/23/15
17			Populate the PCB's	3 days	Fri 4/24/15	Tue 4/28/15
18			Make the final CAN bus	1 day	Mon 4/27/15	Mon 4/27/15
19			Test the final nodes	1 day	Wed 4/29/15	Wed 4/29/15
20			Test the final system	1 day	Thu 4/30/15	Thu 4/30/15
21			Design Day Tasks	4 days	Wed 4/22/15	Mon 4/27/15
22			Print/mount the demo poster	1 day	Fri 4/24/15	Fri 4/24/15
23			Wire the demo poster and add LED's	1 day	Sat 4/25/15	Sat 4/25/15
24			Make the LED controller box	2 days	Wed 4/22/15	Thu 4/23/15
25			Make the desgin day poster	2 days	Sat 4/25/15	Mon 4/27/15

Gantt Chart 2: Schedule



Gantt Chart 2: Graph

```

1  /*****
2  *****/
3      CONTROL ENTIRE BUS FOR DISPLAY ON DESIGN DAY
4  *****/
5
6  void main() {
7      unsigned char RB_F, LB_F, RB_S, LB_S, RB_R, LB_R, Br_R, Br_L, Br_C,
8  Hl_R,
9      Hl_L, BRAKE, L_BLINKER, R_BLINKER, EMERGENCY, HEADLIGHTS, sdata[8];
10     unsigned short config_flag, send_flag;
11     char SJW, BRP, Phase_Seg1, Phase_Seg2, Prop_Seg;
12     long LED_ID, ACTIVATE_ID, mask;
13
14     //Arbitration (Activation) IDs of Nodes
15     LED_ID = 0x01;           //Message ID of node LEDs
16     ACTIVATE_ID = 0x00;      //Message ID of node ACTIVATE
17
18     //Configure port directions
19     TRISC = 0xFF;             //All of PORTC uses switch inputs
20     TRISB = 0xFB;             //RB2 is output(CANTX), RB3 is
21 input(CANRX)
22     TRISA = 0xFF;             //RA4 is input
23
24     //CAN BUS timing parameters
25     SJW = 3;
26     BRP = 8;
27     Phase_Seg1 = 3;
28     Phase_Seg2 = 3;
29     Prop_Seg = 1;
30     sdata[0] = 0x00;
31
32     //Configuration
33
34     config_flag = _CAN_CONFIG_SAMPLE_THRICE           &
35                  _CAN_CONFIG_PHSEG2_PRG_ON           &
36                  _CAN_CONFIG_STD_MSG                 &
37                  _CAN_CONFIG_DBL_BUFFER_ON            &
38                  _CAN_CONFIG_VALID_STD_MSG            &
39                  _CAN_CONFIG_LINE_FILTER_OFF;
40
41     send_flag = _CAN_TX_PRIORITY_0                     &
42                _CAN_TX_STD_FRAME                       &
43                _CAN_TX_NO_RTR_FRAME;
44
45     //Initialize CAN module
46
47     CANInitialize(SJW, BRP, Phase_Seg1, Phase_Seg2, Prop_Seg, config_flag);
48
49     //Set CAN CONFIG Mode
50
51     CANSetOperationMode(_CAN_MODE_CONFIG, 0xFF);
52
53     mask = -1;
54
55     //Set all MASK1 bits to 1's
56     CANSetMask(_CAN_MASK_B1, mask, _CAN_CONFIG_STD_MSG);
57

```

```

58 //Set all MASK2 bits to 1's
59 CANSetMask(_CAN_MASK_B2, mask, _CAN_CONFIG_STD_MSG);
60
61 //Set ID filter of B2_F3 to 3 (ACTIVATE_ID)
62 CANSetFilter(_CAN_FILTER_B2_F3, ACTIVATE_ID, _CAN_CONFIG_STD_MSG);
63
64 //Set CAN module to NORMAL mode (from config mode)
65 CANSetOperationMode(_CAN_MODE_NORMAL, 0xFF);
66
67 //sense for button, then send data when the button is sensed
68 for(;;)
69 {
70 //Define input switches
71 RB_F = PORTC.F0;
72 LB_F = PORTC.F1;
73 RB_S = PORTC.F2;
74 LB_S = PORTC.F3;
75 RB_R = PORTC.F4;
76 LB_R = PORTC.F5;
77 Br_R = PORTC.F6;
78 Br_L = PORTC.F7;
79 Br_C = PORTB.F0;
80 Hl_R = PORTB.F1;
81 Hl_L = PORTB.F4;
82 BRAKE = PORTB.F7;
83 L_BLINKER = PORTB.F6;
84 R_BLINKER = PORTB.F5;
85 EMERGENCY = PORTA.F4;
86 HEADLIGHTS = PORTA.F1;
87 /*-----
88 Front Right Blinker
89 -----*/
90 if(RB_F != 0)
91     sdata[0] &= ~(1 << 0);           //If not pressed - CLEAR BIT 0
92 else
93     sdata[0] |= 1 << 0;             //If pressed - SET BIT 0
94 /*-----
95 Front Left Blinker
96 -----*/
97 if(LB_F != 0)
98     sdata[0] &= ~(1 << 1);           //If not pressed - CLEAR BIT 1
99 else
100     sdata[0] |= 1 << 1;             //If pressed - SET BIT 1
101 /*-----
102 Side Right Blinker
103 -----*/
104 if(RB_S != 0)
105     sdata[0] &= ~(1 << 2);           //If not pressed - CLEAR BIT
106 else
107     sdata[0] |= 1 << 2;             //If pressed - SET BIT
108 /*-----
109 Side Left Blinker
110 -----*/
111 if(LB_S != 0)
112     sdata[0] &= ~(1 << 3);           //If not pressed - CLEAR BIT
113 else

```

```

114         sdata[0] |= 1 << 3;           //If pressed - SET
115 BIT
116 /*-----
117 Rear Right Blinker
118 -----*/
119 if(RB_R != 0)
120     sdata[0] &= ~(1 << 4);           //If not pressed - CLEAR BIT
121 else
122     sdata[0] |= 1 << 4;           //If pressed - SET BIT
123 /*-----
124 Rear Left Blinker
125 -----*/
126 if(LB_R != 0)
127     sdata[0] &= ~(1 << 5);           //If not pressed - CLEAR BIT
128 else
129     sdata[0] |= 1 << 5;           //If pressed - SET BIT
130 /*-----
131 Right Brake
132 -----*/
133 if(Br_R != 0)
134     sdata[0] &= ~(1 << 6);           //If not pressed - CLEAR BIT
135 else
136     sdata[0] |= 1 << 6;           //If pressed - SET BIT
137 /*-----
138 Left Brake
139 -----*/
140 if(Br_L != 0)
141     sdata[0] &= ~(1 << 7);           //If not pressed - CLEAR BIT
142 else
143     sdata[0] |= 1 << 7;           //If pressed - SET BIT
144 /*-----
145 Center Brake
146 -----*/
147 if(Br_C != 0)
148     sdata[1] &= ~(1 << 0);           //If not pressed - CLEAR BIT
149 else
150     sdata[1] |= 1 << 0;           //If pressed - SET BIT
151 /*-----
152 Right Headlight
153 -----*/
154 if(Hl_R != 0)
155     sdata[1] &= ~(1 << 1);           //If not pressed - CLEAR BIT
156 else
157     sdata[1] |= 1 << 1;           //If pressed - SET BIT
158 /*-----
159 Left Headlight
160 -----*/
161 if(Hl_L != 0)
162     sdata[1] &= ~(1 << 2);           //If not pressed - CLEAR BIT
163 else
164     sdata[1] |= 1 << 2;           //If pressed - SET BIT
165 /*-----Below are functions that will trigger multiple LEDs at once-----
166 -*/
167 /*-----
168 Brake
169 -----*/

```

```

170     if(BRAKE != 0)
171         sdata[1] &= ~(1 << 3);           //If not pressed - CLEAR BIT
172     else
173         sdata[1] |= 1 << 3;             //If pressed - SET BIT
174     /*-----
175     Left Blinker
176     -----*/
177     if(L_BLINKER != 0)
178         sdata[1] &= ~(1 << 4);           //If not pressed - CLEAR BIT
179     else
180         sdata[1] |= 1 << 4;             //If pressed - SET BIT
181     /*-----
182     Right Blinker
183     -----*/
184     if(R_BLINKER != 0)
185         sdata[1] &= ~(1 << 5);           //If not pressed - CLEAR BIT
186     else
187         sdata[1] |= 1 << 5;             //If pressed - SET BIT
188     /*-----
189     EMERGENCY FLASHERS
190     -----*/
191     if(EMERGENCY != 0)
192         sdata[1] &= ~(1 << 6);           //If not pressed - CLEAR BIT
193     else
194         sdata[1] |= 1 << 6;             //If pressed - SET BIT
195     /*-----
196     Headlights
197     -----*/
198     if(HEADLIGHTS != 0)
199         sdata[1] &= ~(1 << 7);           //If not pressed - CLEAR BIT
200     else
201         sdata[1] |= 1 << 7;             //If pressed - SET BIT
202
203
204     //Send button state in data frame over the CAN bus
205     CANWrite(LED_ID, sdata, 2, send_flag); //Send button state
206     Delay_Ms(100);
207 }
208
209 }
210

```

```

211  /*****
212      FRONT RIGHT BLINKER
213  *****/
214
215  void main() {
216      unsigned char push_button, read_flag, rdata[8],
217      x, y, z, light, blinker, emergency, flash;
218      unsigned short config_flag, len;
219      char SJW, BRP, Phase_Seg1, Phase_Seg2, Prop_Seg;
220      long LED_ID, ACTIVATE_ID, id, mask;
221
222      //Message identifiers of nodes
223      LED_ID = 0x01;           //Message ID of node LED
224      ACTIVATE_ID = 0x00;      //Message ID of node ACTIVATE
225      x = 0;                   //Bit in string of rdata[0] to check for light
226  on
227      y = 5;                   //Bit in string of rdata[1] to check for
228  blinker
229      z = 6;                   //Bit in string of rdata[1] to check for
230  emergency
231
232      //Configure port directions
233      TRISC = 0;               //RC0 is output (LED Port)
234      TRISB = 0x08;           //RB2 is output, RB3 is input
235
236      //CAN bus timing parameters
237      SJW = 3;
238      BRP = 8;
239      Phase_Seg1 = 3;
240      Phase_Seg2 = 3;
241      Prop_Seg = 1;
242
243      //Configuration
244
245      config_flag = _CAN_CONFIG_SAMPLE_THRICE &
246                  _CAN_CONFIG_PHSEG2_PRG_ON &
247                  _CAN_CONFIG_STD_MSG &
248                  _CAN_CONFIG_DBL_BUFFER_ON &
249                  _CAN_CONFIG_VALID_STD_MSG &
250                  _CAN_CONFIG_LINE_FILTER_OFF;
251      read_flag = 0;
252
253      //Initialize CAN module
254
255      CANInitialize(SJW, BRP, Phase_Seg1, Phase_Seg2, Prop_Seg, config_flag);
256
257      //Seg CAN CONFIG mode
258      CANSetOperationMode(_CAN_MODE_CONFIG, 0xFF);
259
260      mask = -1;
261
262      //Set all MASK1 bits to 1's
263      CANSetMask(_CAN_MASK_B1, mask, _CAN_CONFIG_STD_MSG);
264
265      //Set all MASK2 bits to 1's
266      CANSetMask(_CAN_MASK_B2, mask, _CAN_CONFIG_STD_MSG);
  
```

```

267
268 //Set ID of filter B2_F3
269 CANSetFilter(_CAN_FILTER_B2_F3, LED_ID, _CAN_CONFIG_STD_MSG);
270
271 //Set CAN module to NORMAL mode
272 CANSetOperationMode(_CAN_MODE_NORMAL, 0xFF);
273 PORTC.F0=0;
274 for(;;){
275 //Constantly read for CAN Signal within the data frame
276 read_flag = CANRead(&id, rdata, &len, &read_flag);
277 light = ((rdata[0] >> x) & 1) ; //Light solid ON
278 blinker = ((rdata[1] >> y) & 1); //If blinker is pressed
279 emergency = ((rdata[1] >> z) & 1); //If emergency is pressed
280 flash = blinker | emergency; //Light should flash if both
281 buttons are pressed
282
283 //Check the message ID just in case.
284 if(read_flag != 0 && id == LED_ID)
285 {
286 if((light & flash) | flash) //Light and Flash are both pressed OR
287 Flash only forces the light to go into a flashing loop
288 {
289 PORTC.F0 = 1;
290 Delay_Ms(1000);
291 PORTC.F0 = 0;
292 Delay_Ms(1000);
293 if(~((light & flash) | flash))
294 break; //For some reason, the uC required less resets after
295 including this line
296 }
297 else if(~flash & light)
298 PORTC.F0 = 1;
299 else
300 PORTC.F0 = 0;
301 }
302 }
303
304 Delay_Ms(100);
305 }
306

```