

Rapid Prototyping Environment for Climate Control Development

Final Report

4/25/2014

Michigan State University
Senior Design - ECE 480 - Team 4

Project Sponsor:
General Motors

Project Facilitator:
Lixin Dong

Team Members:
Guiseppe Ferro
Omar Ali
Weiham Yan
Sitong Ge
Ricardo Johnson
Yingshuyu Li



Executive Summary

General Motors is in need of Rapid Control Prototyping (RCP) Local Interconnect Network (LIN) Driver libraries that are capable of measuring the vehicles environmental characteristics from sensors and operating several actuators used to modulate the climate of the vehicle interior compartment. These LIN drivers are to utilize the dSPACE RCP system which can be used to conduct strain tests and simulations on the LIN based sensors and actuators. The dSPACE system makes it possible to develop a reusable driver library that GM requires.

This design team has taken on the task of developing dSPACE libraries that are capable of reading the temperature, ambient light, solar intensity, solar angles, and the humidity level of the sensors surrounding environment. Also, a driver library was designed to control actuators that are used to open and close vents in the climate control system. dSPACE provides MATLAB/Simulink LIN block sets that were used to setup communication between the vehicles Electrical Control Unit (ECU) and the sensors and actuators.

The impact of the Simulink and dSPACE driver libraries developed will set the foundation for rapid control prototyping using LIN based sensors and actuators.

Acknowledgements

The following individuals and companies deserve special recognition for their contributions to the success of our project.

MSU Staff:

- Dr. Lixin Dong - Lixin was team facilitator. He provided the team with feedback on the team's proposal, presentations, design concepts ideas, and progress reports. Lixin also helped to keep the team on track throughout the semester regarding the due dates of the assignments due.
- Greg Mulder and Brian Wright - These two individuals provided basic tools such as a power supply and tools to aid in the completion of the project.

General Motors Staff:

- Andrew Herman – Andrew was the main supervisor and overseer of our project. Andrew also setup our site visit to GM for our project demonstration
- Elliot Keen – Elliot was the main project lead and our first point of contact for direction and issue resolution. Elliot also provided the team with the necessary hardware to be used in the project such as the MicroAutoBox, sensors, actuators, and LIN wire harnesses.
- Todd Tumas – Todd helped with any questions or problems the team had with the sensors. Todd also provided specifications on sensors.
- Gilles Delormes – Gilles provided the LIN Description Files (LDF) to be used in this project. Gilles also provided helpful information and guidance on LIN protocol and functionality.
- Nedra Johnson - Nedra provided actuator specifications and help with issues in setting up proper actuator connection.

dSPACE Staff:

- Chloe Pasque - Chloe provided the team with the complete dSPACE software package. Chloe also made several trips to the MSU campus to see how the team was doing and provide any assistance and conflict resolution with the dSPACE product.
- Matthew O'Connor - Matthew provided guidance in the theory of LIN communication through the Simulink block sets. Matthew also assisted in troubleshooting any problems the team had using the dSPACE LIN block sets as well as the Control Desk software.

Table of Contents

Chapter 1: Introduction and Background	3
1.1: Introduction	3
1.2: Background	4
1.2.1: Local Interconnect Network (LIN)	4
1.2.2: dSPACE	6
1.2.3: HIL Simulation	6
1.2.4: MicroAutoBox 1401/1501	6
1.2.5: Sensors	7
1.2.6: Actuators	8
1.2.7: Integration in a Vehicle	9
Chapter 2: Solution Analysis and Approach	11
2.1: Fast Diagram	11
2.2: Critical Customer Requirements	12
2.3: Conceptual Designs	12
2.4: Ranking of Conceptual Designs	14
2.5: Final Design Decision	14
2.6: Budget	15
2.7: Gantt Chart	16
Chapter 3: Technical Description of Work Performed	16
3.1: Hardware Implementation	16
3.1.1: Setting up the MicroAutoBox and dSPACE Software	16
3.1.2: Switching on the MicroAutoBox	17
3.1.3: Power Supply Requirements	18
3.1.4: Setting MicroAutoBox as the LIN Master	18
3.1.5: Connecting MicroAutoBox to PC	18
3.1.6: Connecting the ILSS to the MicroAutoBox via LIN	19
3.1.7: Connecting the RLHS to the MicroAutoBox via LIN	20
3.1.8: Connecting the Actuators to the MicroAutoBox via LIN	20
3.2: Software Implementation	21
3.2.1: LDF Programming	22
3.2.2: ILSS LIN Driver	23
3.2.3: RLHS LIN Driver	28
3.2.4: Actuator LIN Driver	33
Chapter 4: Test Data	37
4.1: Testing of the ILSS	37
4.2: Testing of the RLHS	41
4.3: Testing of the Actuators	45
Chapter 5: Summary	48
5.1: Summary	48
5.2: Successes and Failures	48
5.3: Suggestions for Future Improvement	48
5.4: Final Budget	49
Appendix 1: Technical Roles	50
Appendix 2: References	55
Appendix 3: Running Charts, Gantt, Chart and Beyond	56

Chapter 1: Introduction and Background

1.1: Introduction

With a constant need for new sophisticated technology to meet the consumer's needs, General Motors (GM) climate control systems must be modified in order to stay competitive. GM's climate control system is moving from analog interface actuators and sensors to devices that use a Local Interconnect Network (LIN) serial protocol communication to communicate between the Electrical Control Unit (ECU) and the sensors and actuators. The analog devices used currently send signals through multiple wires, depending on the number of components, and are processed by the ECU. The extra wires add additional cost, weight, and less precision for the overall system due to the delay of ECU processing. With the use of LIN, single wire communication to and from the actuators and sensors provide less cost, weight, and greater accuracy due to intelligent schedules that are predefined in the LIN protocol. Due to these qualities, LIN is being researched and introduced into RCP environments.

Designing and building an ECU is not a simple task. In control algorithm implementation, register level programming for the ECU is required. This makes designing and testing of control algorithm implementation extremely difficult and costly because of its complexity. In order to reduce the level of difficulty and cost, GM is looking into Rapid Control Prototyping (RCP) systems that have been developed over time which remove the need for register level programming in the prototyping phase of an ECU [10]. The front-runner in RCP development is dSPACE, and GM is working with them to use their latest control systems testing hardware and software packages. dSPACE's systems allow for MATLAB/Simulink controllers to be created and downloaded into an ECU. dSPACE's Control Desk software can then be used to manually change parameters, such as temperature, solar intensity, or humidity to fully simulate a climate control environment for a vehicle.

The design task for this team was to setup the foundation for a fully functional climate control system in a vehicle. The foundation includes designing and developing multiple LIN drivers to be able to transmit and receive signals, via a LIN bus, from the ECU to the actuators and sensors. The team expected the cost of the project upon initial interpretation to be around approximately \$1000 which was \$500 above the allotted budget. This was later determined inaccurate when thoroughly researching and fully understanding each aspect of the team's project. The team must build the final design in such a way that allows for GM to modify the driver design for reusability. A technical document must also be constructed for each driver to fully detail the steps taken to create the driver. Two drivers will be designed for two LIN based sensors including an Indicator Light and Solar Sensor (ILSS) and a Rain Light and Humidity Sensor (RLHS). The LIN drivers are to be designed in Simulink and downloaded to Control Desk to properly read environmental conditions such as ambient light, sensor cover temperature, solar intensity, solar azimuth angle, solar elevation angle, and humidity. Also, a driver for LIN based actuators must be designed to control the basic actuator functions such as speed and end-stop learning functions. The RCP Control Desk software will then be used to conduct testing on the devices to ensure proper communication and functionality. With this foundation set, in perspective, the teams LIN drivers provides the creativity GM needs to utilize and design a fully functional climate control system in a vehicle.

1.2: Background

1.2.1: Local Interconnect Network (LIN)

LIN is a serial network protocol that is used for communication between components in vehicles. As the technology in the automotive industry grows, the necessity for a cheap serial communication network is essential. To illustrate, the Controller Area Network (CAN) bus, which was already in use by the automotive industry, became far too expensive to implement for each component in the vehicle. In the late 1990s, the LIN developers, founded by five automakers (BMW, Volkswagen Audi Group, Volvo Cars, Mercedes-Benz) [6], developed the first fully implemented version of the new LIN 1.3 specification during November of 2002 [9]. A full year later, the latest specification, LIN 2.0, was introduced to expand on LIN 1.3's capabilities and provisions. This was used for additional diagnostics features, such as the ability to be used with the vehicle's battery by use of a Direct Current (DC) LIN transceiver. The reasoning for GM's effort to incorporate LIN 2.0 is caused by a conjunction of wires that is used to connect their current vehicle's analog components. GM's intention is to use this intelligent bus protocol system to create small networks which can all be integrated through the CAN network in their vehicles.

The LIN hardware is a low-cost, single-wire bus network based on an ISO 9141 LIN transceiver, which is an asynchronous serial data clock with a rate of 10.4kBaud. This provides the system with a signal proposition speed and acts as a node in the LIN protocol. This transceiver is based on a UART/SCI hardware interface that is common to most microcontrollers [9]. The LIN hardware also acts as a single master/multiple slave (up to 16) bus network that transmits and receives data in both directions, which can be seen in Figure 1 below. LIN Application Programming Interface (API) provides a given set of C programming functional calls that use a pre-defined set of driver routines which needs to be implemented with each LIN software driver. The LIN software protocol, or API, specifies the nodes in the network, the frames, signals, and schedule table which are all predefined in a LIN Description File (LDF). Time synchronization, via the schedule table, permits the correct transmission and reception of frames and data responses which is initially specified in the LDF. A LIN frame consists of a header and a response segment. To initiate communication with a slave, the master sends the header frame, which consists of a header signal (consisting of a wakeup signal and a node identifier) and response from the LIN slave. If the master wants to send data to the slave, it goes on the header as a Transmit Signal (or Tx Signal). If the master requires data from the slave, the slave sends the response signal (or Rx Signal) [11]. This information is also predetermined in the LDF, which were provided by GM.

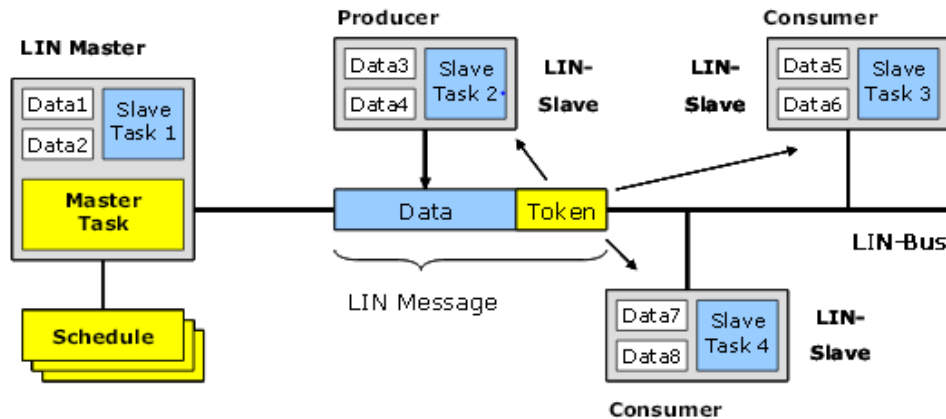


Figure 1 - Lin Network Overview

The LDF can be broken down into 5 different key parts that allow for proper signal propagation to the desired component based on the LIN 2.0 specification [12].

- Master Node and Task - the LDF must specify the master node (e.g. ECU) which is responsible for sending all the frame headers on the bus. This node also controls the timing and schedule table for the bus.
- Slave Node and Task - the LDF must specify the slave node (e.g. actuators and sensors) which are responsible for listening to all the frame headers on the bus and reacting accordingly either by publishing a frame response or subscribing to it.
- Frames - frames are the entities that are transferred on the LIN bus from the master to the slave and vice versa. The frame is constructed with a header, which is sent by the master at the beginning of each new sequence. The frame has a header that carries between one to eight bits of data that contains information such as a break, slave node identifier, sync byte, and slave response delay.
- Signals - a signal is a value transported on the LIN cluster, or LIN bus wire plus all nodes, within the data field of a signal-carrying frame. Several signals can be packaged into one frame as long as they do not overlap in time. The signal response time is arranged in the schedule table.
- Schedule Table - schedule tables are a key component in LIN protocol. A schedule table makes it possible to assure that the bus will never be overloaded by assigning specific and scheduled time delays thus guaranteeing the periodicity of Tx and Rx signals.

The LIN implementation hierarchy can be seen in Figure 2 below, which shows the hardware and software integration in a LIN bus system.

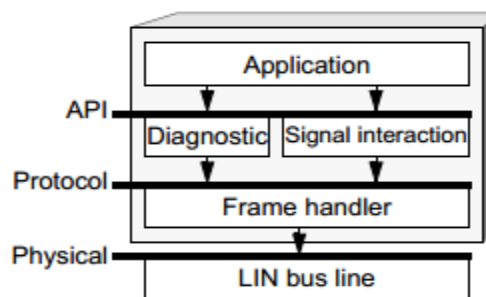


Figure 2 - LIN Implementation Hierarchy

1.2.2: dSPACE

Technology is growing and RCP solutions in all industries are determined to develop new and innovative control systems. dSPACE solutions span into various fields such as automotive, medical engineering industries, and aerospace. As an independent system partner, dSPACE has done pioneering work on tools such as ECU development in the automotive industry to medical device design and testing. dSPACE applications support the complete development phase of their customers, from architecture-based system design and block-diagram-based function prototyping to ECU auto-coding and Hardware-in-the-Loop (HIL) testing. The advantages for automotive customers are considerable savings in cost and time, greatly enhanced software quality, and more efficient cooperation between manufacturers and suppliers.

1.2.3: HIL Simulation

As the number of mechatronic components increases, the complexity of electronic vehicle systems, tools and technologies for developing and testing ECUs are needed for optimizing RCP systems. For this reason an increasing number of HIL systems are being used in the development process, by suppliers as well as automotive manufacturers. HIL involves connecting the actual control inputs to the real-time simulation application, dSPACE Control Desk software, for testing in a vehicle [10]. This makes it possible to perform organized, automated testing of the ECU, sensors, and actuators that are integrated into a vehicle. The advantage of HIL simulation in comparison with iterative ECU tests in prototype vehicles is that the HIL simulation can be performed very early during the development process by use of dSPACE testing software. This allows for errors to be detected and eliminated during the design process, which reduces cost in the overall production of the system.

GM's advancement in the spatial and functional integration of intelligent LIN based sensors and actuators bring a particular challenge to HIL Simulation. With Intelligent sensors and actuators, ECUs no longer have simple analog or digital interfaces, but communicate with sensors and actuators through a standard bus protocol (e.g. LIN) [10]. This is the challenge that this design team has been presented with and the solution will be exposed in the following chapters.

dSPACE provides support of their basic tools, such as a MicroAutoBox and Control Desk software, for developing, testing and calibrating the ECU. The MicroAutoBox used in this project, shown in Figure 3, is the ECU hardware component that is installed in the vehicle. The Control Desk software is the HIL simulation application that allows for the manipulation of variables, such as the saturation limits of an actuator and testing boundary cases on vehicle components. For this project, a LIN driver is to be constructed in Simulink and downloaded in Control Desk through Real Time Workshop (RTW) and ran on the MicroAutoBox for testing of vehicle components. This procedure is the definition of a HIL system. The Control Desk application also supports full installation of a LIN driver in the MicroAutoBox. This provides autonomous use in a vehicle without being tethered to a computer.

1.2.4: MicroAutoBox 1401/1501

The MicroAutoBox 1401/1501 used in this project, shown in Figure 3, has the ability to handle one LIN channel that corresponds to the functionality of one sensor or actuator based on the construction of the LDF. If the LDF is written only for a particular sensor or actuator, only one component can be run on the MicroAutoBox. This constraint will limit the design of the

team's product by not being able to produce a fully functional climate control system that can be installed into a vehicle. This MicroAutoBox was used by GM because of its ability to handle multiple analog input/output channels used in their current climate control systems. This MicroAutoBox will be used as the ECU in this design and will be assigned as the master node in the LDF. It will be connected to the team's computer port via cable and a 34 mm Express Card shown in Figure 4. The Express Card allows for the computer to send and receive the information built in Control Desk and Simulink to the MicroAutoBox for the real-time simulation of the sensors and actuators.



Figure 3 - MicroAutoBox 1401/1501



Figure 4 - DS821 34 mm Express Card

1.2.5: Sensors

Sensors are used in various control systems to gather real-time data for feedback to a controller. An intelligent sensor processes the electrical values provided by the measuring device and makes the technical information available to the control unit via the LIN bus [13]. In a climate control system, it is necessary to have an ILSS and an RLHS sensor in order to provide the control system with the numeric values of the environment inside and outside an automobile. These real-time values can then be used to modulate the temperature output based on the user input. The ILSS is a control unit that is located in the middle of a vehicle close to the windshield. The ILSS, shown in Figure 5 below, has the ability to read the vehicle interior sensor temperature, ambient light, solar intensity, and solar angles such as azimuth and elevation. The ILSS also has two LEDs located just below the top cover which are used to show the charge status and theft deterrent. This sensor uses the solar intensity and solar angles to monitor heat from the sun on the vehicle exterior. The ILSS also monitors the temperature of a vehicle. These two inputs are used as control inputs for climate control modulation given a specific user input. It also reads the ambient light just above the top cover that is used to automatically turn on the vehicle headlights when below a certain value. The ILSS was manufactured to have LIN capability which means that it is able to send and receive signals from the master via a single wire bus.



Figure 5 - ILSS Sensor

The RLHS, shown in Figure 6 below, has the ability to read relative humidity, vehicle interior temperatures, and glass temperature. The RLHS shall be mechanically attached to the front windshield through a windshield mounted attachment device, which is adhered to the glass windshield with an adhesive film. The RLHS is able to recognize the humidity level on both the windshield and inside the vehicle compartment, and be able to monitor the ambient lighting conditions. These two conditions are to be used as the control inputs for climate modulation within the vehicle interior compartment and for light on/off information. They are also used to calculate the fog risk on the windshield as well as providing the ability to reduce fuel consumption by decreasing the air conditioning output to its minimum without causing moisture build upon the windshield. Like the ILSS, the RLHS was manufactured with LIN capability.



Figure 6 - RLHS Sensor

1.2.6: Actuators

Actuators are used in a climate control system in order to control the interior climate of the vehicle. Intelligent actuators work contrary to sensors, in that they receive information from the ECU and turn it into electrical values to control mechanical, hydraulic or electrical actuating elements [13]. LIN based actuators are used to move the Air Inlet (recirculation), Air Delivery (or mode), and Temperature Blend (mix) Flaps in an Heating Ventilation and Air Conditioning

(HVAC) case module, which is the part that contains the Blower Motor, Evaporator (cooling the discharge air), and Heater Core (heating the air delivered to the vehicle interior). Integrated into a complete system, the actuators described can modulate the climate based on a user set point.

For this design project, the team was given the Air Inlet and Air Delivery actuators from GM with the purpose of establishing a LIN controller using Simulink to drive both actuators simultaneously. These two actuators are used to open and close vents based on the user input. These actuators can be seen in Figures 7-8. The other four actuators are not going to be tested in this report due to the fact that they could not be provided by GM and were well over the budget allotted to the team. However, the team was capable of programming the missing actuators in the Simulink LIN Driver for GMs use.



Figure 7 - Air Inlet Actuator



Figure 8 - Air Delivery Actuator

1.2.7: Integration in a Vehicle

When the full system is integrated in a vehicle, the previously stated actuators will be controlled in the following manner. Figure 9 shows the new 2015 GM Z06 Corvette climate control interface where the user can set their preferred settings. The user can choose to have the air recirculated by pressing the Air Inlet button. The user can also adjust the delivery style by choosing from several options and the speed by tuning a dial. The vehicle interior temperature can also be altered setting a temperature on the temperature blend flaps dial. If the user sets the temperature higher than the sensor reading values, the heater core actuator will initiate delivering warmer air into the interior of the vehicle. If the user set point is lower the sensor readings, the evaporator actuator will initiate delivering cool air into the vehicle interior. This design project will establish a driver to control the air inlet and air delivery options shown below.



Figure 9 - GM Z06 Corvette Climate Control User Interface

Figure 10 shows a flow diagram of how the system is integrated together. The multiple sensors and user settings are used as inputs to the climate controller. In this diagram, the user settings are inputted into a controller and matched to the sensor readings. The controller then produces an output value which is used to modulate the blower motor, evaporator, heater core, and temperature blend flap actuator characteristics to achieve the user setting value for the optimal climate and delivery method.

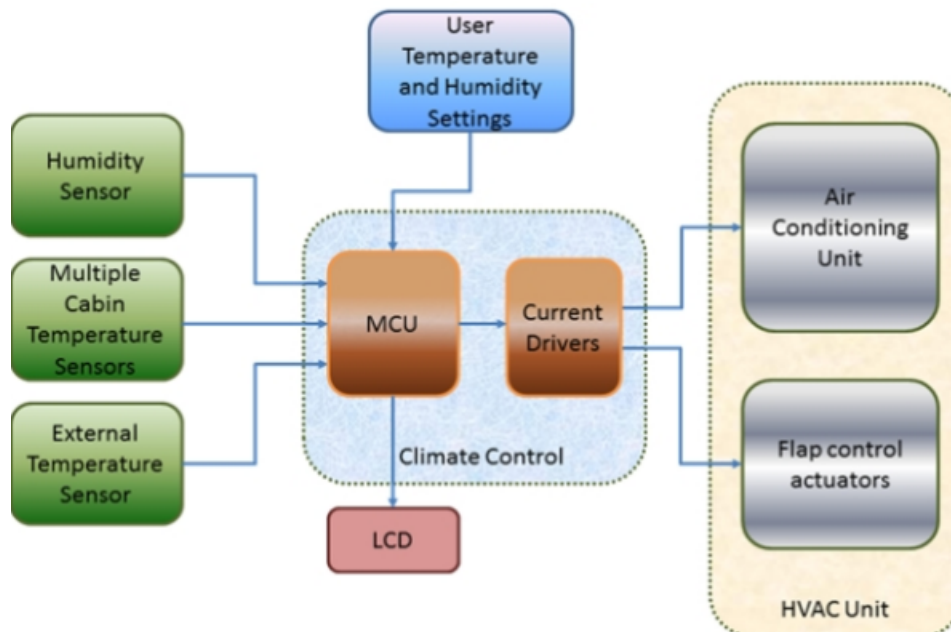


Figure 10 - Climate Control Flow Diagram

Chapter 2: Solution Analysis and Approach

2.1: Fast Diagram

The Function Analysis System Technique (FAST) Diagram is used to show the functions that the RCP system will implement, which is shown in Figure 11 below. This diagram moves from left to right stating the main function then continuing to the primary function and finally the secondary functions. The main function of this system is to create an RCP Environment, which is shown in the first column of the FAST diagram. The next columns are three different primary functions that are needed for the primary function to work. The next columns are secondary functions that describe how the primary or secondary functions will be accomplished. This FAST diagram is an easy way to visualize the basic functions of an RCP system and how they rely on one another.

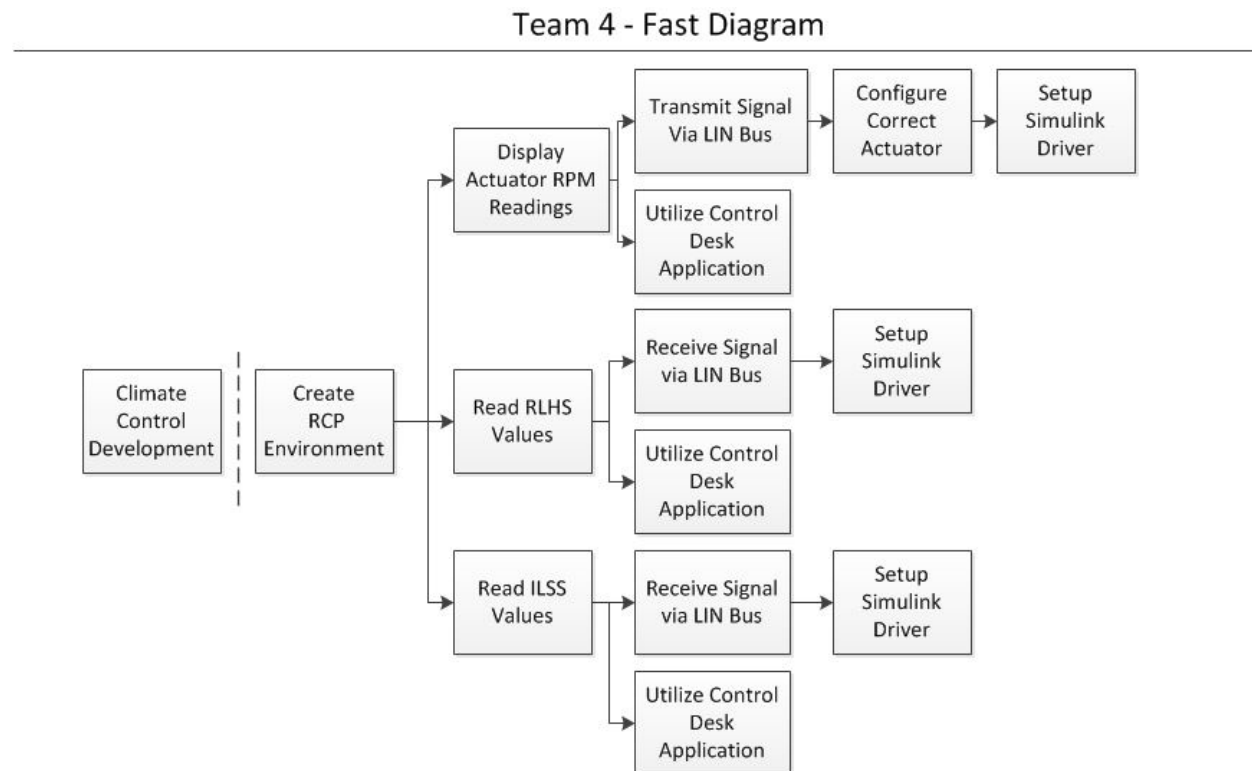


Figure 11 - FAST Diagram of Proposed Climate Control Development

The FAST diagram above references the software and hardware components that are needed to create the teams RCP environment. The LIN driver must be created in Simulink and simulated on Control Desk. The hardware consists of the sensors and actuators connected to the ECU via a LIN bus. The Simulink driver sets up signal transmission by utilizing the LDF. This driver block structure is converted into C code and ran on the ECU which is the LIN master. The master then delegates which signal can be propagated to each of the slaves based on a schedule table. The slaves then send a response signal, which are transmitted through the same LIN wire and read through Control Desk, thus creating an RCP environment through HIL theory.

2.2: Critical Customer Requirements

The Critical Customer Requirements (CCRs) can be determined based on the House of Quality technique. The customer laid out certain requirements that the team had to meet while designing the LIN drivers which are bulleted below.

Requirements:

- Must use Simulink to design LIN driver
- Need to create a re-usable dSPACE driver library in Control Desk that can easily be modified
- Must use LDF's provided by GM
- Operating Voltage must be 12 V DC
- Data read by sensors through LIN bus must be within signal transmission tolerances noted in specifications
- Data transmitted to actuators must be within signal transmission tolerances noted in the specification
- Must provide a detailed technical report for each LIN driver created

After all the customer requirements were known, the team related these to the House of Quality design. The House of Quality measures the critical customer requirements and ranks them based on their importance to the customer. It is crucial to meet all the customer specifications in order to produce a useful product that the customer can benefit from. These customer requirements also affect the decisions designing each LIN driver and on what scale of development which is discussed in the next section.

2.3: Conceptual Designs

Individual LIN Driver Design using GM LDFs

Currently, GM does not have a LIN driver to read the environmental conditions from their sensors. They also do not have a LIN driver to properly send a signal to an actuator and read the speed of the actuator through signal transmission via the LIN bus. Using the LDFs provided by GM, the team is capable of simulating and testing one sensor or up to four actuators at a time. This is due to the LDF having node identifiers, frame, and signal characteristics for only one vehicle component. This design strategy would require the team to design three different Simulink drivers: One for each of the sensors and one for the two actuators provided by GM. These drivers will then be simulated individually in Control Desk to test signal propagation from the LIN master to the single slave. This design establishes the foundation of LIN bus communication but does not utilize the single master multiple slave performance benefit that the LIN protocol was developed for. A flow chart of this design can be seen in Figure 12 below.

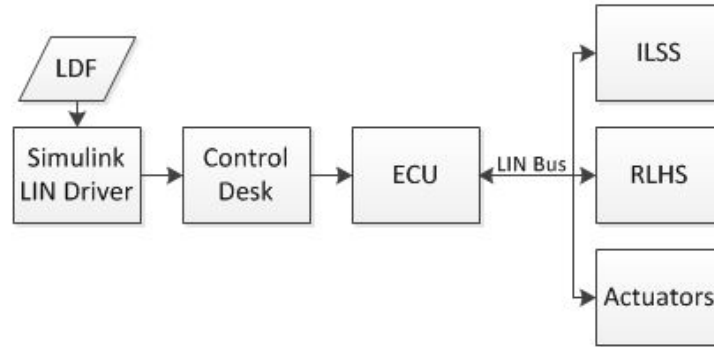


Figure 12 - LIN Driver Flow Diagram

Integrated LIN Driver Design Combining LDFs

GM provided the necessary signal and frame characteristics for each of the actuators and sensors in the LDFs. Another conceptual design would be to create a single LDF file that combines the three individual LDFs. This design will allow for more flexibility of our final design because the team could create a single LIN driver for the three vehicle components and simulate them all at once using Control Desk. This design truly utilizes LIN's advantages of being able to use a single wire to communicate with multiple slaves. The flow chart for this design is the same as Figure 12.

Fully Integrated Climate Control System

For full scale implementation, the team would use the fundamental design concept of either of the previous two conceptual designs and the integration of a feedback controller. The two previous design concepts set up the drivers for the individual sensors and actuators. To implement the idea of a full scale climate control system, the system should be able to adjust the heater core, evaporator and blower motor actuators according to the difference between the user input and the data from the environment. In order to control the functionality of the actuators and sensors, a controller must be developed to achieve the desired outcome. The parameters of Proportional-Integral (PI), Proportional-Derivative (PD), and Proportional-Integral-Derivative (PID) controllers would be compared to see which controller will give the optimal results in the testing of the complete system.

A PI closed loop controller can minimize the system response time and remove the steady state error which could potentially be a good solution for controlling the actuators. PD controllers, due to having a quick rise and transient response time [2], would be the best controller to match the high speed feedback from the sensors to the LIN master. A PID controller combines the benefits of the PI and PD controllers which will maximize the speed and accuracy of the feedback signal from the actuators and sensors to the LIN master. These controllers could be created in Simulink and be used in a similar way the previous two design concepts. The difference in this full scale model is that the team would need to acquire the missing actuators that were not provided such as the evaporator, heater core, and blower motor. This would greatly increase the total cost of the design experiment and is beyond the scope of what GM requires from the team. A flow chart of this design can be seen in Figure 13 below.

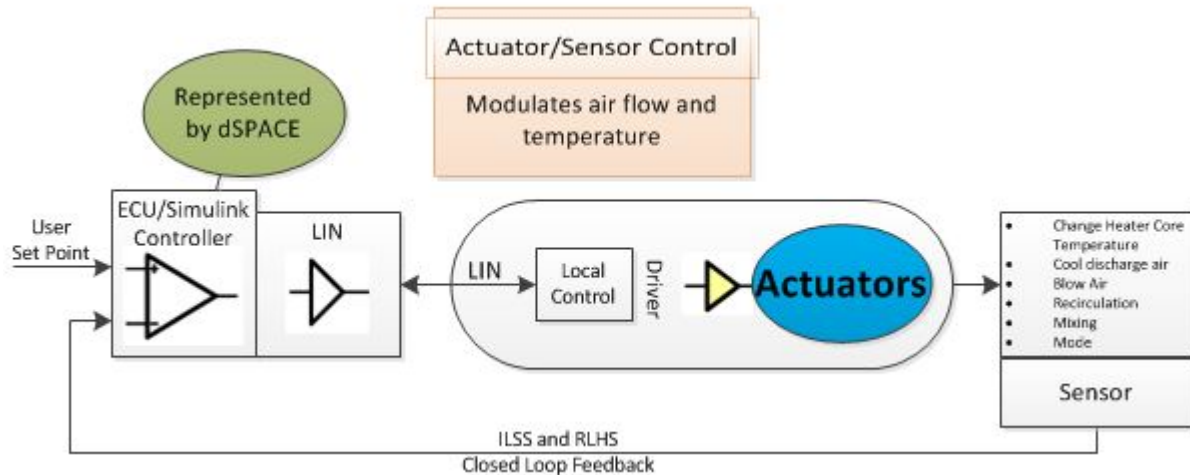


Figure 13 - Fully Integrated Climate Control Flow Diagram

2.4: Ranking of Conceptual Designs

In order to see the best design solutions, the prior three designs were compared using a feasibility matrix which is shown in Table 1 below. This matrix has three important details for each of the designs which are cost feasibility, implementation complexity, and lead time. The three designs are ranked on a scale from 1-5 in each category. A rating of 5 represents a great feasibility option and a rating of 1 representing an infeasible option.

Table 1 - Prototype Conceptual Design Stages

Design #	Description	Cost Feasibility (5-great 1-poor)	Implementation Complexity (5-simple 1-complex)	Lead Time (5-best 1-worst)	Average Feasibility Ranking
1	Individual LIN Driver Design using GM LDFs	4	4	5	4.33
2	Integrated LIN Driver Design Combining LDFs	4	3	3	3.33
3	Fully Integrated Climate Control System	1	1	1	1

2.5: Final Design Decision

Design 3 was proven not to be possible in the time allotted for this project. The team was not able to acquire the necessary components necessary to begin this design such as the evaporator, heater core, and blower motor. To acquire these components with the budget allowed was also out of the question.

Design 2, being integrated LDF design, was not able to be implemented in this project due to the node addressing of the sensors conflicting in the LDF. The LDF's specified address nodes caused an error when inserted in the dSPACE Simulink block set. The assembly programming within the sensors would need to be reprogrammed with non-conflicting node

addresses in order to be implemented in the same LDF. The lead time that was needed by GM to reprogram the sensors was insufficient for the time allotted for this project.

Design 1 was the implemented design that the team used. The team was able to design an individual Simulink LIN driver and dSPACE Control Desk setup for the sensors and actuators which will be described in detail in the next chapter.

2.6: Budget

The budget in Table 2 below shows how much it would cost if the team were to buy all of the equipment. Some of the information in the table is GM confidential and cannot be documented. These pieces of equipment are denoted with the symbol ** and are market at \$1. The actuators, sensors, LIN buses, and the ECU were provided by GM with no cost to the team. The Dongle, which includes the RCP and RTI software, and the DS821 Express Card were generously provided by dSPACE through a university loan which was no cost to the team if returned in the proper condition. The MATLAB/Simulink 2010a license was provided by MSUs Department of Computer Services (DECS) which was also no cost to the team. The total preliminary cost, which can be seen in the Table below, to begin building was well over the initial budget of \$500 allotted to the team. The team spent a total of \$0 out of the budget for the preliminary design.

Table 2 - Preliminary Budget Estimate

Product	Description	Qty.	Price/Per	Total
DS821 Express Card	34mm DS821 Host Interface (Express Card) SN#380475	1	\$750	\$750
Dongle	Includes the following software: CDP(Control Desk, RTI,Platform_API), CCPPPC, RTILINMM Blockset	1	\$500	\$500
MATLAB/Simulink 2010a	MATLAB/Simulink 2010a License	1	\$299.95*	\$299.95*
ECU	MicroAutoBox 1401/1501	1	\$10,000	\$10,000
Actuators	Air Inlet & Air Delivery (Mode)	3	\$1**	\$1**
ILSS Sensor	Indicator, Light, and Solar Sensor	1	\$1**	\$1**
RLHS Sensor	Rain, Light, and Humidity Sensor	1	\$1**	\$1**
LIN bus	LIN bus each sensor and actuator	5	\$20	\$100
			Total Cost =	\$11,652.95

* Typical price of MATLAB 2010a off the internet

** Denotes GM confidential information that cannot be documented in this report. The price is presented as one dollar

2.7: Gantt Chart

The Gantt chart was a way for the group to stay organized and have an overview of the different tasks that needed to complete this project. Identification of the team individual tasks, deadlines, and categories helped to keep the team organized and on a schedule. The Gantt chart also allowed the team to visualize the tasks that needed to be completed and to monitor the progress. The Gantt chart started from the research aspect of the project to the final product which will be displayed on design day. The Gantt chart can be seen in Appendix 3 Table 3 due to it being very large in size.

Chapter 3: Technical Description of Work Performed

3.1: Hardware Implementation

3.1.1: Setting up the MicroAutoBox and dSPACE Software

The MicroAutoBox features an RCP system and an ECU in one package. This makes it a suitable hardware for dSPACE prototyping in a vehicle. The MicroAutoBox used in this project is the 1401/1501 variant. The package content for this variant includes three boards in a milled aluminum box. One of those is the DS1401 baseboard and the other two are the DS1501 I/O boards. There is a 156-pin Zero Insertion Force (ZIF) connector for matching the corresponding connector on the MicroAutoBox that needed to be configured for power, ground, and LIN. This provides access to the input and output signals provided by the MicroAutoBox. The I/O connector pins are specially made to work with the ZIF connector. The image in Figure 14 below shows the components necessary for configuring the MicroAutoBox.

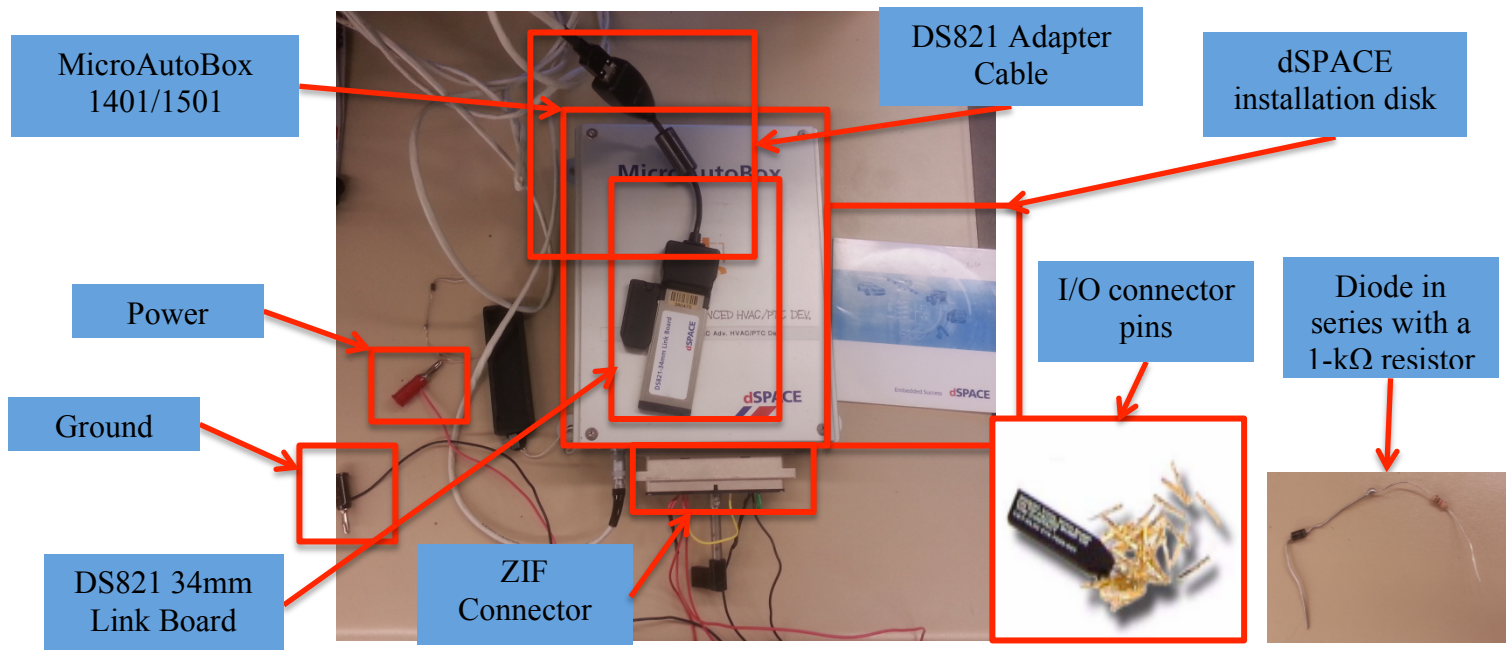


Figure 14 - Parts Required

The dSPACE software was installed on the team's personal computer (PC) before getting started. The block sets that came with this software are not open sourced. The price can be found in Table 2. The dSPACE software contains all the necessary Simulink block sets to establish a LIN driver as well as the Control Desk Software. The team needed administrative rights on the PC to install the dSPACE software.

3.1.2: Switching on the MicroAutoBox

To turn on the MicroAutoBox when it is connected to a battery, the team connected a wire from the Remote signal (pin M3) to pin V4 (power pin) on the ZIF connector as shown in Figure 15 below. This allows the MicroAutoBox to turn on when power is introduced to the unit. The direct connection method that the team used is not suitable for installation in a vehicle. This is because the MicroAutoBox configured in the current setting, when connected to a battery or power supply, will constantly be running and drawing power from the power source. This would rapidly drain the battery in a vehicle when the vehicle is turned off. For the proper use of the remote signal in a vehicle, it must use a KL15 switch that triggers through by the output of the ignition/driving switch shown in Figure 16 below. This allows the MicroAutoBox to turn on when the switch receives a signal that the car is trying to be started thus solving problem of continuous operation and battery depletion.

An I/O pin was inserted to both ends of the wire. The yellow wire shows Remote signal connected to the power input. The green wire shown below is used to connect the VDRIVE pin to the VSENS pin on the ZIF connector. When the power and ground was connected to the battery, the LED located in the top left corner rendered red. With a red LED, the MicroAutoBox was in the reset state meaning the power supply is switched on but no application is downloaded to the real-time processor. When the LED was green, the Simulink LIN driver application was running through Control Desk.

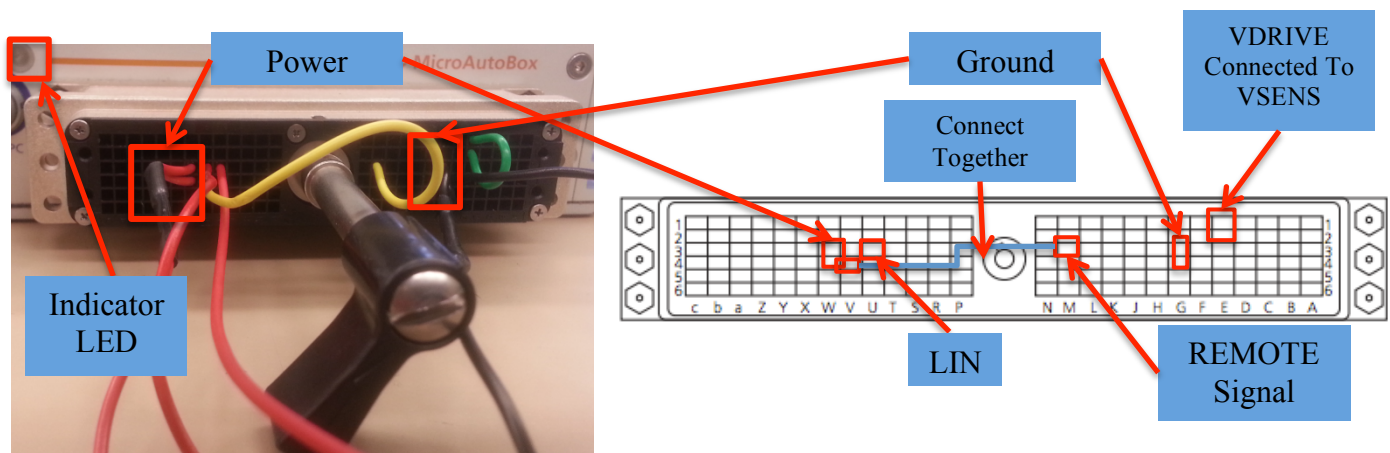


Figure 15 - Configuring Power, Ground, REMOTE Signal, and VSENS/VDRIVE Pins

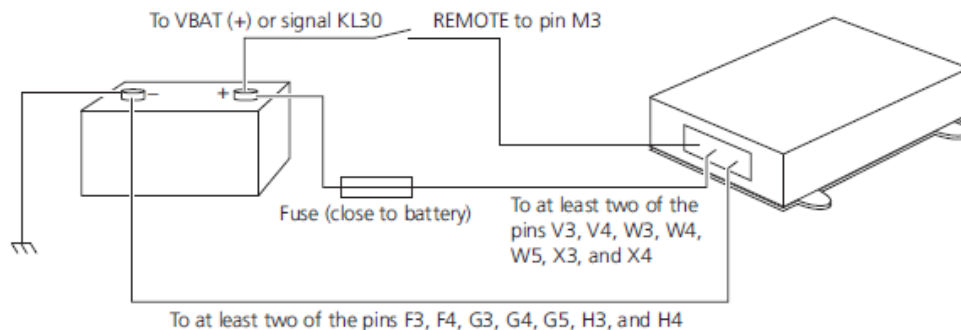


Figure 16 - KL15 Switch and Fusing

3.1.3: Power Supply Requirements

The MicroAutoBox requires a power supply in the range of 6-40 V (30 W with a 5 A peak). Since the MicroAutoBox is to be powered by a vehicle, the team set a power supply, shown in Figure 17 below, at an operating range of 12 V DC with a 5 A peak current. The 5 A current was the current specification for the MicroAutoBox to turn on. The team also inserted a 10 A fuse to the positive terminal of the power supply in order to prevent a large current spike produced by our system from damaging the MicroAutoBox and the sensors and actuators. When the sensors and actuators were tested, they drew 1 A of current which was well below the startup current level.



Figure 17 - Power Supply

3.1.4: Setting MicroAutoBox as the LIN Master

By default, the MicroAutoBox is configured as a LIN slave on the LIN network. To configure it as the LIN master, a diode in series with a 1 K Ω resistor was soldered together as shown in Figure 14. The diode was used in order to prevent the DC power source from being connected backwards and causing damage and the 1 K Ω resistor was added to act as a pull up resistor delivering the proper voltage to the LIN transceiver. The diode/resistor series connection was then wired in parallel between the LIN transceiver pin (U3) and an empty VBAT (Power) pin as shown in Figure 18 below. This ensured that the MicroAutoBox was configured as the LIN master.

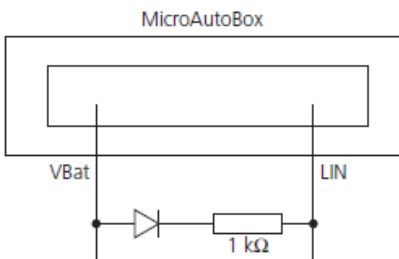


Figure 18 - LIN Master Configuration

3.1.5: Connecting MicroAutoBox to PC

With the MicroAutoBox turned off (e.g. not connected to the battery) connect the pin end of the crossed-over patch cable to the MicroAutoBox and the Ethernet end to one of the DS821 adapter cables shown in Figure 19 A-B that came with the dSPACE package. The Express Card was inserted in the other end of the adapter cable and then connected to the team's computer's PCI card slot. Figure 19 A shows a description of the indicator LED status on the 34 mm Express Card. The PCI card slot looks similar to Figure 19 B.

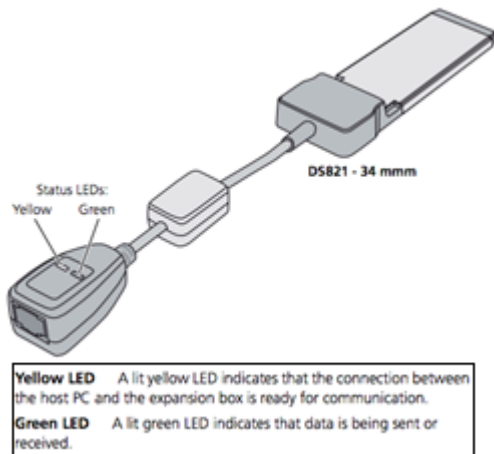


Figure 19 A - DS821 Adapter Cable



Figure 19 B - PCI Card Slot

3.1.6: Connecting the ILSS to the MicroAutoBox via LIN

The ILSS has a pin assignment that is necessary when connecting to the MicroAutoBox. It is required all variants are applied correctly to ensure proper functionality. Table 4 below shows the pin assignment for the ILSS sensor. Pin 1 was not used in this application. Pin 2 was connected to pin U3 on the MicroAutoBox to establish LIN serial data communication on an external proto board. Pin 3 and Pin 4 were not used but can be used to externally drive the LEDs. The design of the teams' LIN driver controlled the LEDs. Pin 5 was connected to one of the VBAT inputs on the ZIF connector. This design team used the power pin V3 connected to the positive bus of an external proto board. Pin 6 was grounded to one of the GND pins of the ZIF connector. This was connected from pin H4 on the MicroAutoBox to the negative bus of the external proto board. The proto board was used to easily connect and disconnect each components LIN wire for separate testing due to the fact that the MicroAutoBox only has one LIN channel.

Table 4 - ILSS Pin Layout

Connector Terminal Pin Number	Type	Function
1	Output - OBD LED GND	(if needed)
2	BiDir - Serial Data	LIN Serial Data
3	Input - OBD LED HSD	Charge Status LED (Green)
4	Input - OBD LED HSD	Charge Status LED (Amber)
5	Input - Pwr	12V Battery (+)
6	Output - Pwr	Common Ground (-)

It was critical to orient the ILSS sensor in the proper position in order to have the correct pins of the ILSS housing connected to the LIN wire harness as shown in Figure 20 below.

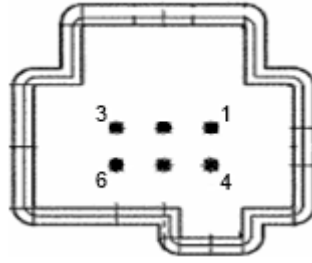


Figure 20 - ILSS Pin Orientation

3.1.7: Connecting the RLHS to the MicroAutoBox via LIN

The RLHS has a pin assignment that was necessary when being connecting to the MicroAutoBox. Table 5 below shows the pin assignment for the ILSS sensor. Pin 1 was connected to one of the VBAT inputs on the ZIF connector. This design team used the power pin V3 connected to the positive bus of an external proto board. Pin 2 was connected to pin U3 on the MicroAutoBox to establish LIN serial data communication on the external proto board. Pin 3 was grounded to one of the GND pins of the ZIF connector. Like the ILSS, this pin was connected from H4 of the ZIF connector to the negative bus of the external proto board.

Table 5 - RLHS Pin Assignment

Pin Number	Pin Name	Description	Current Load
1	BAT	Power (+)	30mA (Nominal) 600mA (peak)
2	DATA	Bi-Directional Interface	10mA
3	GND	Ground (-)	30mA (Nominal) 600mA (peak)

It was critical to orient the RLHS sensor in the proper position in order to have the correct pins connected to the housing of the RLHS as shown in Figure 21 below.

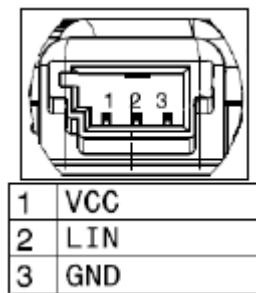


Figure 21 - RLHS Pin Orientation

3.1.8: Connecting the Actuators to the MicroAutoBox via LIN

The actuator has a pin assignment that is necessary when connecting to the MicroAutoBox. It is required that all variants are applied correctly to ensure proper functionality. Table 6 below shows the pin assignment for all of the actuators provided. Pin 1 is to be grounded to one of the GND pins of the ZIF connector. This design team used pin H4 connected to the negative bus of an external proto board for ground. Pin 2 is to be connected from pin U3 on the

MicroAutoBox to the external proto board to establish LIN serial data communication for the actuators. Pin 3 was not used in this application. The purpose of Pin 3 is to give the actuator an alternative address which can be used in a combined LDF (more than one device) if the fundamental address is taken by another devices node. Pin 4 was connected to one of the VBAT inputs on the ZIF connector. This design team used pin V3 for our VBAT input connected to the positive bus of an external proto board.

Table 6 - Actuator Pin Assignment

Pin Number	Pin Name	Description
1	GND	Ground (-)
2	LIN	Bi-Directional Interface
3	Adress Pin	Alternative Address
4	Vact	Power (+)

It is critical to orient the actuator in the proper position in order to have the correct pins of the actuator housing connected to the LIN wire harness as shown in Figure 22 below.

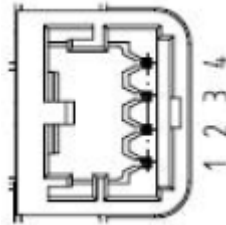


Figure 22 - Actuator Pin Orientation

The final setup of the external proto board with the power and ground connections can be seen below in Figure 23.

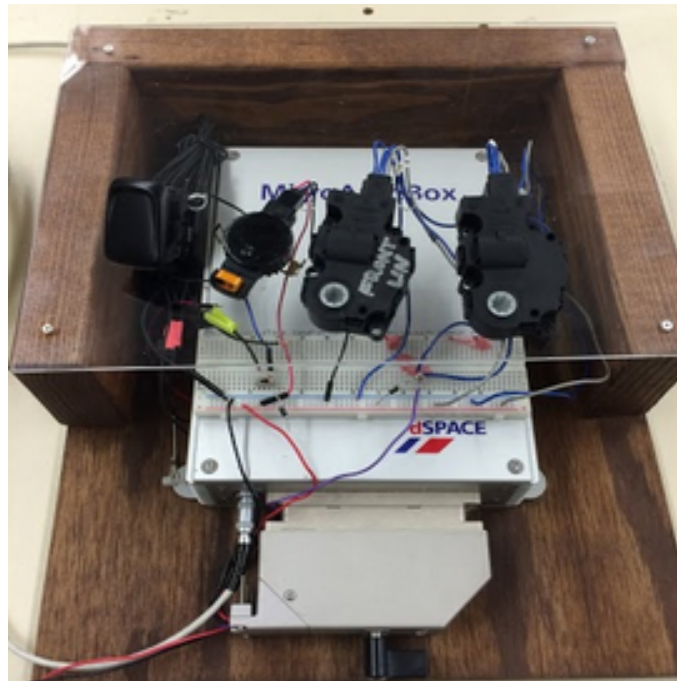


Figure 23 - Final Proto Board Layout

3.2: Software Implementation

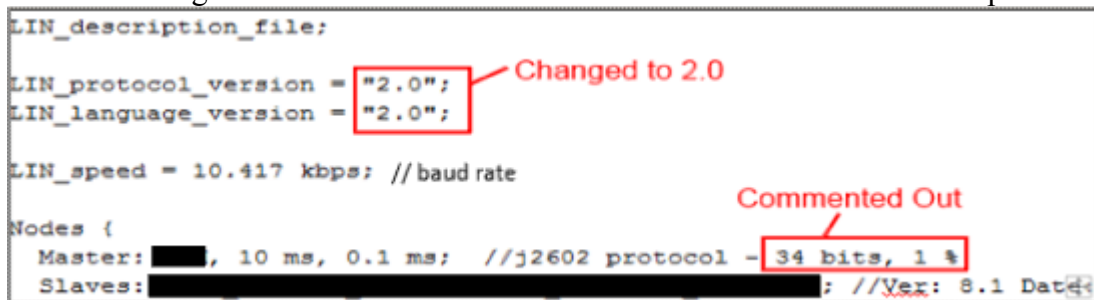
3.2.1: LDF Programming

Due to the LDF's being GM proprietary information, their contents are prohibited from being displayed in this report.

For the LDF's to work with the DSPACE Real-Time Interface LIN Multi Message (RTILINMM) Simulink block sets, they needed to be altered from the form that GM delivered them in. The LDF's were written with a J2602 protocol which is not supported with the teams given version of Control Desk 6.6 or MicroAutoBox 1401/1501. J2602 is a protocol written over the top of LIN 2.0 that allows for the reduced complexity of some software elements of the specification in an effort to reduce the size and complexity of embedded software required in LIN slave nodes [4]. Although J2602 is a great feature, it was unable to be used in this design project. The team had to retailor each LDF in order to be written purely in the LIN 2.0 specification which allowed for the correct compilation in the Simulink Main Setup Block which will be described in the next section.

The following changes needed to be done in all three of the LDF's:

- 1) In the LIN version and node section, as shown in Figure 24 below, the following code was changed and commented out because it is not used in the LIN 2.0 specification.



```

LIN_description_file;

LIN_protocol_version = "2.0";
LIN_language_version = "2.0";

LIN_speed = 10.417 kbps; // baud rate

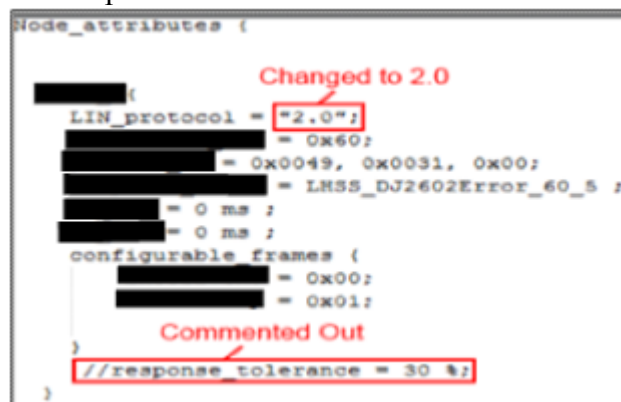
Nodes {
  Master: [redacted], 10 ms, 0.1 ms; //j2602 protocol - 34 bits, 1 %
  Slaves: [redacted]; //Ver: 8.1 Data
}

```

Figure 24 shows a snippet of LIN description file code. Two red boxes highlight changes: the first box contains "2.0" for both LIN_protocol_version and LIN_language_version, with a red arrow pointing to it from the text "Changed to 2.0"; the second box contains "34 bits, 1 %" for the j2602 protocol, with a red arrow pointing to it from the text "Commented Out".

Figure 24 - LIN Version and Node Section

- 2) In the Node Attributes section of the LDF, the LIN_protocol version was changed to 2.0 and the response tolerance was commented out because it is not used in the LIN 2.0 spec. These two changes have to be done for all of the nodes in each LDF. Figure 25 below is an example one node.



```

Node_attributes {
  [redacted] {
    LIN_protocol = "2.0";
    [redacted] = 0x60;
    [redacted] = 0x0049, 0x0031, 0x00;
    [redacted] = LMS5_DJ2602Error_60_5 ;
    [redacted] = 0 ms ;
    [redacted] = 0 ms ;
    configurable_frames {
      [redacted] = 0x00;
      [redacted] = 0x01;
    }
    //response tolerance = 30 %;
  }
}

```

Figure 25 shows a snippet of Node attributes code. Two red boxes highlight changes: the first box contains "2.0" for LIN_protocol, with a red arrow pointing to it from the text "Changed to 2.0"; the second box contains "//response tolerance = 30 %;", with a red arrow pointing to it from the text "Commented Out".

Figure 25 - Node Attributes Section

3.2.2: ILSS LIN Driver

The final ILSS Simulink LIN driver designed by the team reads the temperature, ambient light, solar intensity, solar angles, LIN signal propagation response time, and controls the theft deterrent indicator (amber) LED and charge status indicator (green) LED is shown below in Figure 26.

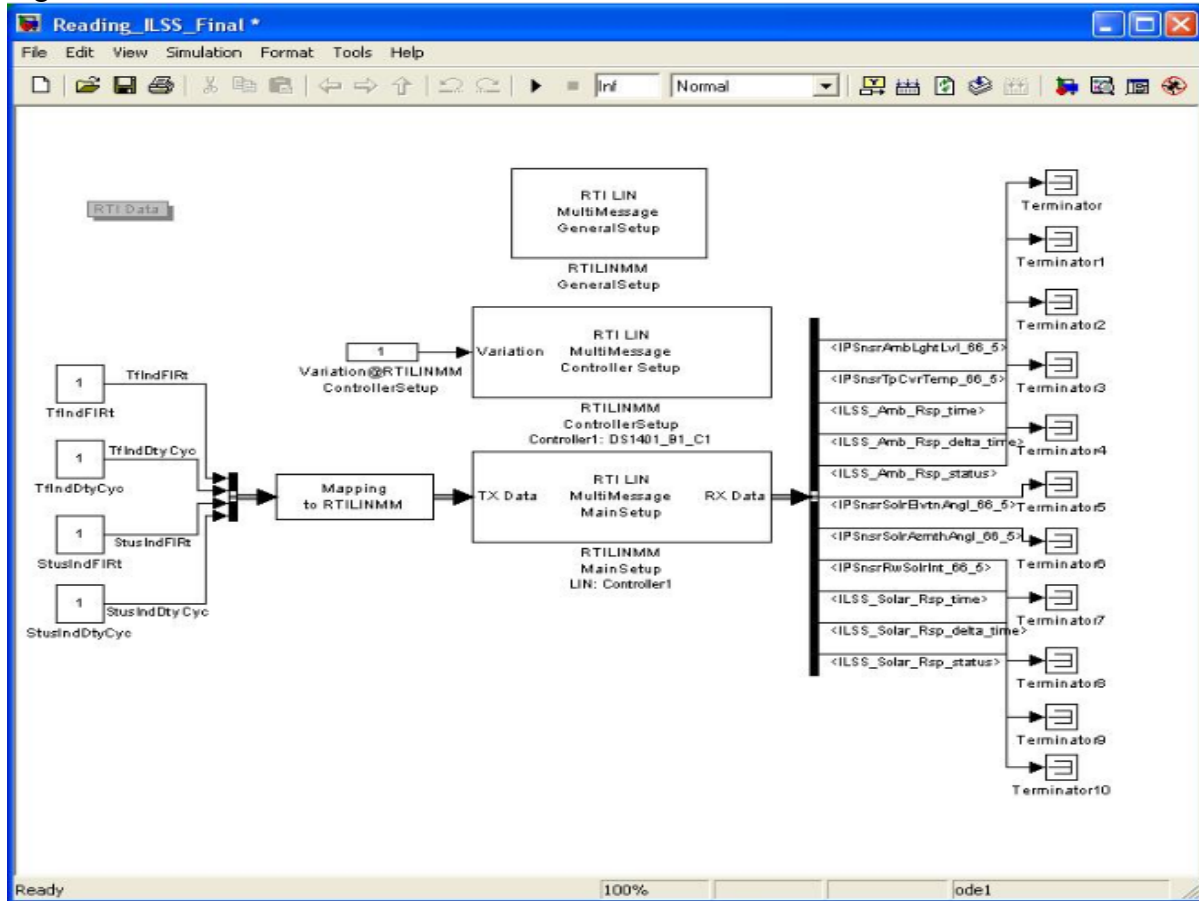


Figure 26 - ILSS Simulink LIN Driver

The first block is the General Setup Block that allowed the team to specify a folder that Simulink was used to store all the relevant files. This block is the same for all three LIN drivers besides the folder where each driver is stored. The contents of the block can be seen below in Figure 27.

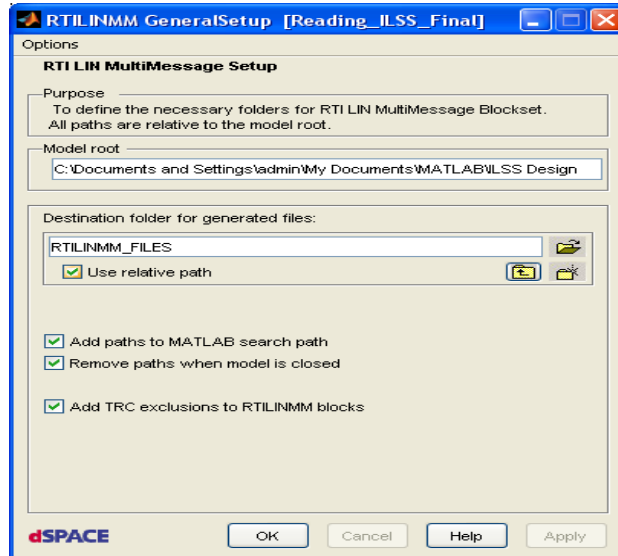


Figure 27 - General Setup Block

The next block used was the Controller Setup Block shown in Figure 28 below. This block specified the controller name, interface specifications, interface settings, and transceiver settings. The baud rate was chosen the same as the value in the LDF. The break length and delimiter have an industry standard setting of 13 bits and 1 bit respectively, which is detailed in the LIN 2.0 specification [11]. This block creates S-Function API which is a dynamically linked sub-routine that the MATLAB interpreter can use to automatically load and execute Simulink code.

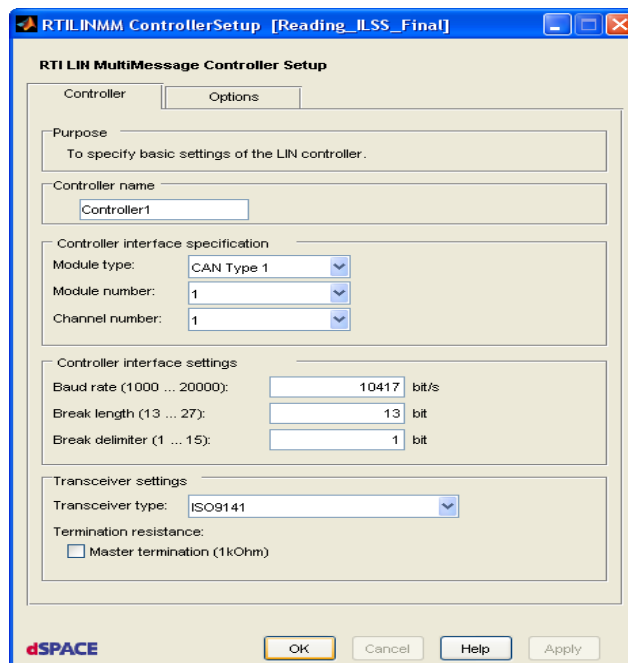


Figure 28 - Controller Setup Block

The final RTILINMM block used was the Main Setup block which specified the LDF and allowed for the selection the LIN nodes, frames, signals, etc. The following Figures show the main items that needed to be populated in order for the ILSS LIN driver to function. The LDF

was inserted into the main block as shown in Figure 29. Using the information in the LDF, the main block uploaded all the information into a tabular hierarchy structure that allowed the team to visualize the contents of the LDF.

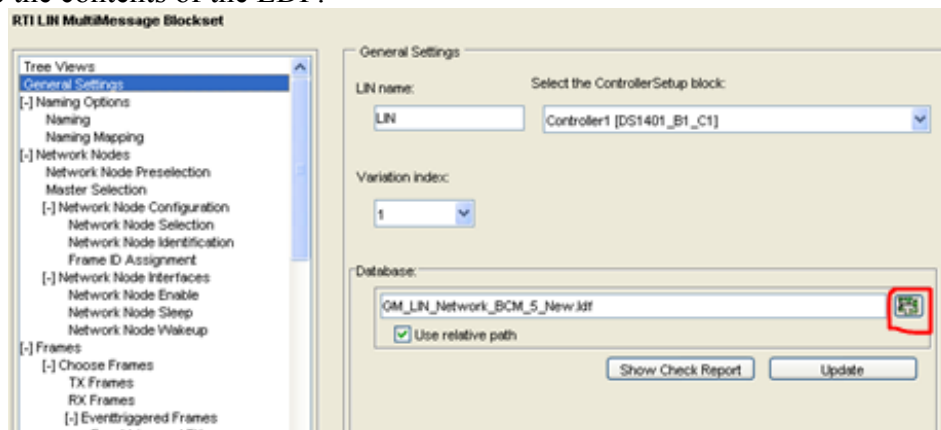


Figure 29 - LDF Specification

Figure 30 allowed the team to preselect which nodes were to be active in the LIN driver. The ILSS node contains all the frames and signals necessary to read the desired environmental characteristics that the ILSS is capable of reading. For this reason, the team chose this particular node. The team also chose the Basic Control Module (BCM) node which is used as the LIN master node. This node specifies the schedule table and sends the initial wake-up frame to the ILSS node. The Control Desk software requires a .trc file which saves the LIN driver settings and lays it out in Control Desk. This will be seen in the next chapter. Figure 31 below allowed the team to specify the schedule table source as a .trc file for the reason stated previously. Due to the limited design the team was able to complete in this project, the other nodes seen in the figure below were not used.

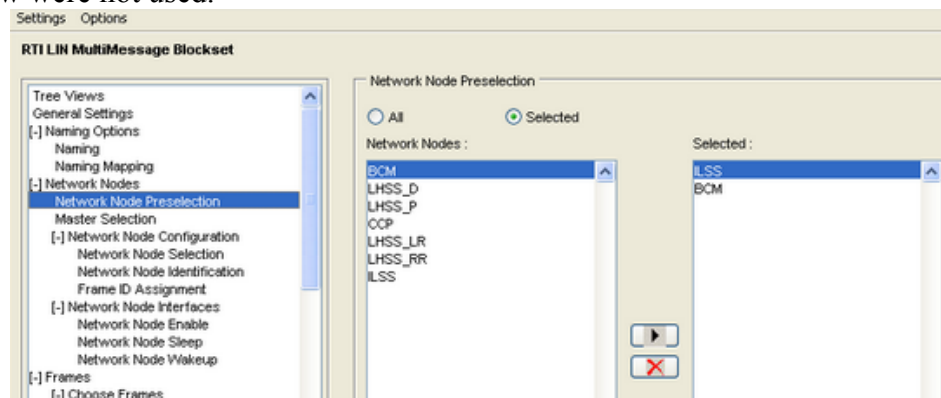


Figure 30 - Network Node Pre-selection

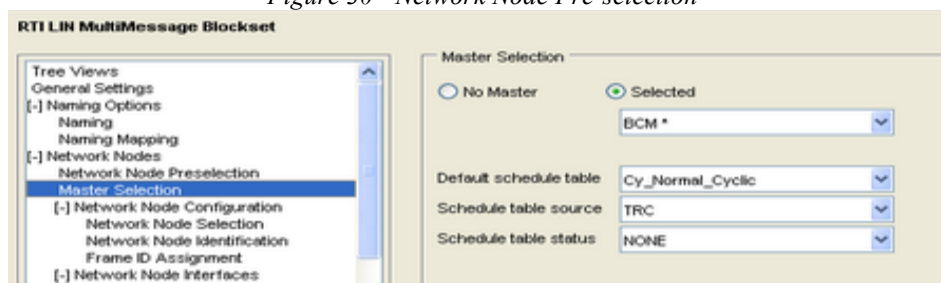


Figure 31 - Master Selection

The Tx Frame's tab was used to specify what frames were to be transmitted by the BCM node to the ILSS node. The frame selected in Figure 32 contains the address information for the theft deterrent and charge status LEDs. The ILSS_CTD_CSI_Cmd frame was sent by the BCM node with the LED signal information that is predefined in the LDF. Using this frame, the ILSS is capable operating the LEDs. The MasterReq and SlaveResp were also selected as Tx frames. These two frames allow for the transmission of an 8 byte data sequence between the master and slave nodes which contain diagnostics information that could be used to troubleshoot a problem if a fault occurred. The frame would return an address corresponding to a particular fault address that could be matched in the ILSS specification for isolating an issue.

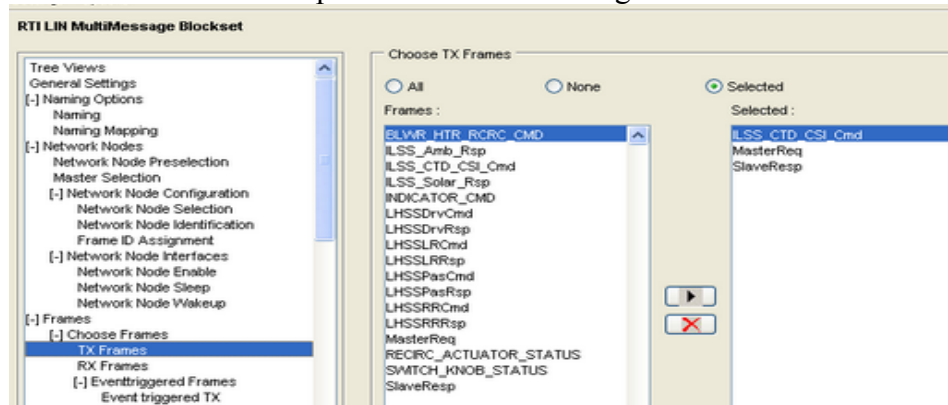


Figure 32 - Tx Frames

The Rx Frame's tab shown in Figure 33 was used to specify what frames were to be transmitted by the ILSS node to the BCM node. The frames selected (ILSS_Amb_Rsp and ILSS_Solar_Rsp) contain the raw data information collected from the ambient and the solar detectors on the ILSS. The LDF specified a specific address for each of the two ILSS slave nodes that the master requested in order to receive the raw data transmitted on the two Rx frames. The rate at which the BCM node received the raw data was set according to the frame position in the schedule table. The SlaveResp frame selected was used to provide diagnostics information as explained earlier.

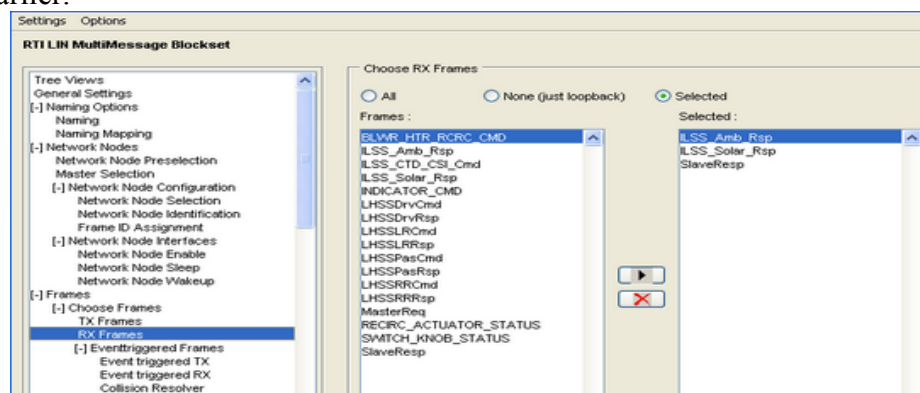


Figure 33 - Rx Frames

The Tx signals selected in Figure 34 below contain the information for the theft deterrent and charge status LED's. This information was transmitted on the Tx frame to the ILSS node via the LIN bus. The LDF predefined the flash rates and duty cycles of both LEDs.

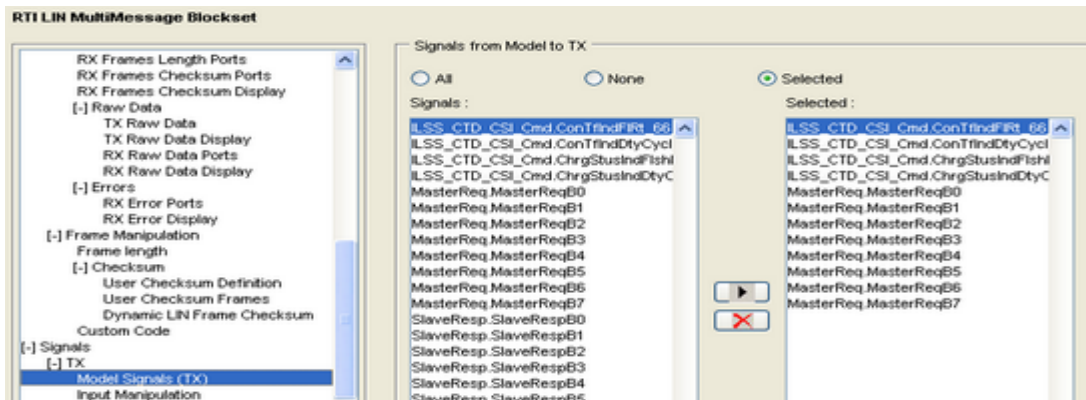


Figure 34 - Tx Signals

The Rx signals, shown in Figure 35, contain the raw data environmental information that was sent on the Rx frame from the ILSS node to the BCM. The LDF defined a specific range that can be detected by these signals. In extreme cases, if the sensor detects a number out of the predefined range, the sensor will enter a sleep mode in order to preserve the integrity of the internal circuitry of the sensor.

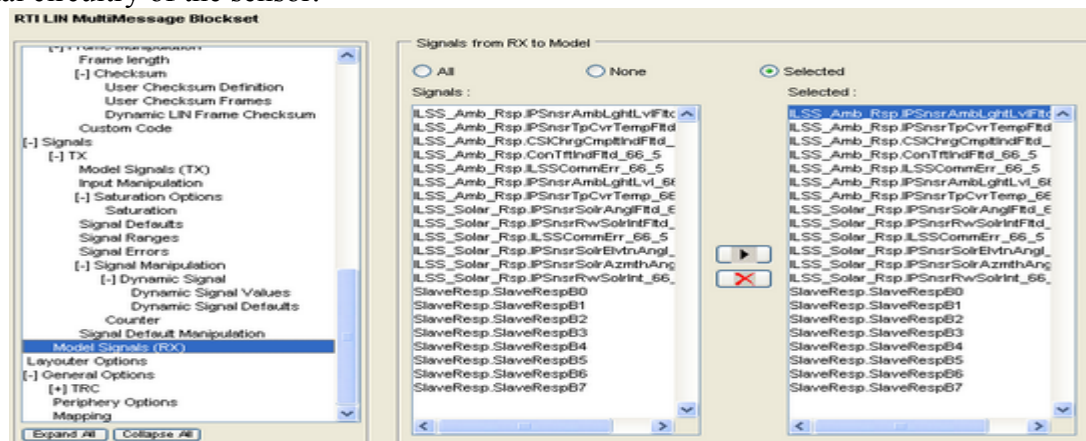


Figure 35 - Rx Signals

Once the Rx signals were specified in the main setup block, the team then chose which signals were to be viewed by Control Desk. The Rx signals chosen were the ones coming out of the Simulink bus selector block to the right side of Figure 26. These signals consist of the ambient light, top cover temperature, solar intensity, LIN propagation speed of both the ambient and solar frames, and the total run time. These signals were attached to a terminator block which allowed the real values of the output signals to be terminated to prevent Simulink warning issues. Any signals not chosen here would be unable to be viewed in Control Desk.

The four constant blocks to the left side of Figure 26 are external inputs to the LIN controller. These are used to either increase or decrease, through Control Desk manipulation, the flash rate and duty cycle of the theft deterrent and charge status LEDs. In order for these external inputs to alter the desired signals, they had to be mapped to the Tx signals pre-defined in the LDF which can be seen in Figure 36 below. This was done by selecting the required Tx signal from a Simulink bus selector block. Once mapped, the external inputs can be altered in Control Desk to verify the proper functionality of the LED of the ILSS.

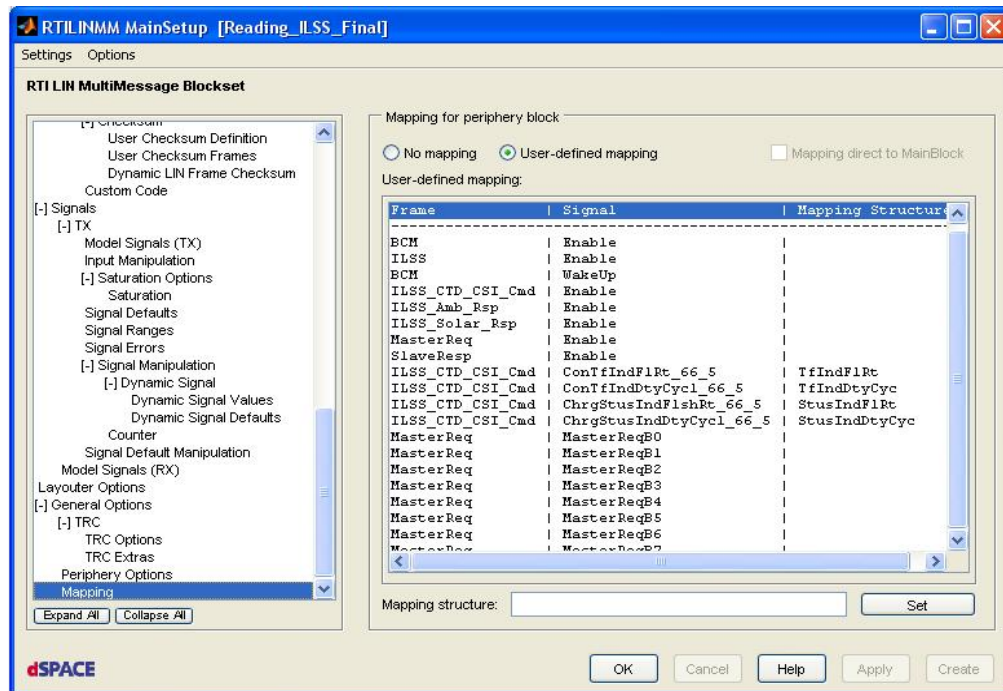


Figure 36 - Tx Signal Mapping

3.2.3: RLHS LIN Driver

The RLHS LIN driver's design is very similar to the ILSS design. The difference is that the RLHS does not need an external input to control an LED since it does not have any external indicators. For this reason, the RLHS was a simpler design which can be seen in Figure 37 below. This LIN driver has the ability to read the relative humidity level, relative humidity temperature, ambient light level, forward looking light level, sensor glass temperature, and signal propagation speed from the master to each slave node.

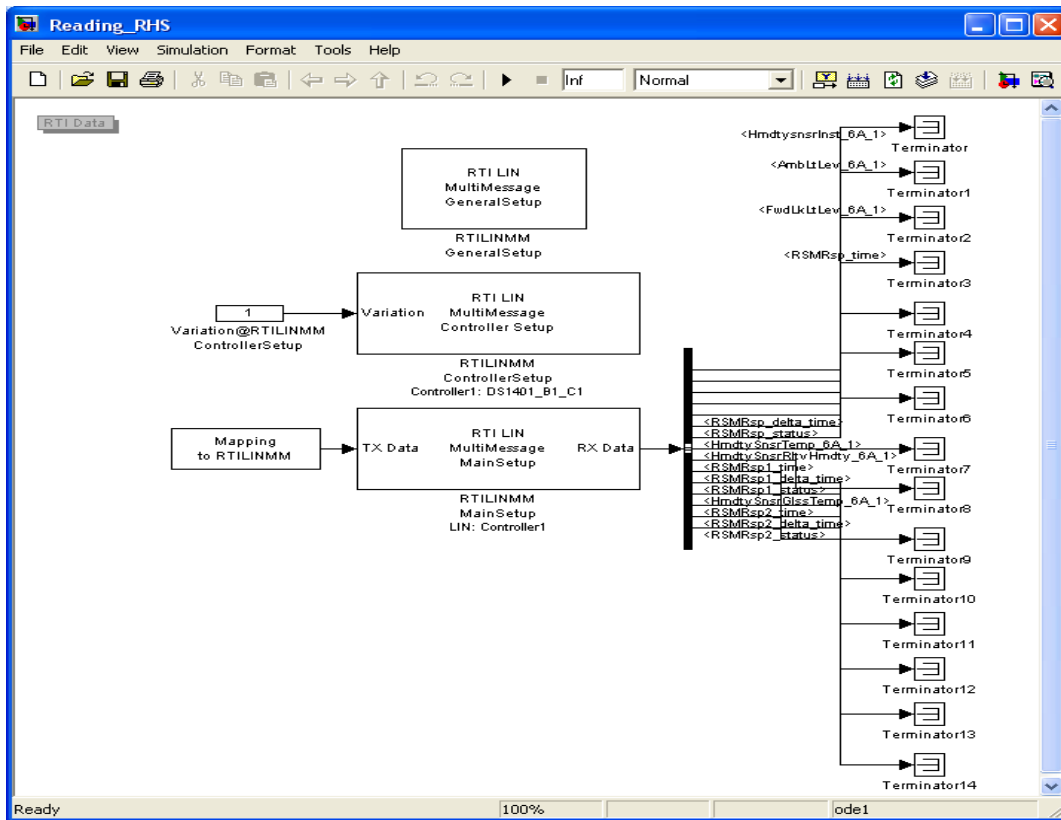


Figure 37 - RLHS Simulink LIN Driver

In a similar manner than the ILSS in Figure 31, the team chose the LDF written for the RLHS. Inserting this LDF in main setup block uploaded all the information into a tabular hierarchy structure that allowed the team to visualize the contents of the LDF for the RLHS.

Figure 38 allowed the team to preselect which nodes were to be in communication in the LIN driver. The Rain Sensing Module (RSM) node contained all the frames and signals necessary to read the desired environmental characteristics that the RLHS is capable of reading. For this reason, the team chose this particular node. The team also chose the BCM node which was used as the LIN master node. This node specifies the schedule table and sends the initial wake-up frame to the RSM node which can be seen in Figure 39. Like before, the schedule table source was set to be included in the .trc file generated. Once again, due to the limited design the team was able to complete in this project, the other nodes seen in the figure below were not used.

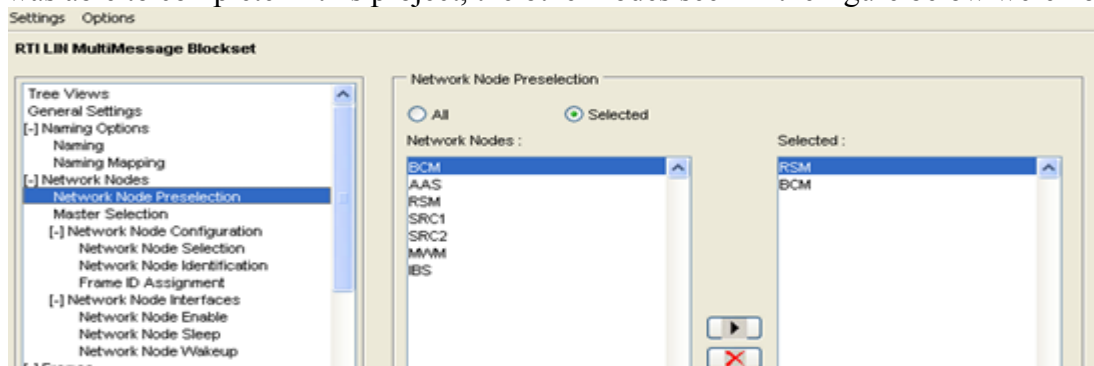


Figure 38 - RLHS Network Node Preselection

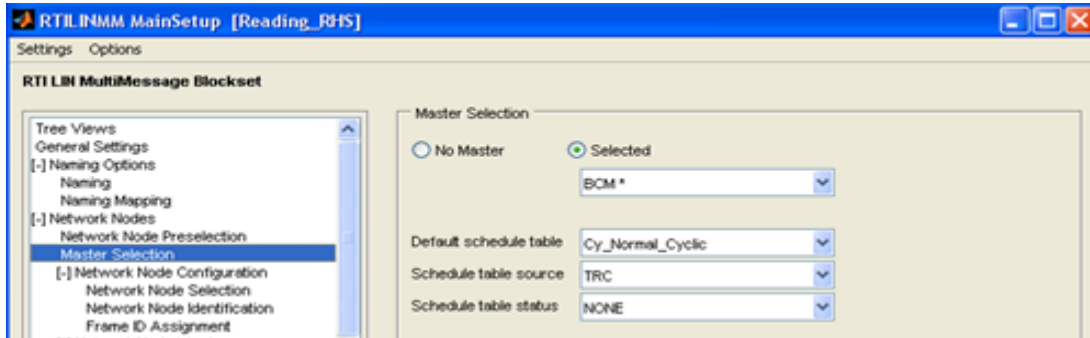


Figure 39 - RLHS Master Selection

The Tx Frame's tab was used to specify what frames were to be transmitted by the BCM node to the RSM node. The RSMCmd frame selected in Figure 40 contains the address information for three slave nodes that contain the desired environmental signals the RLHS is capable of reading. The MasterReq and SlaveResp were also selected as Tx frames for the same reason as the ILSS Tx frames stated in the previous section 3.2.1.

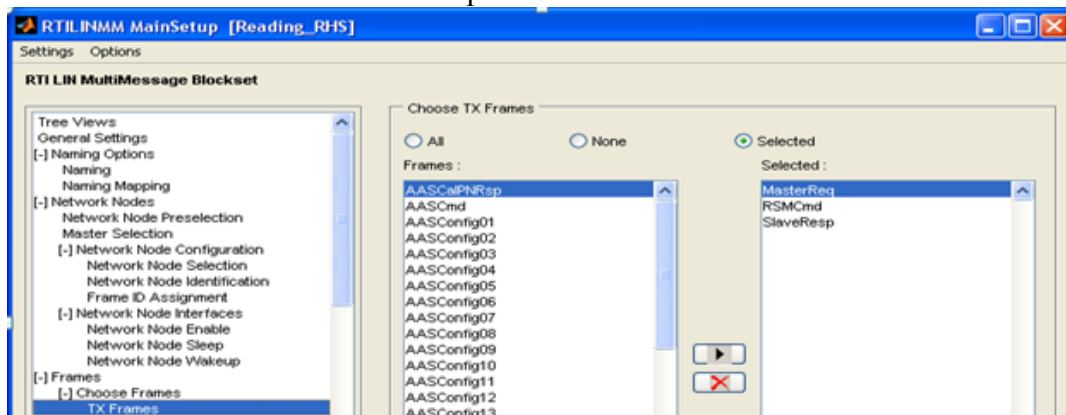


Figure 40 - Tx Frames

The Rx Frame's tab shown in Figure 41 was used to specify what frames were to be transmitted by the RSM node to the BCM node. The frames selected (RSMRsp, RSMRsp1, and RSMRsp2) contain different raw data information collected from the ambient and humidity detectors on the RLHS. The LDF specified specific addresses for each of the three RSM slave nodes. The master can then request this information through the RSMCmd Tx Frame. With this frame enabled, the RSMRsp's frames were then able to transmit the raw data signals from each node on the Rx frames. The rate at which the BCM node received the raw data signals was also set according to the frame position in the schedule table. The SlaveResp frame selected was used to provide diagnostics information as explained earlier.

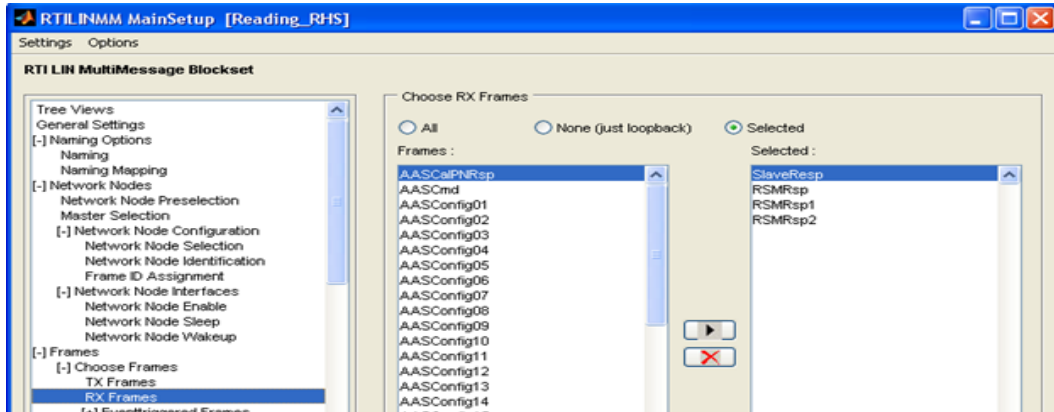


Figure 41 - Rx Frames

The Tx signals selected in Figure 42 below contain information that the BCM node transmitted to the RSM nodes. This information was transmitted on the Tx frame to the RSM nodes via the LIN bus.

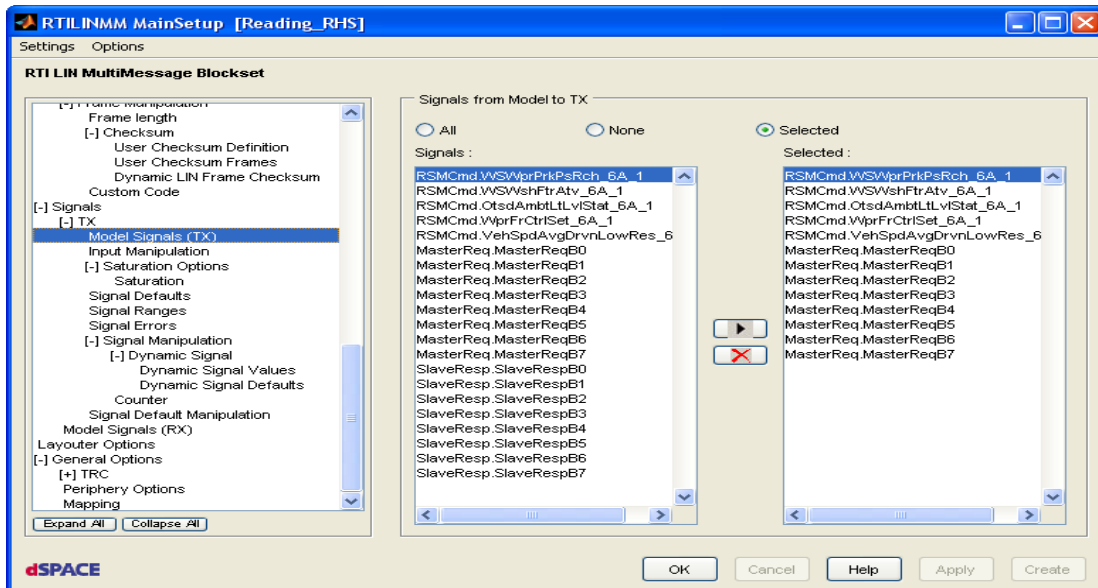


Figure 42 - Tx Signals

The Rx signals shown in Figures 43A-B contain the environmental raw data information that was sent on the Rx frame from the RSM node to the BCM. Like the ILSS, the LDF defined a specific range that can be detected by these signals. In extreme cases, if the sensor detects a number out of the predefined range, the sensor will enter a sleep mode in order to preserve the integrity the internal circuitry of the sensor.

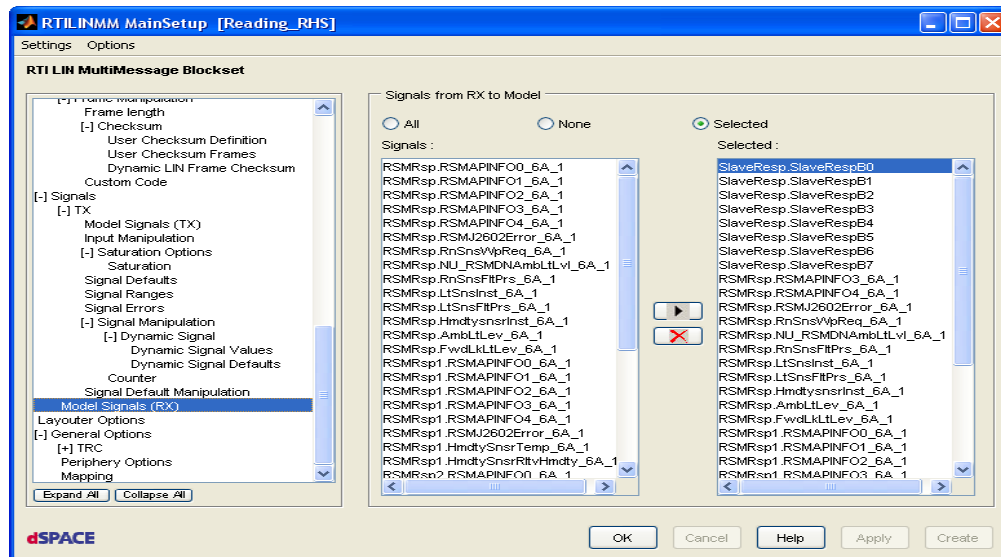


Figure 43 A - Rx Signals

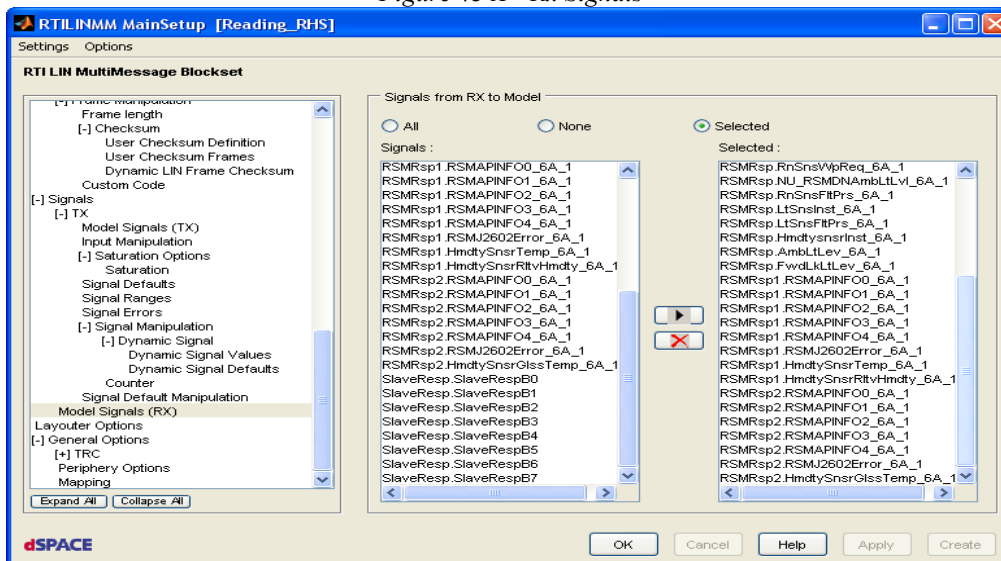


Figure 43 B - Rx Signals

Once the Rx signals were specified in the main setup block, the team then chose which signals were to be viewed by Control Desk. The Rx signals chosen were the ones coming out of the Simulink bus selector block to the right side of Figure 37. The signals consist of the ambient light level, forward looking light level, sensor temperature, relative humidity, and the windshield glass temperature, LIN propagation speed of each RSM slave node, and the total run time. These signals were attached to a terminator block which allowed the real values of the output signals to be terminated to prevent Simulink warning issues. Any signals not chosen here would be unable to be viewed in Control Desk.

3.2.4: Actuator LIN Driver

The actuator LIN driver works similar to the ILSS driver LEDs in that it needed an external input in order to respond and be manipulated. The external inputs to the left side of Figure 44 were used to initiate and set the current position, future position, speed, and direction. The LIN actuator works by having its current position (CPOS) stored in Random Access Memory (RAM). Based on the user set point, it will be told a future position (FPOS) and count up until it reaches the future position. The opposite will happen if the FPOS is set to a lower address than the CPOS. For the Air Inlet and Air Delivery actuators, the speed can change between 3, 4 and 5 rotations per minute (rpm). These actuators are also allowed to rotate clockwise or counter clockwise by setting the direction register to a 0 or a 1 respectively.

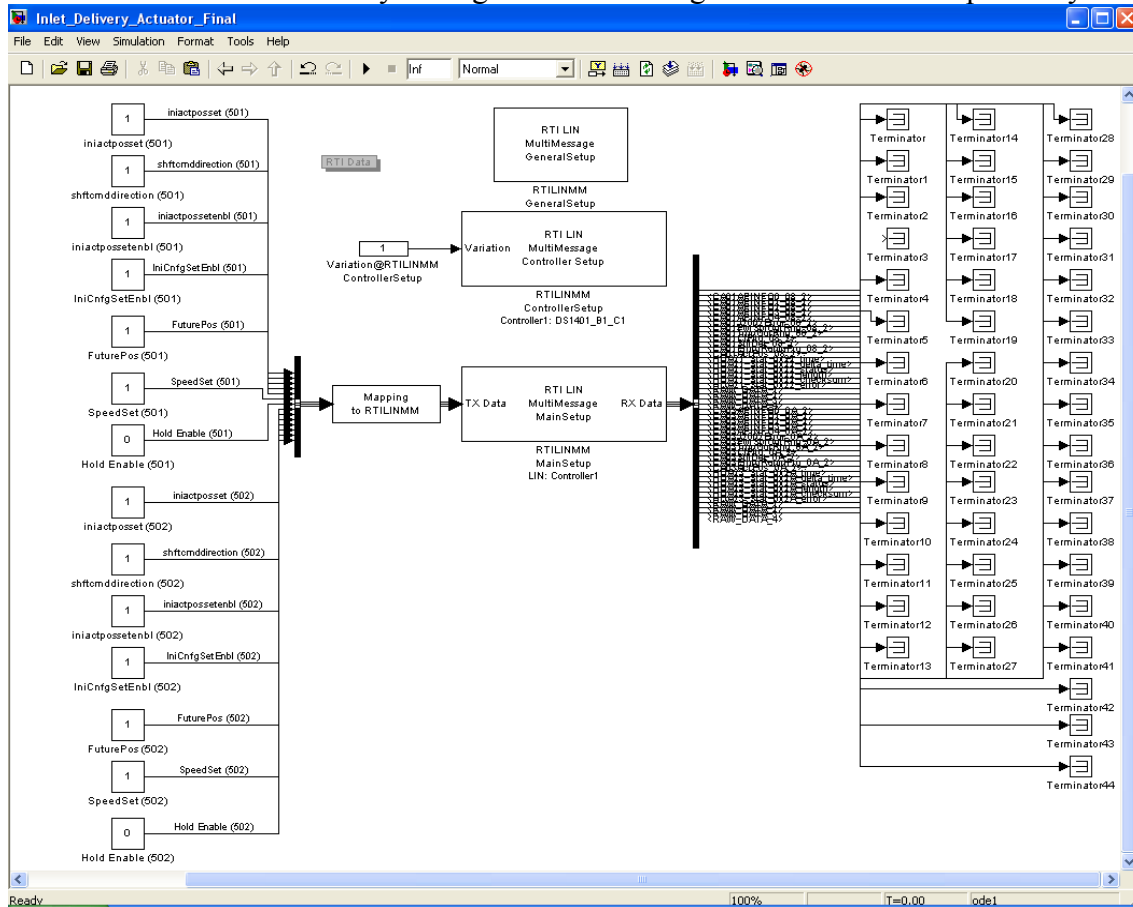


Figure 44 - Actuator Simulink LIN Driver

In a similar manner than the ILSS and RLHS LIN Simulink drivers, the team chose the LDF written for the actuators. This LDF has the capability of driving up to four actuators but can be expanded for full scale implementation. The team only needed to use the nodes for the two actuators received by GM from this LDF. To program the actuators not received in this project, GM must simply select all the other nodes not selected in Figure 45, which correspond to the blower motor, evaporator, heater core, and temperature blend flaps. Inserting this LDF in main setup block uploaded all the information into a tabular hierarchy structure that allowed the team to visualize the contents of the LDF for the actuators. As in the previous LIN drivers, the master node, schedule table, and schedule table source were set as shown in Figure 46 below.

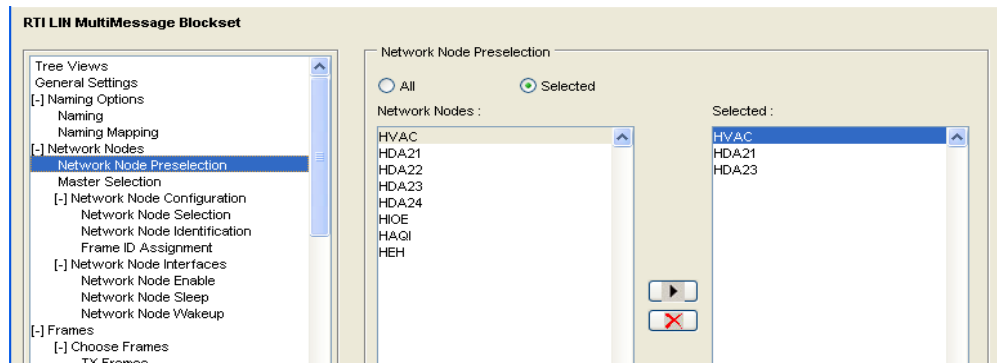


Figure 45 - Network Node Preselection

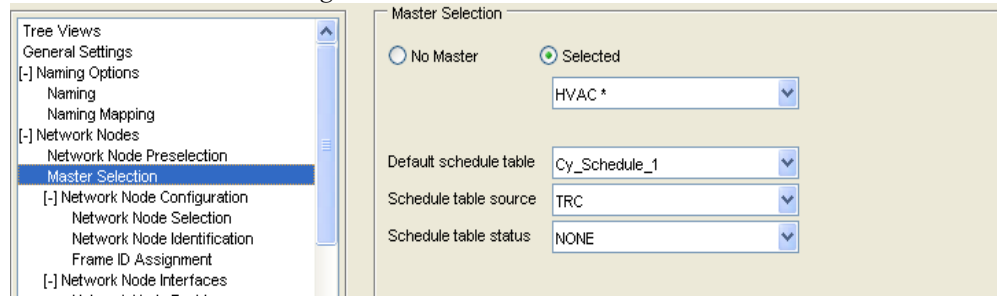


Figure 46 - Master Selection

The Tx frames, shown in Figure 47, are frames sent by the master HVAC node to the actuator slave nodes. These frames carry the initiation and set signals that are used for enabling the slave nodes and setting the actuators FPOS, speed, direction, and other settings that were to be used when integrated with the sensors. The HDA22 and HDA24 nodes, as seen below, are available by grounding the address pin 3 on the LIN actuator in order to resolve a node address confliction in the LDF. Because the team was unable to combine the LDFs, the HDA21 and HDA23 ungrounded address pin nodes were sufficient.

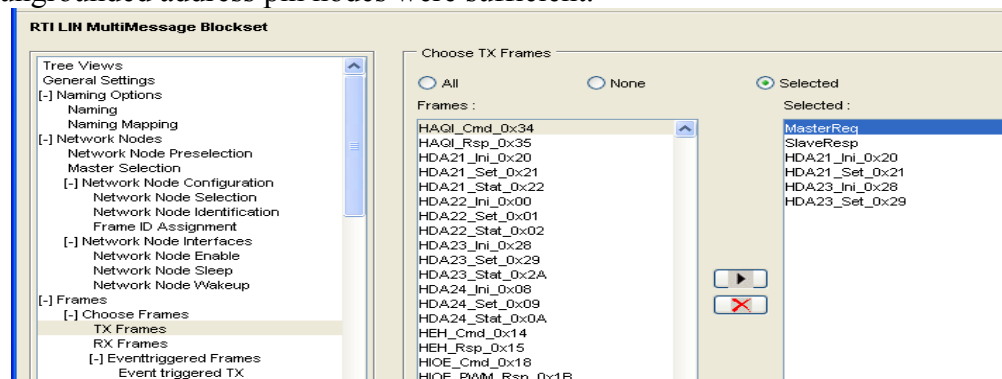


Figure 47 - Tx Frames

The Rx frames, shown in Figure 48, are frames sent by the actuator slave nodes to the master HVAC node. These frames were used to carry the actual position of the user set CPOS, speed, and direction that the actuator is at in real-time.

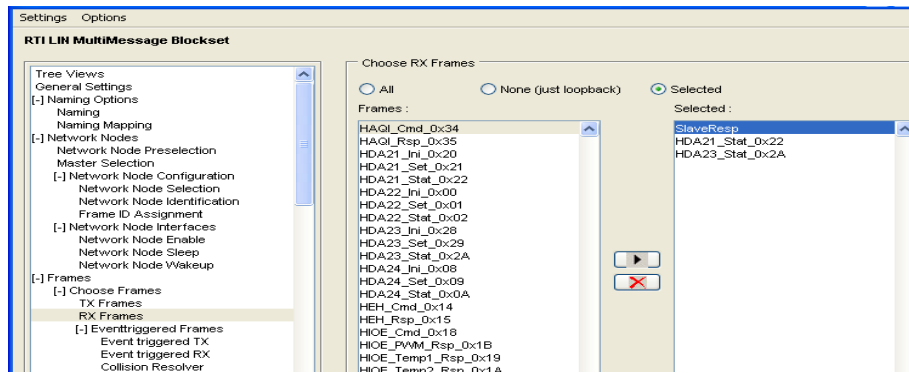


Figure 48 - Rx Frames

The options in Figure 49 below were very important in specifying how the Tx frames would be controlled. By setting the Ini and Set frames to signals as seen below, the team was able to manipulate the signals through the external inputs via Control Desk. The external input blocks to the LIN driver would not be able to be manipulated if the Ini and Set frames were set as a raw data input. A description of what happened when set to raw data will be explained in Chapter 4.

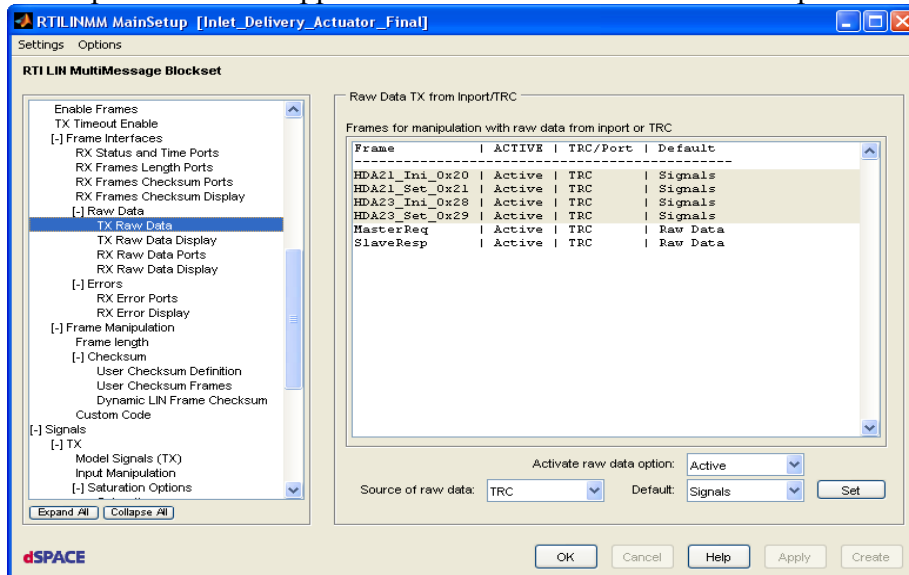


Figure 49 - Tx Raw Data

The Tx signals selected in Figure 50 below contain information that the HVAC node transmitted to the actuator HDA slave nodes. This information was transmitted on the Tx frame to the HDA nodes via the LIN bus. These signals contain information for both the Air Inlet and Air Delivery actuators such as if the node is enabled, the desired FPOS, speed, and direction.

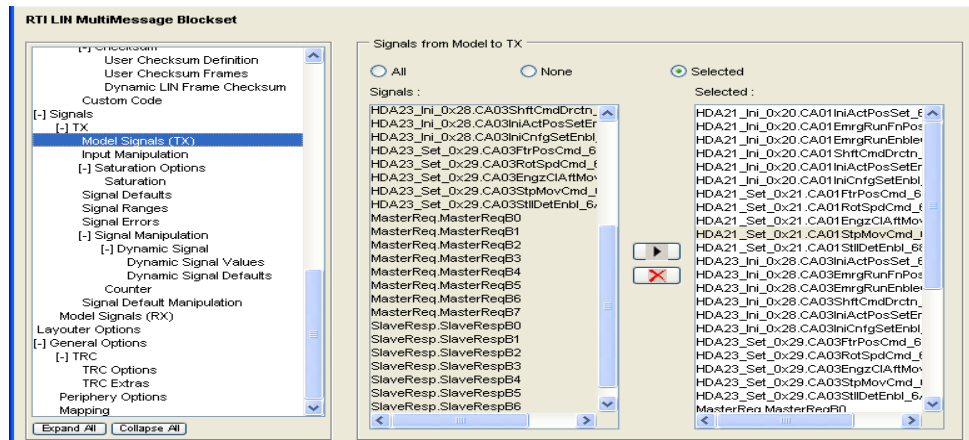


Figure 50 - Tx Signals

In order for the actuator to saturate at a specific value, the address signal ranges were set according to the predefined ranges in the LDF. This can be seen in Figure 51.

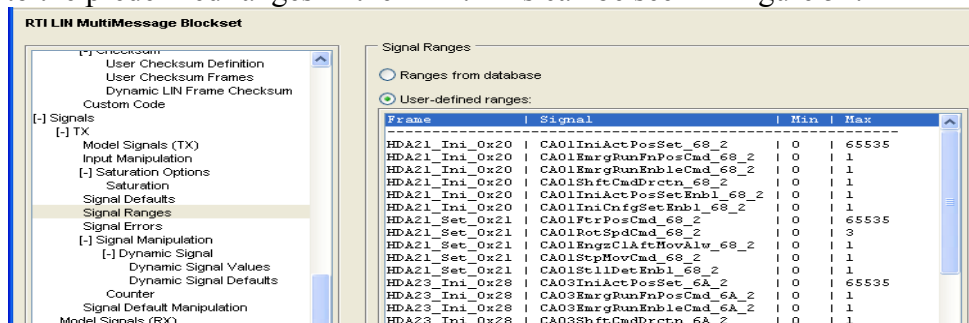


Figure 51 - Signal Ranges

Figures 52 A-B were used by the team to choose the desired Rx signals. The main Rx signal of interest to the team was the ActPos signal which is user to see the actual position of the actuator as it is trying to reach the FPOS. The rest of the signals were used for diagnostics such as the rate of signal propagation and error analysis if a fault occurred. The slave response bits 0-7 were also chosen for diagnostics as explained earlier.

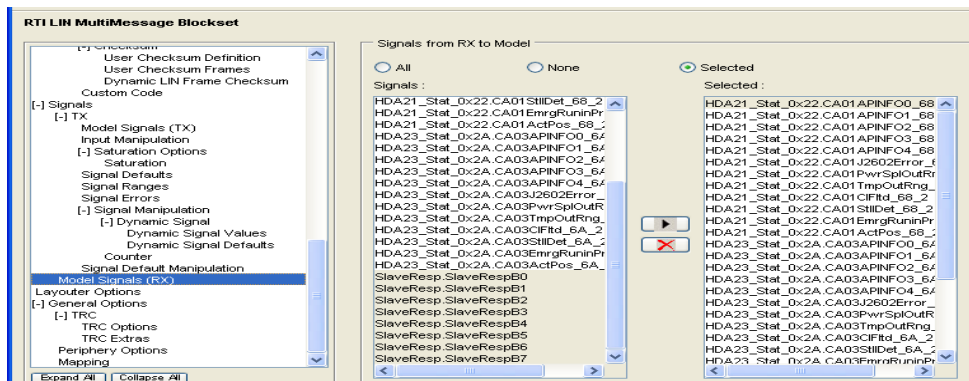


Figure 52 A - RX Signals

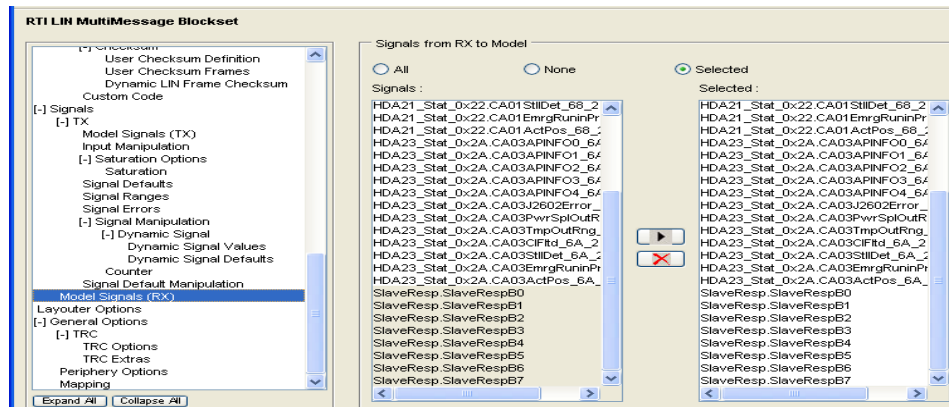


Figure 52 B - RX Signals

Once the Rx signals were specified in the main setup block, the team then chose which signals were to be viewed by Control Desk. The Rx signals chosen were the ones coming out of the Simulink bus selector block to the right side of Figure 44. The signals consist of the actual position for each actuator, LIN propagation speed of both actuators, the total run time, and signal diagnostics for fault analysis. These signals were attached to a terminator block which allowed the real values of the output signals to be terminated to prevent Simulink warning issues.

The constant blocks to the left side of Figure 44 are external inputs to the LIN controller. These are used to either increase or decrease, through Control Desk manipulation, the CPOS, FPOS, speed, direction, and enabling of the actuator. In order for these external inputs to alter the desired signals, they had to be mapped to the Tx signals pre-defined in the LDF which can be seen in Figure 53 below. This was done by entering selecting the required Tx signal from a Simulink bus selector block. Once mapped, the external inputs were altered in Control Desk to verify the proper functionality of the actuator.

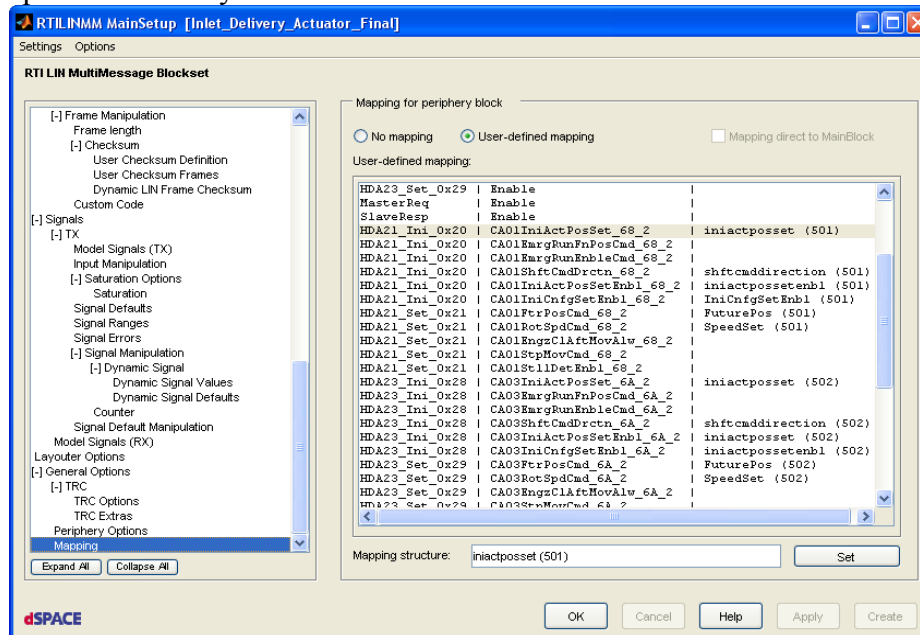


Figure 53 - Signal Mapping

Chapter 4: Test Data

The teams design for the RCP climate control system is built virtually. This approach allows for the simulation and manipulation of different LIN drivers very easily. This increases the efficiency of the team while testing our automating system design. Before the design of the fully integrated system can be completed, it is critical for the team to test individual sections of the system. Ensuring each segment is working properly is vital before any assembly takes place. The three main sections tested in this chapter were the ILSS, and RLHS, and the actuators. Each individual section was tested separately in order to allow for quick trouble shooting as well as giving the team the ability to spread the work among individuals. All data that was collected was condensed for analysis, interpretation, and to allow changes to reconfiguration of the LIN driver for better yields. The following sections include the component testing through Control Desk.

4.1: Testing of the ILSS

When testing the ILSS, the team considered all possible cases. The team noted the difference in solar azimuth and elevation angles, ambient light levels, temperature, solar intensity, and LED operation. The following cases show the operation of the ILSS through Control Desk. Figure 54 below shows the ILSS in its equilibrium state which is nothing interfering with the sensor.

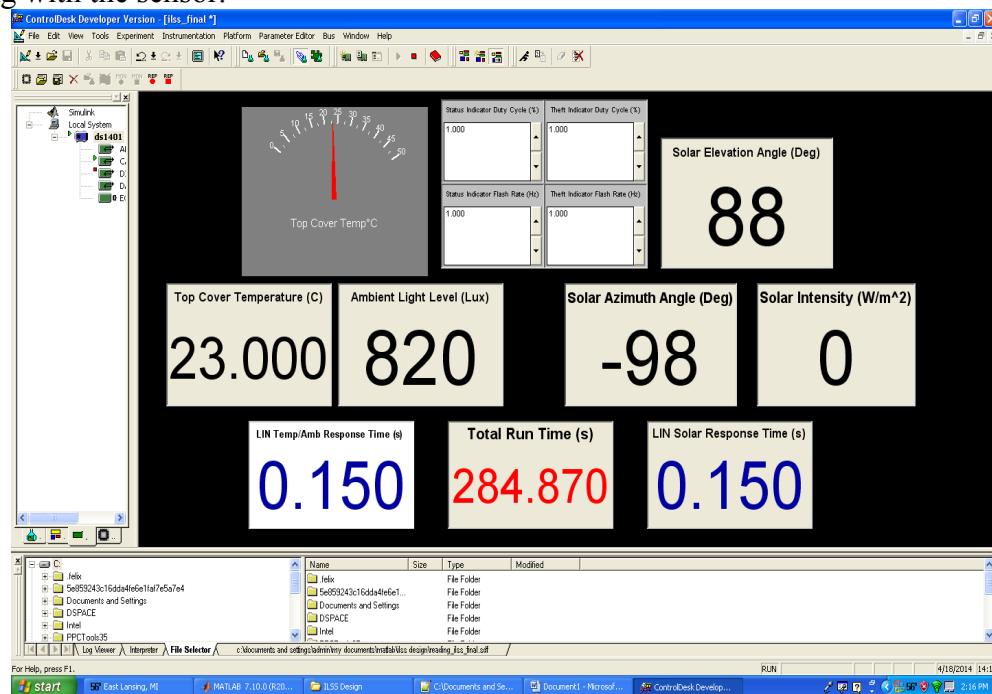


Figure 54 - ILSS in Equilibrium

Table 7 below makes it easy to see the data collected in the figure above in the equilibrium state.

Table 7 - ILSS Test Values

Solar Intensity (W/m^2)	Solar Elevation (Deg.)	Solar Azimuth Angle (Deg.)	Top Cover Temperature (C.)	Ambient Light (LUX)
0	25.877	-98	23 C	820

At equilibrium the sensor reads the current values with no user interference. An example of no user interference could be at startup in a vehicle or while the vehicle is stationary. Also, one of the requirements for this design project was to verify that signal propagation delay speeds for the ILSS nodes were within tolerance as defined in the specification. The LIN signal propagation speed, as seen above in Figure 54, is 150 milliseconds. The ILSS specification defines a total of 120 milliseconds for the startup and continuous time averaged signal response delay. The simulated values have a 30 millisecond slower response time which corresponds to a 25% error. The error calculation was done with the following calculation in Equation 1:

$$\text{Error} = \frac{\text{Measured Values} - \text{Specification Values}}{\text{Specifications Values}} * 100\% \quad (1)$$

This large error could be due to the 8 ft. long LIN wire harness that was used for this sensor or the team simply using the larger schedule table that includes information not used in this project.

The next test, shown in Figure 55, was to cover the top cover of the sensor to decrease the ambient light. It can be seen by comparison with Figure 56 that the ambient light level has been greatly reduced.

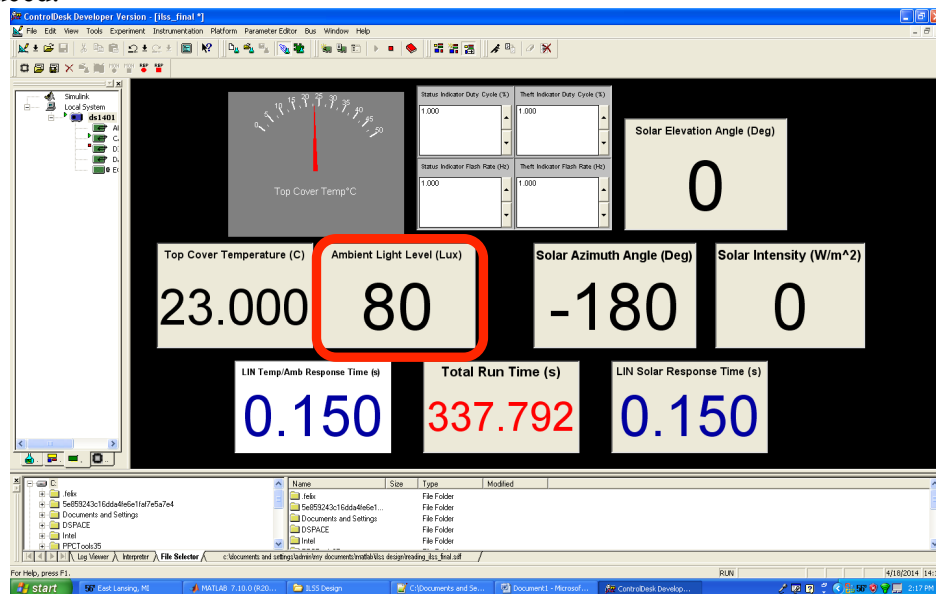


Figure 55 - ILSS Ambient Response

Table 8 below makes it easy to see the data collected in the figure above for the covering test.

Table 8 - Covered Sensor Values

Solar Intensity (W/m^2)	Solar Elevation (Deg)	Solar Azimuth Angle (Deg)	Top Cover Temperature (C)	Ambient Light (Lux)
0	0	-180	23	80

Covering the top cover reduces the ambient light of the sensor. This test also ensured that the ILSS top cover temperature Tx signal in the LIN driver is working properly.

The next test, shown in Figure 56, was to shine a flashlight on the top cover of the sensor to mimic the solar intensity and solar angles. It can be seen by comparison with Figure 55 that the solar intensity level was increased and the solar angles changed when changing the position of the light source.

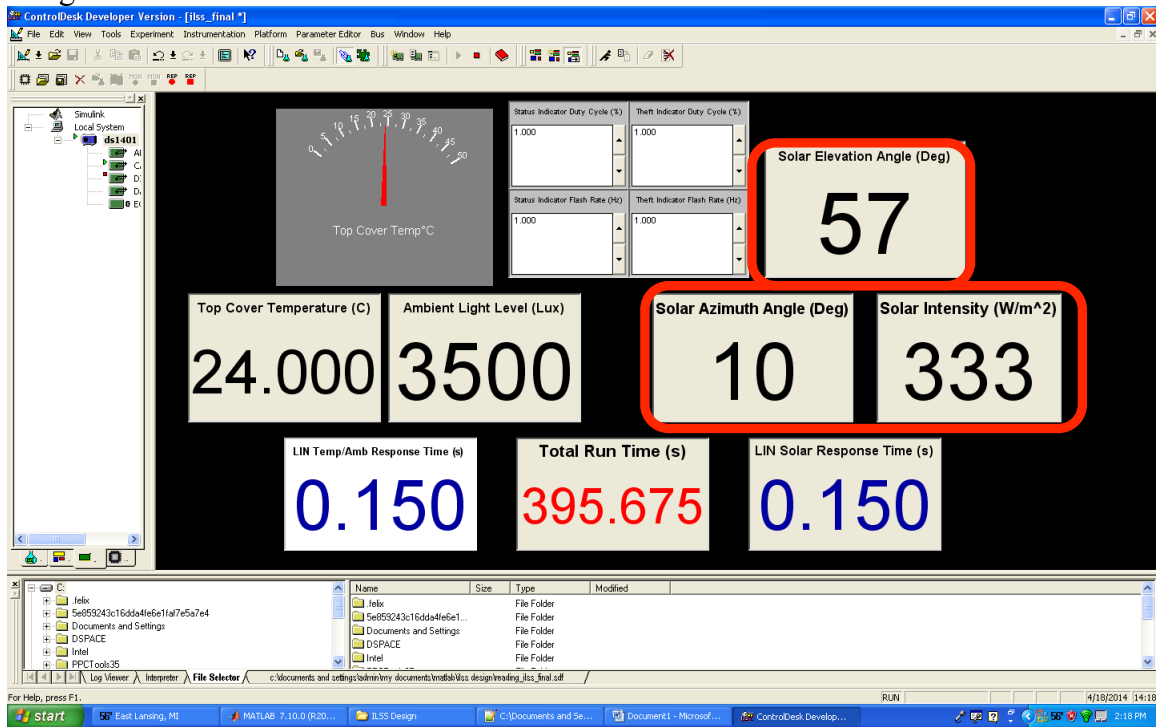


Figure 56 - ILSS Flash Light Test

Table 9 below makes it easy to see the data collected in the figure above for the flash light test.

Table 9 - Flashlight Test Values

Solar Intensity (W/m ²)	Solar Elevation (Deg)	Solar Azimuth Angle (Deg)	Top Cover Temperature (C)	Ambient Light (Lux)
333	57	10	24	3500

To account for the different intensities of the sun, the team used a flashlight in comparison to measure possible solar elevation and intensity readings. This proves the solar node in the LIN driver is functioning properly.

The next test, shown in Figure 57, was to verify the working functionality of the theft deterrent and charge status LEDs. The highlighted section includes the duty cycle and flash rates for the charge status and theft deterrent of LEDs. When both parameters were set to zero for the charge status indicator, the red theft deterrent LED would only flash, as shown in Figure 58. When both parameters of the theft deterrent LED were set to zero and the charge status parameters were set to values greater than or equal to one, the green LED would only flash, as shown in Figure 59. The values of the numeric inputs Control Desk blocks needed to be set to a high number in order to record the LEDs being on. The default operating speed is 1 Hz but it was too difficult to record the LEDs being on at this frequency.

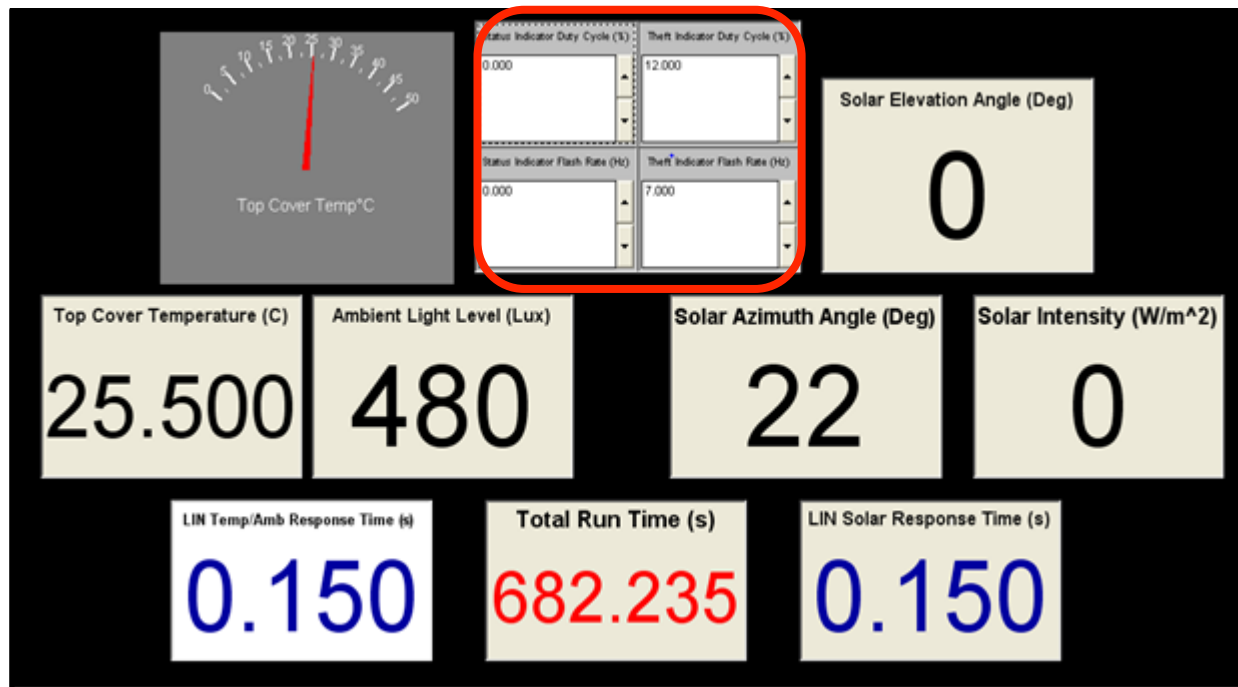


Figure 57 - ILSS LED Testing

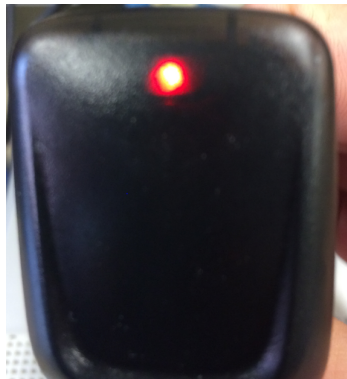


Figure 58 - Theft Indicator LED

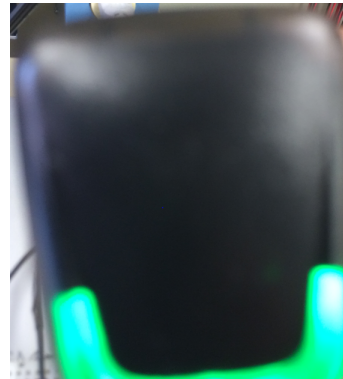


Figure 59 - Charge Status LED

As the figures progressed through this ILSS testing section, it could be seen that the top cover temperature increases with time. Due to the fact that sensing mechanism is deep within the sensor, it is not easily manipulated by physically covering or touching the sensor. To prove this aspect of the sensor was functioning correctly, the team let the system run for a long time to heat up the internal components of the sensor thus increasing the temperature. This testing of the ILSS proves the functionality of the teams LIN driver with respect to real-time recognition of signals.

4.2: Testing of the RLHS

When testing for the RLHS, the team tested for several variables. These included the humidity settings (moisture levels), ambient light conditions, temperature, and temperature of a windshield. Figure 60 below shows the RLHS in its equilibrium state, which is nothing interfering with the sensor.

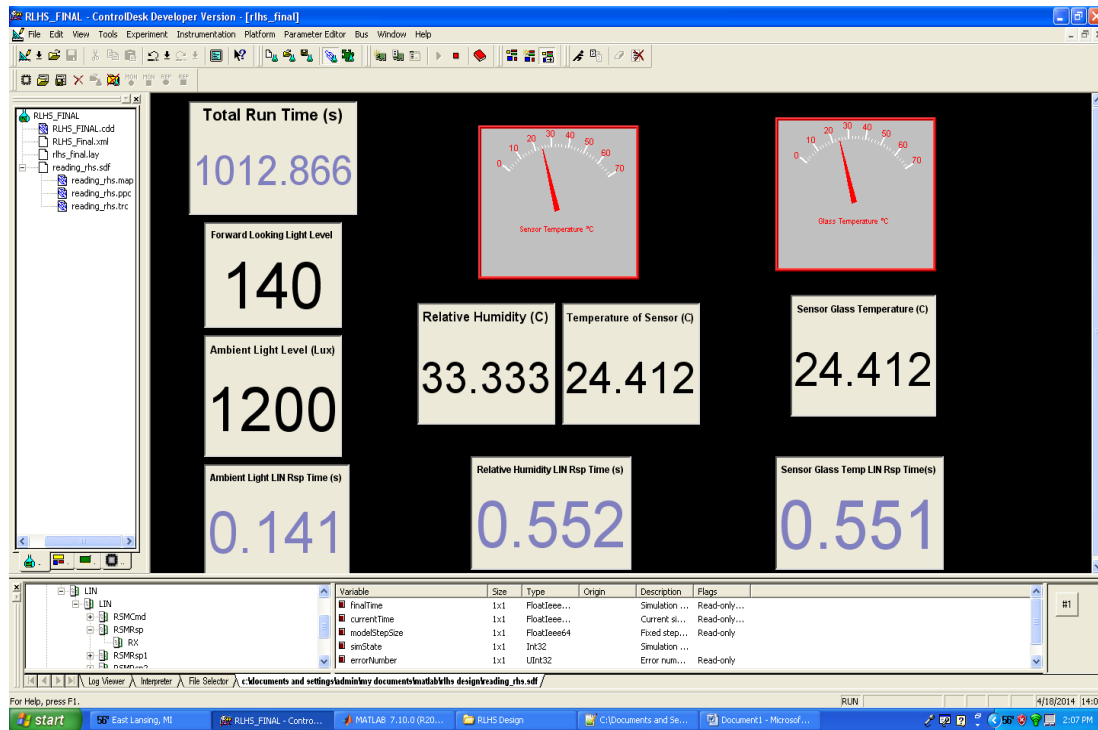


Figure 60 - RLHS in Equilibrium

Table 10 below makes it easy to see the data collected in the figure above for the equilibrium test.

Table 10 - RLHS Test Values

Glass Temperature (C)	Ambient Light (Lux)	Forward Looking Light (Lux)	Relative Humidity (%)	Temperature (C)
24.412	1200	140	33.333	24.412

The test above provides the environmental values of the natural state of the sensor without interference. An example of no user interference could be at startup in a vehicle or while the vehicle is stationary. The LIN signal propagation speed, as seen above in Figure 60, is 150 milliseconds for the ambient response and 552 milliseconds for the temperature and humidity responses. The RLHS specification specifies a minimum of 10 milliseconds and a maximum of 500 milliseconds for the startup and continuous time averaged signal response delay for the RLHS signals. However, it is understood that the interval of this message will need to have a minimum update rate of 200 milliseconds to meet system performance requirements. The simulated ambient response signal delay is faster than the specification delay minimum tolerance by 61 milliseconds, which is well below the delay tolerance. The temperature and humidity response delays are simulated at 552 milliseconds which is slower than the maximum signal response tolerance by 52 milliseconds which correspond to a 10.4% error. This was also calculated by using Equation 1. This large error could be due to the team using the larger schedule table that includes information not used in this project.

The next test, shown in Figure 61, was to cover the top cover of the sensor to decrease the ambient light and forward looking light level. It can be seen by comparison with the equilibrium state that the ambient light level and forward looking light level have been greatly reduced.

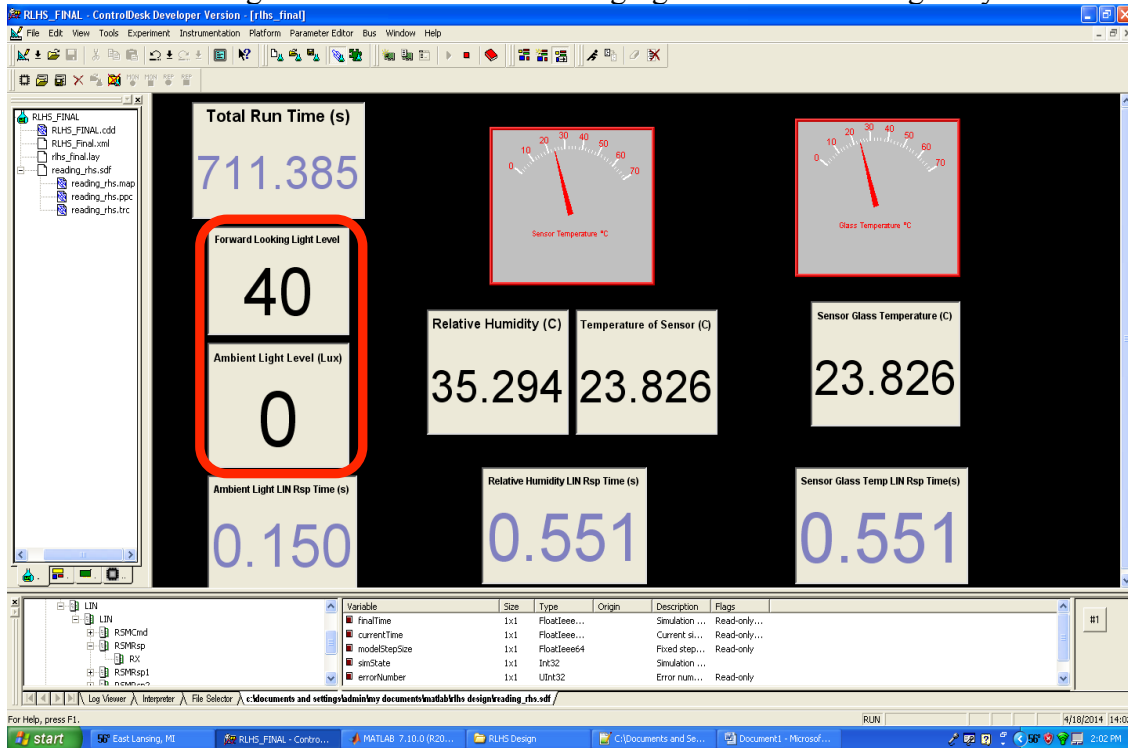


Figure 61 - Covering the RLHS

Table 11 below makes it easy to see the data collected in the figure above for the covering test.

Table 11 - Sensor Covered Results

Glass Temperature (C)	Ambient Light (Lux)	Forward Looking Light (Lux)	Relative Humidity (%)	Temperature (C)
23.826	0	40	35.924	23.826

When conducting this test, the team tried to simulate environmental settings in dusk/night scenario when there is little to no light available. The team covered the sensor with our hands which results in a zero ambient light response and a very low forward looking light level.

The next test, shown in Figure 62, was to touch the bronze strip of the RLHS which contains the sensing mechanism to read the temperature. It can be seen by comparison with equilibrium state that the temperature has increased.

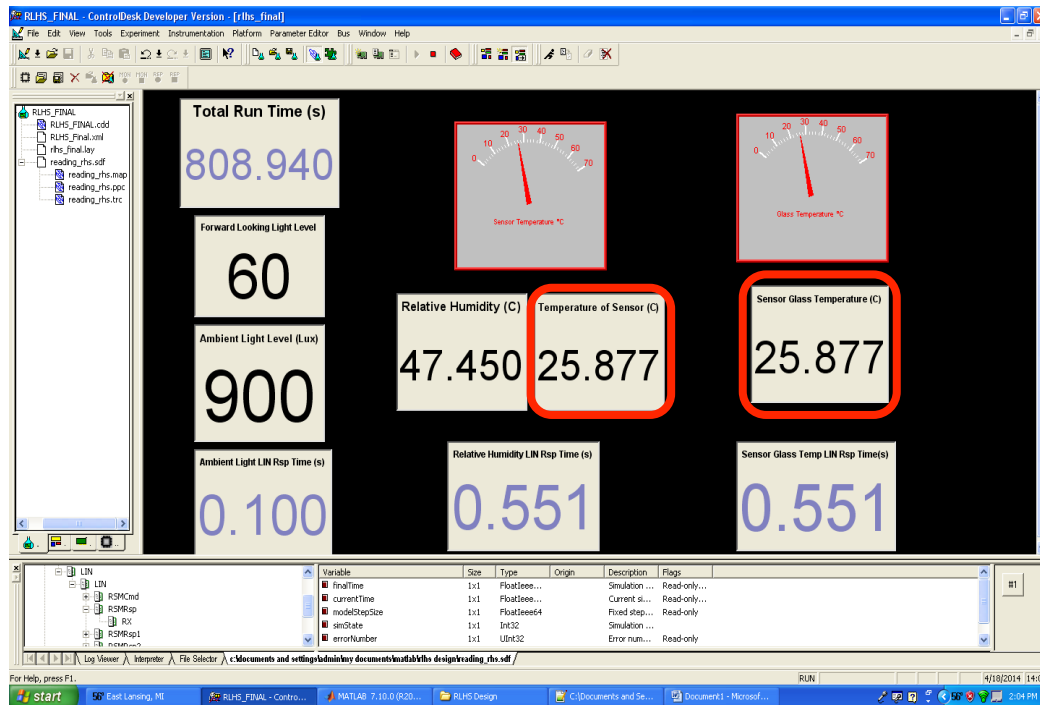


Figure 62 - Increasing Heat to the RLHS

Table 12 below makes it easy to see the data collected in the figure above for the temperature increase test.

Table 12 - Bronze Strip Sensor Results

Glass Temperature (C)	Ambient Light (Lux)	Forward Looking Light (Lux)	Relative Humidity (%)	Temperature (C)
25.877	900	60	47.450	25.877

This test determined the team's body temperature was reasonable to alter the sensor readings.

The next test, shown in Figure 63, was to blow on the bronze strip of the RLHS which also contains the sensing mechanism to read the humidity. It can be seen by comparison with equilibrium state that the humidity was greatly increased when applying a source of moisture.

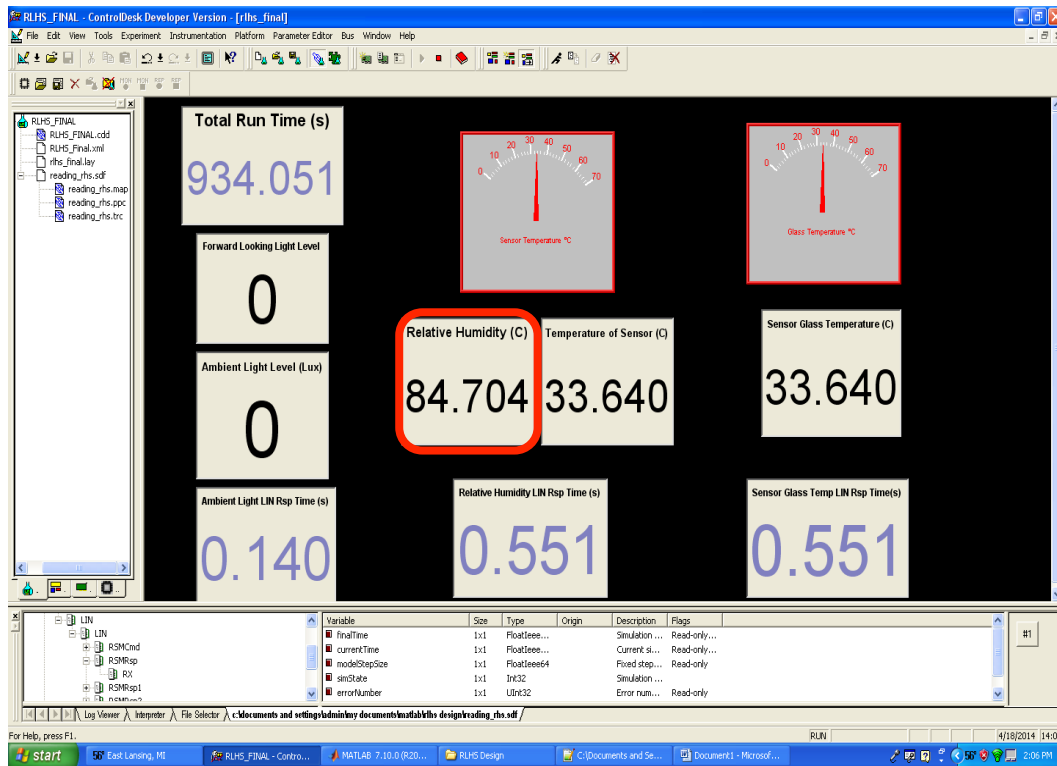


Figure 63 - Blowing on the RLHS

Table 13 below makes it easy to see the data collected in the figure above for the humidity increase test.

Table 13 - Relative Humidity Testing Values

Glass Temperature (C)	Ambient Light (Lux)	Forward Looking Light (Lux)	Relative Humidity (%)	Temperature (C)
33.640	0	0	87.704	33.640

As seen above, the teams testing values for relative humidity change drastically when moisture was applied to the sensing mechanism on the RLHS. These test cases for the RLHS prove the functionality of the teams LIN driver with respect to real-time recognition of signals.

4.3: Actuator Testing

Initial Actuator Testing

Before the Tx raw data tab in the Simulink LIN driver was changed to read Signals as shown in Figure 49, the actuators behaved in completely different manner than the final design. In this design, the actuators only responded to raw data and not to the external input signals. Because there was no raw data being supplied to the actuator, the actuators FPOS address was set to its default address of 0x7FFF corresponding to 32767 counts and proceeded to count down at 3 rpm to the address 0x0000 which corresponds to 0 counts (also the default value for CPOS in this case). After this process concluded and the FPOS address was equal to the CPOS address, the actuator needed a power on reset to repeat that process. A screen shot of the simulation can be seen below in Figure 64 below. For this figure, the Air Delivery actuator was allowed to run all the way down and the Air Inlet actuator was connected immediately when the Air Delivery

actuator was finished to show the start and end positions. The LIN signal propagation speed, as seen below in Figure 64, is about 600 milliseconds for both actuators. The actuator specification defines a 1 second delay time for the startup and continuous time averaged signal response delay. The simulated time delays are below the specification tolerance by about 400 milliseconds.

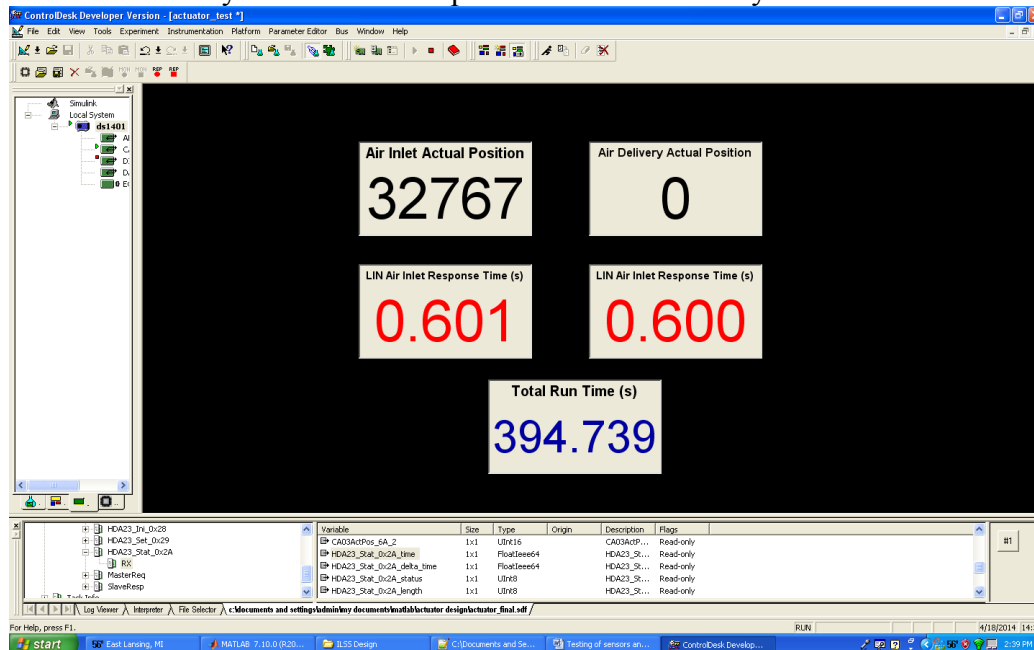


Figure 64 - Actuator Initial Testing

The team experienced some faults in the actuators due to connecting the power and ground pins backward while connected to power. This rewrote the internal assembly programming within the actuators writing the addresses initially programmed by the team to a different fault address. In order to correct this problem, the team had to visit GM and have an engineer re-download the assembly programming. Only then would the actuators be able to properly work again.

Final Actuator Testing

When performing the final tests on the actuators, the team conducted several tests. With the use of Control Desk, the team was able to implement user settings that would be in an automobile.

Enabling signal manipulation allowed the team to send signals to the actuators which modified the actuators response and allowed for testing to occur. The tests performed on the actuators are as follows:

- The first test was controlling the current and future positions of the actuators which allows for the motor to rotate. This was done by either increasing or decreasing the numeric input dSPACE block that was mapped to the external input of the LIN Simulink driver.
- The second test was manipulating the shaft direction by changing the address to 0 or 1 through the numeric input block corresponding to counter and clockwise rotations.

- The third test was to manipulate the speed of the actuator. This was achieved by changing the numeric input block to a 0, 1, 2, or 3 corresponding to 3, 3, 4, or 5 rpm respectively. Address 0 is the default speed of the actuator.

Figure 65 below shows an example test for both actuators with the initial position set to 0, the future position set to a higher address than the current position, the shaft direction set to counter clockwise, and the speed set to 3 rpm with a numeric input of 1.

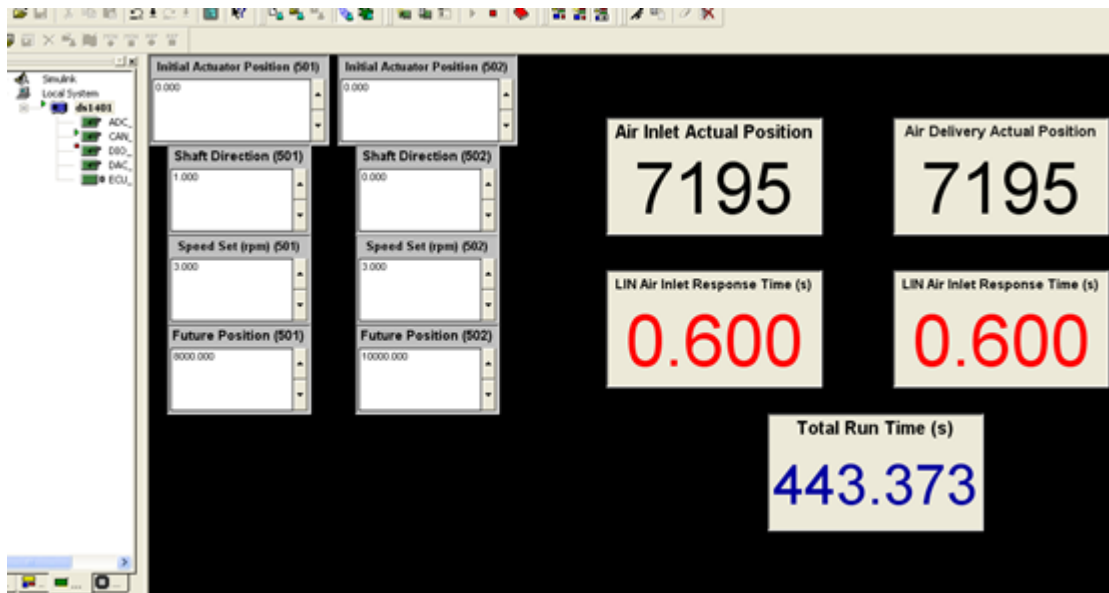


Figure 65 - Actuator Test

Table 14 below makes it easy to see the data collected in the figure above for the actuator test.

Table 14 - Actuator Test Values

Actuator Type	Speed Set (RPM)	Direction	Actual Position (Count)	Future Position (Count)
Air Inlet	5	counter-clockwise	7195	8000
Air Delivery	5	counter-clockwise	7195	10000

When conducting this test, the team tried to simulate the user input of air recirculation by enabling the Air Inlet actuator and the setting (e.g. blowing air to windshield, feet, or face) by enabling the Air Delivery actuator. This test proved that the team was able to simulate a user input through manipulation of the LIN driver.

Chapter 5: Summary

5.1: Summary

The project was a success in the fact that the team was able to create a LIN driver for each of the sensors and the actuators. The team was also able to create individual Control Desk re-usable libraries for each of the LIN drivers. The libraries created were able to read the desired climate characteristics from the sensors and had the ability to drive the actuators. The team was also able to write a technical document that describes the associated functionality and implementation which was an expected deliverable by GM. The greatest success of this design team was the knowledge taken away on the benefits of using an RCP system when designing a vehicle. Also, the team gained knowledge on the use of LIN protocol which is being implemented in the automotive industry today.

The final cost paid out by the team was significantly less than the allotted budget. Unfortunately, the total cost of implementing the RCP system in this project is substantial because of the cost of the ECU and dSPACE software. The final budget can be seen at the end of this chapter in Table 15.

5.2: Successes and Failures

The greatest success for this project was actually being able to design a reusable and updatable LIN driver and dSPACE Control Desk library for the sensors and actuators. This is a positive demonstration that the team could implement what was researched for this project such as LIN protocol, HIL simulation, Simulink block structure programming, and hardware and software integration. GM will be able to use this team's designs and build off of them to prototype their future climate control system in their vehicles. The fact that this design team played a part in making the foundation for this system work is an extraordinary accomplishment that can be used to benefit vehicles, which play a major role in the economy.

The greatest failure in this project was that the team was not able to design a LIN driver that allowed for the simulation of both sensors and the actuators at the same time. Although this was beyond the scope of the deliverables that GM required from this design team, it would have been a great accomplishment that would have allowed the team to utilize the full power of the LIN bus system.

5.3: Suggestions for Future Improvement

A suggested future improvement to this project could be to both sensors and all the actuators integrated into a single LDF. This would allow for the simulation of the multiple devices at once. This would also allow for the sensors to be able to drive the actuators autonomously instead of setting an external constant input for solely testing purposes. This improvement would bring a future design team closer to installing their climate control prototype into a vehicle.

Another suggested improvement would be to acquire the additional actuators that were missing from this project in order to test for the proper functionality of the actuator LIN Simulink driver created by the team. Acquiring these components, such as the temperature blend flaps, blower motor, heater core, and evaporator actuators would ensure that the LIN drivers designed by the team have been developed correctly.

5.4: Final Budget

Table 15 - Final Budget

Product	Description	Qty.	Price/Per	Total
DS821 Express Card	34mm DS821 Host Interface (Express Card) SN#380475	1	\$750	\$750
Dongle	Includes the following software: CDP(Control Desk, RTI,Platform_API), CCPPPC, RTILINMM Blockset	1	\$500	\$500
MATLAB/Simulink 2010a	MATLAB/Simulink License supplied by MSU Engineering	1	\$299.95*	\$299.95*
ECU	MicroAutoBox 1401/1501	1	\$10,000	\$10,000
Actuator	Air Inlet, Air Delivery, & Temperature Blend Flaps	3	\$ 1**	\$ 1**
ILSS Sensor	Indicator, Light, and Solar Sensor	1	\$ 1**	\$ 1**
RLHS Sensor	Rain, Light, and Humidity Sensor	1	\$ 1**	\$ 1**
LIN bus	LIN bus for each sensor and actuator	5	\$20	\$100
Power Supply	40 VDC 10 A Power Supply	1	\$270	\$270
10 A Fuse	10A fuse for component protection	1	\$0.39	\$0.39
Proto Board	Proto Board for external wiring	1	\$3.59	\$3.59
2 x 4	2 in. x 4 in. x 8 ft. #2 Pressure-Treated Lumber for final display	1	\$2.97	\$2.97
Plywood	Project Panels 1/4 in. x 24 in. x 24 in. Pine Plywood for final display	1	\$13.60	\$13.60
Plexy Glass	Plaskolite 8 in. x 10 in. Polystyrene Sheet final display	1	\$2.48	\$2.48
			Total Cost =	\$11,945.98

* Typical price of MATLAB 2010a off the internet

** Denotes GM confidential information that cannot be distributed in this report. The price is presented as one dollar

Table 16 - Budget Spent by the Team

Final Budget the Team Spent	
Hard ware Purchased	\$23.03

Appendix 1: Team Technical Roles

Guiseppe Ferro: Team Manager



My technical roles in this project were the main programming of the ILSS, RLHS, and actuator LIN drivers through Simulink, establishing the hardware connections between the MicroAutoBox and the sensing and actuating components via LIN, and verifying the proper functionality of the drivers by conducting component simulation testing through Control Desk. The software design consisted of configuring the LIN drivers individually for each sensor and actuator. This design strategy was heavily constrained due to the way in which GM gave the team the supporting LDF files, which describe all the aspects of signal transmission for a single sensor or actuator individually. The process of programming each LIN driver was crucial because it laid out the foundation for communication between the ECU and the sensors and actuators. I did this by manually selecting the information from the LDF using the dSPACE Simulink RTILINMM block sets.

For establishing the hardware connections, I wired the MicroAutoBox with power, ground and the LIN channel within the ZIF connector by referring to the dSPACE MicroAutoBox Configuration specification. Using the pins in the ZIF connector, I then connected the sensors and actuators with the correct pin assignments to configure the LIN bus system.

The work I completed in this project took a considerable amount of research. In order to design each LIN driver, I had to thoroughly read through sensor, actuator, and MicroAutoBox specifications to identify which information was necessary. I also had to read through dSPACE Simulink tutorials to learn how to use the RTILINMM block sets worked. I was assisted in the initial setup of the RTILINMM block sets by dSPACE engineers over the phone and in person. The engineers familiarized me with the steep learning curve of the system in order to quickly begin the development of the LIN drivers. The research and supplier interfacing I did played an important role in designing the LIN drivers.

For simulating and testing the responses of the sensors and actuators, I had to learn how to use the dSPACE Control Desk software which was the RCP software that integrates the Simulink LIN driver code with the MicroAutoBox 1401 processor board. This simulation software made it possible for me to test the sensor/actuator responses and conclude that the LIN drivers designed were capable of producing responses that were within the signal tolerance specifications written by GM. In order to make this process easy reusable by GM, I used the Control Desk library feature to save the final layouts created for each component.

Omar Ali: Web-Master



My technical role in the overall design of the project was to create the driver for the RLHS. In order to accomplish this I commenced research on the LIN protocol to understand how nodes are transmitted and received between the master (MicroAutoBox) and slave (RLHS). With the background about the LIN protocol, I was able to go through the LDF that was provided by GM and select the frames that were required in order to send and receive the desired signals for the RLHS. There was, however, a problem with, not only the RLH sensor LDF, but also with the LDF's for the other sensor and actuators while I was working to configure the RLH sensor. After speaking with GM I understood that the LDF's needed to be configured for LIN 2.0. I figured out how to configure the LDF's for LIN 2.0 by

researching about the structure of LDF's. From this I was able to understand that the LIN protocol version and language version needed to be changed from the original "j2606" to "2.0". This configured the LDF for LIN 2.0. Along with this I needed to comment out the j2606 protocol in the master node that was configuring the master for 32 bits at 1% tolerance. The last thing I needed to do was to configure each LIN protocol version for the node attributes to 2.0 and then comment out the response tolerance. This configured the node attributes to be recognized for LIN 2.0. With the LDF configured for LIN 2.0 I was then able to use Simulink to assist in building the driver for the RLHS. In Simulink I needed to use the RTI LIN MultiMessage GeneralSetup, ControllerSetup, and MainSetup blocks that were provided by dSPACE as the base for assisting in setting up the driver for the RLHS. I then configured the blocks by referencing the RLHS LDF that was provided. With the correct nodes configured I was then able to load the LDF in Control Desk and display the required data such as the relative humidity level, relative humidity temperature, ambient light level, forward looking light level, sensor glass temperature, and signal propagation speed using display tools in Control Desk. This satisfied the requirement that was asked for by General Motor's for the RLHS.

Weiham Yan: Webmaster



My portion, as defined in proposal, is actuator interfacing. The first thing is to understand what the sponsor wants: demonstrated dSPACE set-up that drives multiple LIN actuators with associated required diagnostics and actuator with end-stop learning functions. There is no specific constraint except the LIN itself, which means there is only one wire used to transmit data so we cannot use any other protocol, such as CAN. The only criterion is making the stuff I did reliable. Because we only have one authorized computer, we always worked together. We firstly finished the setup of sensors and then turned to the actuators. Specifically, the works on actuators are kind of similar with those on sensors. There are multiple things that needed to be shown in

the interface of the Control Desk. I assisted in selecting input windows to type in the rotation speed for actuators. I assisted in inputting integers here to control the rotation speed at 3, 4 and 5 rotations per minute. Then I assisted in inserting display windows into the interface for

monitoring certain data from actuators such as Air Inlet actual Position, Air Delivery actual position and response time. After finishing the interface of the Control Desk for actuator, I assisted in reloading the LDF file so that we can drag the signals we need into the display and input windows. According to the LDF file the sponsor provided, it was known what types of frame are needed for actuator setup and what nodes are respectively belong to TX and RX. After finishing the Simulink setup, we build the interface in dSPACE Control Desk. By opening Simulink blocks, loading LDF file, drawing display windows and selecting signals, I assisted in being able to read those specific numbers we prefer on the screen and change the rotation speed of actuators by adjusting the constants of Control Desk.

Sitong Ge: Document Preparation



Sitong Ge was the document preparation of Team 4. My role included but was not limited to provide support for professional written communication in the team's engineering climate control setup design process, coordinate the preparation of written reports of, coordinate peer-editing of documents and maintain documentation portfolio.

For the technical role, I researched different type of controller. Because of the scale of our project, we used open-loop control. The two main difficulties for the sensors' setup were the understanding of LIN and the setup of dSPACE. I helped in setting up the ILSS and RLHS drivers. LIN is single master multiple slaves' communication, so we needed set up both the master and slaves as the LDF file defined. I helped with converting LIN J2602 to LIN 2.0 in the LDF file. I read the LDF file to assist in determining the Tx and Rx frames and nodes respectively to the frames used for the sensors' setup block. For the setup of ILSS and RLHS sensors, I investigated the characteristic of LIN, the Tx and Rx frames. The LDF file was important to this project, so reading it and understanding the characteristics of LIN communication were crucial. After the setup blocks were done, the control desk was used to do the testing. The setup of Control Desk was difficult to us, because all of us were new to dSPACE. I helped with the setup of Control Desk. After talking with dSPACE specialists, we had a more clear idea about how to set up dSPACE Control Desk. We also investigated the demo to do the setup. The Setup model needed to be built, and the Control Desk need load the right SDF file. The signals were shown at the bottom of the Control Desk. The signals needed which were preselected by the bus selector needed to be pulled up to the displays on the Control Desk. Then, it should be changed to Animation mode, so we could see the changing of the parameters which were needed. We could also use the Control Desk to control the direction and the speed of the actuators. I also helped with the testing of ILSS sensor, RLHS sensors and the actuators.

Ricardo Johnson: Presentation Preparation



Technical/Non-Technical Roles - I had an active role throughout our entire designing process of our climate-control system. My non-technical role assigned to me was lead in preparing for all presentations conducted during our group project. Comprehending appropriate content and style of oral presentations to relevant standards to explain objectives was my main focus when presenting to our audience. Understanding critical points in our project enabled me to assist my team to illustrate targets from our information and data. Accomplishing this leads to my technical roles in our climate control system which concludes understanding each step of our design. Working with a MicroAutoBox, Simulink, dSPACE and knowing how to constructively connect our MicroAutoBox to a computer was the

first step. Second is the ability to understand how the MicroAutoBox's LIN network communication system communicates to the actuators and sensors which were used to read and determine climate air conditioning control in an automobile. One of my roles during our design process was working with the actuators, helping set up its LIN communication driver and understanding all limitations especially when concluding for our application note. Once our design was complete I had the opportunity of testing which enabled me to understand in full detail the capabilities and limitations of the actuator design. This ensures a well maintained product that yields to any potential failures that could occur. The challenge in evaluating these limitations gives an effective approach ensuring efficiency throughout the lifecycle of the teams design. Once all tests were conducted, comprehending the signals needed to send to the actuator was vital. The actuator undergoes signals sent through user input which could be manipulated and displayed through Control Desk software. Once the signals are received from the Control Desk Software we have full control of the actuators. Control Desk gives the ability to interface as if it was actual user input in an automobile. This is important to ensure all functions of the actuator work appropriately. The Control Desk Software allowed signals sent to determine future position, mode of direction, and speed of rotation of each actuator.

Yingshuyu Li: Lab-Coordinator



Yingshuyu's main technical role in the project was doing research about dSPACE Control Desk Software, LIN, and actuators. With the use of dSPACE, this tool provides for developing, testing and calibrating ECU's in the automotive field. For this project she helped overlook the simulation part of the project, which tied to dSPACE. She looked through dSPACE source code, helped with the Simulink file and made sure that each part works. She also did research about LIN, which includes the function of the transmission carrier frame, and functions as part of the header. LIN allows for the communication between a single master, which is the 1401/1501MicroAutoBox, and up to sixteen slaves, which could be various different sensors or actuators, on a single bus system. Based on her research, she figured out the

identifiers were split in four categories: values 0 to 59 (0x3b) were used for signal-carrying

frames, 60 (0x3c) and 61 (0x3d) were used to carry diagnostic data, 62 (0x3e) was reserved for user-defined extensions, and 63 (0x3f) was reserved for future protocol enhancements. She also helped to overlook the actuators, which are LIN based that provides the air inlet, delivery, and mode in a vehicle-air-conditioner. The transmission is bi-directional and unsynchronized communication that is set up as a slave receiving and responding to a command. In this case, it is important to ensure the actuators interfacing to avoid damage. Also, The MicroAutoBox requires a power supply in the range of 6-40 V (30 V with a 5 AMP peak). The power supply should operate at 12 V and deliver 5 AMP peak current. To achieve this task, she assisted in wiring and inserting a fuse within the range of 8 to 10 AMP into the power supply to protect the MicroAutoBox.

Appendix 2: References

- [1] Anders Rylander & Erik Wallin. "LIN - Local Interconnect Network - for use as sub-bus in Volvo trucks". 2003. Available online: <http://www.mrtc.mdh.se/rtis/contrib/Exjobb/03.PDF>
- [2] Anti-Windup Control Using a PID Controller. February 19, 2014 Available online: <http://www.ni.com/white-paper/9733/en/>
- [3] Automotive HVAC Market by Vehicle Type (Passenger Cars, LCVs & HCVs), Technology (Manual & Automatic), Components & Cabin Comfort Market by Type (Power Seats, Power Windows, Heated Seats & Sunroofs) - Global Trends & Forecast to 2018. February 01, 2014. Available online: <http://www.marketresearch.com/MarketsandMarkets-v3719/Automotive-HVAC-Vehicle-Type-Passenger-7750028/>
- [4] STMicroelectronics. "J2602 Software Package". November 2012. Available online: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00068799.pdf
- [5] Gilles Delorme, Phillip Jacquet. "GM LIN Actuator" General Motors. 15 Mar. 2012
- [6] dSPACE Application fields in the automotive industry available online 2014 <https://www.dspace.com/en/ltd/home/applicationfields/automotive.cfm>
- [7] Gilles Delorme. "Solar/Ambient Light/Temperature Sensor With Indicators." General Motors. 15 Jan. 2013
- [8] K.J. Astrom and B. Wittenmark, *Computer-Controlled Systems, 3rd Ed*, Prentice Hall, 199
- [9] "Local Interconnect Network." *Wikipedia*. Wikimedia Foundation, 29 Jan. 2014. Web. 01 Feb. 2014. Available online: http://en.wikipedia.org/wiki/Local_Interconnect_Network
- [10] M. Honek, J. Csamb' al, S. Wojnar, M. Kopačka, P. Simončíř, and M. Lauko. "RAPID CONTROL PROTOTYPING SYSTEM DSPACE USED FOR CONTROL OF COMBUSTION ENGINE PROCESSES." *Slovak University of Technology*. N.p., n.d. PDF Available Online: http://dsp.vscht.cz/konference_matlab/MATLAB10/full_text/039_Honek.pdf
- [11] Natalie A. Wienckowski, Merv Rose Jr. "J2602/LIN 2.0 Seminar CTIS # 28302." *General Motors*. 19 May, 2010
- [12] National Instruments. "Introduction to the Local Interconnect Network (LIN) Bus". February 01, 2014 Available online: <http://www.ni.com/white-paper/9733/en/>
- [13] Klaus Lamberg, Peter Wältermann, "Using HIL Simulation to Test Mechatronic Components in Automotive Engineering". dSPACE GmbH, Paderborn. 16 November, 2000. Available Online: http://www.dspaceinc.com/ftp/papers/hdt00_e.pdf
- [14] Gilles Delorme. "Rain Light Tunnel Sensor with Humidity Sensor" General Motors. 30 Nov. 2009

Appendix 3: Running Charts, Gantt Chart and Beyond

Table 3 - Gantt Chart

Task Name	Duration	Start	Finish
Project Definition and Objectives	16 days	Fri 1/10/2014	Fri 1/31/2014
First Meeting with Team 4	1 days	Mon 1/14/14	Tue 1/14/14
First Contact with GM Sponsor	1 days	Wed 1/15/14	Thu 1/15/14
Research dSPACE/Lin Operating System	3 day	Fri 1/15/14	Fri 1/17/14
Meet with GM Sponsors via Teleconference	7 days	Mon 1/20/14	Tue 1/28/14
Acquire multiple licenses	7 days	Mon 1/20/14	Tue 1/28/14
Review meeting notes to discuss with facilitator	2 days	Tue 1/21/14	Wed 1/22/14
First Meeting with Facilitator	8 days	Wed 1/22/14	Fri 1/31/14
Start Pre-Proposal and Gantt Chart	7 days	Thu 1/23/14	Fri 1/31/14
Divide sections to equalize writing load	7 days	Thu 1/23/14	Fri 1/31/14
Research on past projects that are similar	5 days	Thu 1/23/14	Wed 1/29/14
Find out if any like systems are available	5 days	Thu 1/23/14	Wed 1/29/14
Preliminary research on LIN/dSPACE found	5 days	Thu 1/23/14	Wed 1/29/14
Overview of the problem	5 days	Thu 1/23/14	Wed 1/29/14
Executive Summary	5 days	Thu 1/23/14	Wed 1/29/14
Risk Analysis/Objective and Design Specs	5 days	Thu 1/23/14	Wed 1/29/14
Conceptual Design Development	5 days	Thu 1/23/14	Wed 1/29/14
Start web page design with GM logo	5 days	Thu 1/23/14	Wed 1/29/14
Find out how to display for Design Day	5 days	Thu 1/23/14	Wed 1/29/14
Second meeting with GM and deliverables	5 days	Tue 1/28/14	Sat 2/1/14
Start creating a logic flow/diagram for LIN driver	4 days	Tue 1/28/14	Fri 1/31/14
Determine systems input/output parameters	4 days	Tue 1/28/14	Fri 1/31/14
Second meeting with Facilitator	1 day	Wed 1/29/14	Wed 1/29/14
Pre-proposal Due	0 days	Mon 2/3/14	Mon 2/3/14

Acquire Material to Begin Design	5 days	Mon 2/3/14	Fri 2/7/14
Pre-trial software/understanding/learnings	5 days	Mon 2/3/14	Fri 2/7/14
Pre-trial actuators/understanding/learnings	5 days	Mon 2/3/14	Fri 2/7/14
Begin Programming/Virtual Design	9 days	Mon 2/10/14	Thu 2/20/14
Program Simulink to dSPACE	2 days	Mon 2/10/14	Wed 2/12/14
Utilize LIN to actuators and sensor	9 days	Mon 2/10/14	Thu 2/20/14
Connect entire system	9 days	Mon 2/10/14	Thu 2/20/14
Begin testing parameters of virtual design	9 days	Mon 2/10/14	Thu 2/20/14
Begin Building Prototype for Display	20 days	Mon 3/1/4	Thu 3/20/14
Brainstorm Ideas	20 days	Mon 3/1/4	Thu 3/20/14
Start prototyping	20 days	Mon 3/1/4	Thu 3/20/14
Testing prototypes	20 days	Mon 3/1/4	Thu 3/20/14
Determine test edge cases	20 days	Mon 3/1/4	Thu 3/20/14
Collaborate on errors for final design	28 days	Mon 3/1/4	Thu 3/28/14
Final display prototype design	15 days	Tue 4/1/14	Tue 4/15/14
Repair/Finalize Design Specs	15 days	Tue 4/1/14	Tue 4/15/14
Due Dates	86 days	Fri 1/10/14	Tue 4/15/14
Proposal Presentation	1 day	Fri 2/14/14	Fri 2/14/14
Final Proposal Due	1 day	Mon 2/21/14	Mon 2/21/14
Site Visit to GM for Testing	1 day	Mon 2/24/14	Mon 2/24/14
Team Technical Lectures	1 day	Mon 3/24/14	Mon 3/24/14
Design Issues Paper	1 day	Fri 4/11/14	Fri 4/11/14
Professional Self-Assessment Paper	4 days	Wed 4/16/14	Wed 4/16/14
Final Report Due	1 day	Wed 4/23/14	Wed 4/23/14
Design Day	1 day	Fri 4/25/14	Fri 4/25/14

MicroAutoBox Configuration:

Pins	Signal	Description/Function
V3, V4, W3, W4, W5, X3, X4	VBAT	Main power supply input. Connect at least 2 pins to the plus of your car battery.
F3, F4, G3, G4, G5, H3, H4	GND	Main power supply return and reference for all input and output signals. Connect at least 2 pins to the minus of your car battery. Use unused pins to reference your signals. This signal is also connected to the case of MicroAutoBox.
E1	VDRIVE	This input supplies all digital input and output circuits. - Connect this input to VSNS to get TTL-compatible logic levels to your inputs/outputs. - Connect this input to VBATprot to get automotive-compatible logic levels to your inputs/outputs. - Do not connect this pin directly to VBAT, because the input and output circuits are not load dump or reverse voltage protected at this signal. - While MicroAutoBox is being powered down, the output stages have pull-up behavior to VDRIVE. So the outputs may reach the level of VDRIVE. If this behavior is critical in your application, you have to power down your external devices or VDRIVE before MicroAutoBox is powered down. You can also use the REMOTE signal (K1.15) with the power control software functionality to first switch off VDRIVE via a relay.
E2	VSNS	Sensor Supply output. Use this output to supply your sensors and/or VDRIVE.
c2	VBAT prot	Protected VBAT output. Use this output to supply VDRIVE when automotive logic levels are needed.

1	2	3	4	5	6	
DAC 7 out	DAC 8 out	Group 1 ch 1 out	Group 1 ch 2 out	Group 1 ch 3 out	Group 1 ch 4 out	A
DAC 5 out	DAC 6 out	Group 1 ch 5 out	Group 1 ch 6 out	Group 1 ch 7 out	CTM ch 1 out	B
DAC 4 out	DAC 2 out	CTM ch 2 out	CTM ch 5 out	CTM ch 6 out	CTM ch 7 out	C
DAC 3 out	DAC 1 out	CTM ch 8 out	CTM ch 3 out	CTM ch 4 out	Group 6 ch 1 out	D
VSNS in	VSNS out	Group 6 ch 7 out	GND in	GND in	Group 6 ch 4 out	E
Group 6 ch 6 out	Group 6 ch 7 out	GND in	GND in	GND in	Group 6 ch 8 out	F
TPU ch 2 out	TPU ch 3 out	GND in	GND in	GND in	TPU ch 4 out	G
TPU ch 5 out	TPU ch 6 out	GND in	GND in	TPU ch 7 out	TPU ch 8 out	H
TPU ch 9 out	TPU ch 10 out	TPU ch 11 out	TPU ch 12 out	TPU ch 13 out	TPU ch 14 out	J
TPU ch 15 out	TPU ch 16 out	Group 2 ch 1 out	Group 2 ch 2 out	Group 2 ch 3 out	Group 2 ch 4 out	K
Group 2 ch 5 out	Group 2 ch 6 out	Group 2 ch 7 out	Group 2 ch 8 out	Group 3 ch 1 out	Group 3 ch 2 out	L
Group 3 ch 3 out	CTM ch 1 in	REMOTE in	CTM ch 2 in	CTM ch 3 in	CTM ch 4 in	M
Group 6 ch 1 in	Group 6 ch 2 in	Group 6 ch 3 in	Group 6 ch 4 in	Group 6 ch 5 in	Group 6 ch 6 in	N
Group 6 ch 7 in	Group 6 ch 8 in	TPU ch 1 in	TPU ch 2 in	TPU ch 3 in	TPU ch 4 in	P
TPU ch 5 in	TPU ch 6 in	TPU ch 7 in	TPU ch 8 in	TPU ch 9 in	TPU ch 10 in	R
TPU ch 11 in	TPU ch 12 in	TPU ch 13 in	TPU ch 14 in	TPU ch 15 in	TPU ch 16 in	S
Group 2 ch 1 in	Group 2 ch 2 in	Group 2 ch 3 in	Group 2 ch 4 in	Group 2 ch 5 in	Group 2 ch 6 in	T
Group 2 ch 7 in	ADC Type 1 Con 4 Ch 4	Serial 2 K / LIN i/o	Serial 2 L in	Serial 1 TXD out	Serial 1 RXD in	U
ADC Type 1 Con 2 Ch 4	ADC Type 1 Con 3 Ch 4	VBAT in	VBAT in	CAN 1 low i/o	CAN 1 high i/o	V
ADC Type 1 Con 1 Ch 4	ADC Type 1 Con 4 Ch 3	VBAT in	VBAT in	VBAT in	Group 2 ch 8 in	W
ADC Type 1 Con 2 Ch 3	ADC Type 1 Con 3 Ch 3	VBAT in	VBAT in	CAN 2 low i/o	CAN 2 high i/o	X
ADC Type 1 Con 1 Ch 3	ADC Type 1 Con 4 Ch 2	Group 4 ch 1 in	Group 4 ch 2 in	ECU / IF RX+ in	Group 4 ch 3 in	Y
ADC Type 1 Con 2 Ch 2	ADC Type 1 Con 3 Ch 2	Group 4 ch 4 in	Group 4 ch 5 in	ECU / IF RX- in	Group 4 ch 6 in	Z
ADC Type 1 Con 1 Ch 2	ADC Type 1 Con 4 Ch 1	Group 4 ch 7 in	Group 4 ch 8 in	ECU / IF TX- out	Group 5 ch 1 in	a
ADC Type 1 Con 2 Ch 1	ADC Type 1 Con 3 Ch 1	Group 5 ch 2 in	Group 5 ch 3 in	ECU / IF TX+ out	Group 5 ch 4 in	b
ADC Type 1 Con 1 Ch 1	VBAT prot out	Group 5 ch 5 in	Group 5 ch 6 in	Group 5 ch 7 in	Group 5 ch 8 in	c
1	2	3	4	5	6	

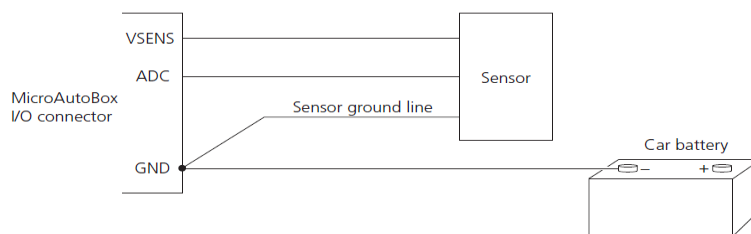
ZIF Connector Pin Description

Date	Revision	Features	Boot Firmware Version	dSPACE Release
DS1501				
Q2/2002	07	Multi-channel PWM added	not relevant	3.4
Q3/2003	09	LIN support added	not relevant	Using the LIN support requires at least Release 4.0.

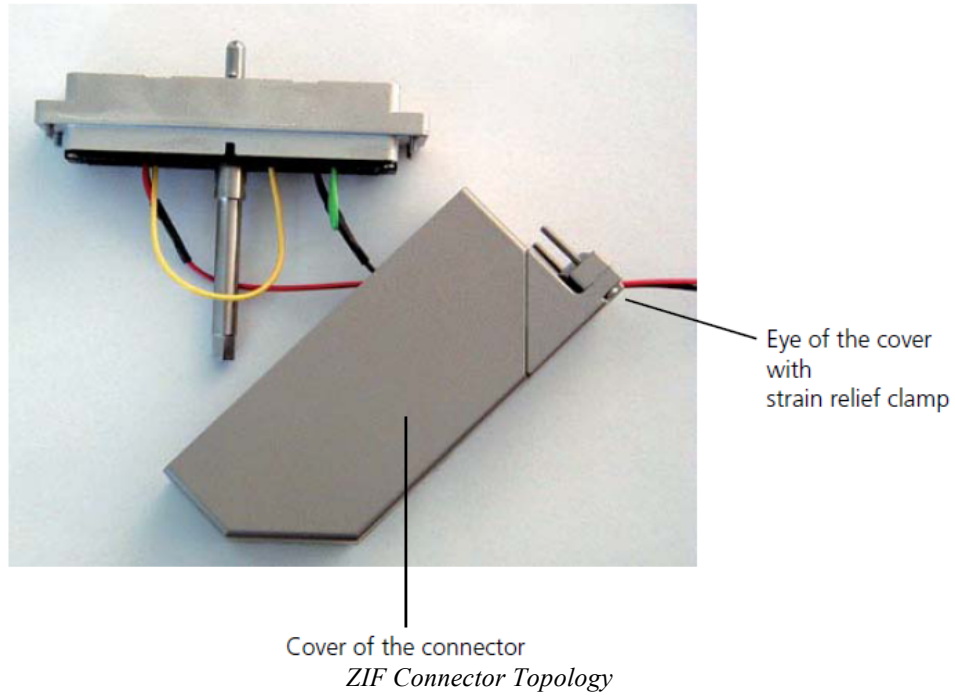
Requirements for 1501 Board

Ideal grounding

Ideal grounding as shown in the illustration below is strongly recommended if there are high load currents in your application and if you want to measure highly accurate values.



Ideal Grounding



Running Charts:

