ECE 59500: Embedded Autonomous Systems

Project #1 Final Report - Group 03

Adaptive Cruise Control Using LIDAR

Priyank Kalgaonkar – Team Member
Kavyashree Prasad S P – Team Member
Arjun Narukkanchira Anilkumar – Team Member

Department of Electrical and Computer Engineering,
Purdue School of Engineering and Technology at IUPUI.

Submitted to:

Course Instructor
Dr. Mohamed El-Sharkawy
Department of Electrical and Computer Engineering, IUPUI.

Submitted as partial fulfillment for the requirement of Spring 2020 - ECE 59500: Embedded Autonomous Systems course.

Date of Submission
March 11, 2020

Executive Summary

The design of Adaptive Cruise Control system uses two microcontroller units, a Garmin LIDAR module which is compact and high-performance optical distance measurement sensor, and Brushless DC (BLDC) electric motor. The main objective of Adaptive Cruise Control Using LIDAR project is to enable the cruise control system used in road vehicles to automatically slow down or speed up the vehicle up to a speed limit set by the driver. The result is improved driver's comfort, and safety of the vehicle and other traffic on the road by minimizing rear-end collisions.

According to a report from the National Highway Transportation Safety Administration, rear-end crashes are the most frequently occurring type of collision. About 29 percent of all car crashes are rear-end collisions. These crashes result in a substantial number of injuries and fatalities each year. In fact, roughly 1.7 million rear-end collisions take place in the United States each year. Of these nearly 2 million accidents, about 1,700 people die and another 500,000 are injured in these types of crashes [1]. These numbers constitute a significant portion of highway accidents, injuries, and fatalities.

Our proposed Automatic Cruise Control (ACC) system utilizes a LIDAR sensor that communicates with the on-board ECU (MCU for our project) using I^2C protocol to enable this feature. The ACC system will control the speed of the vehicle the same way a driver does – by adjusting the throttle (accelerator) position. I^2C master (MCU #1) will receive data from I^2C Slave (LIDAR sensor) which will then be transferred to MCU #2 via UART. MCU #2 will then utilize this information to control the BLDC motor which will simulate the ECU controlling the speed of the road vehicle.

This report provides detailed information that will be useful for companies interested in this concept and the engineering team(s) responsible for further research and development.

Acknowledgement

This project consumed an immense amount of work, research, and dedication. Nonetheless, implementation would not have been possible without the support of many individuals and organizations.

We are highly indebted to our professor: Dr. Mohamed El-Sharkawy, and our teaching assistants: William Singleton and Niranjan Ravi, at the Purdue School of Engineering and Technology at IUPUI for providing us components for completing this project. During the course of this work, their constant association with my team has been most pleasurable. Therefore, on behalf of all my team members, I would like to extend our sincere gratitude to all of them.

We are also grateful to Mr. Jeffrey Sears who provided us with access to the SL111 lab in the Engineering Science & Technology (SL) Building and with necessary tools to build, develop, solder and test hardware components for this project. Without his time and support in providing necessary tools and components, our hardware team would have been greatly challenged to develop, integrate and test the hardware subsystems and components for this project.

My thanks and appreciations also go to my colleagues involved in developing this project and especially to those who have willingly helped me out with abilities at their best. I would like to extend my special gratitude to my team members: Arjun Narukkanchira Anilkumar and Kavyashree Prasad S P, the members of Group 03 actively involved in this project.

We would like to express our sincere thanks towards volunteer helpers, family, and friends who provided us with essential support and additional resources in the implementation of this project.

Without the help and counsel of the aforementioned individuals and organizations, always generously and unstintingly given, the completion of this work would have been immensely difficult and time consuming.

On behalf of Group 3,
Priyank Kalgaonkar.

Table of Contents

List of Tables

List of Figures

1.0 INTRODUCTION

Cruise control is a servo mechanism that communicates with the on-board ECU to maintain a steady speed set by the driver. Traditional cruise control system commonly used in road vehicles today operates by maintaining a constant speed regardless of traffic up ahead. This system cannot slow down or speed up the vehicle. The goal of our project is to enable this cruise control system to automatically slow down or speed up the vehicle up to a speed limit set by the driver.

The idea of the Adaptive Cruise Control Using LIDAR project is to make the traditional cruise control system more safer and smarter by utilizing LIDAR sensors to maintain a safe distance from vehicles up ahead and significantly reduce chances of rear-end collisions. LIDAR, which stands for Light Detection and Ranging, is a surveying sensor module that measures distance to a target by illuminating the target with laser light and measuring the reflected light with a sensor. This sensor fires rapid pulses of laser light at a surface, at up to 150,000 pulses per second, which makes LIDAR a perfect choice since it enables faster distance measurements as well as also offers precise measurement data over short to long ranges, even in challenging weather and lighting conditions.

For building a prototype of this system, we chose to implement a Garmin LIDAR-Lite v3 sensor module, two NXP S32K144-EVB development boards and a BLDC motor as below.

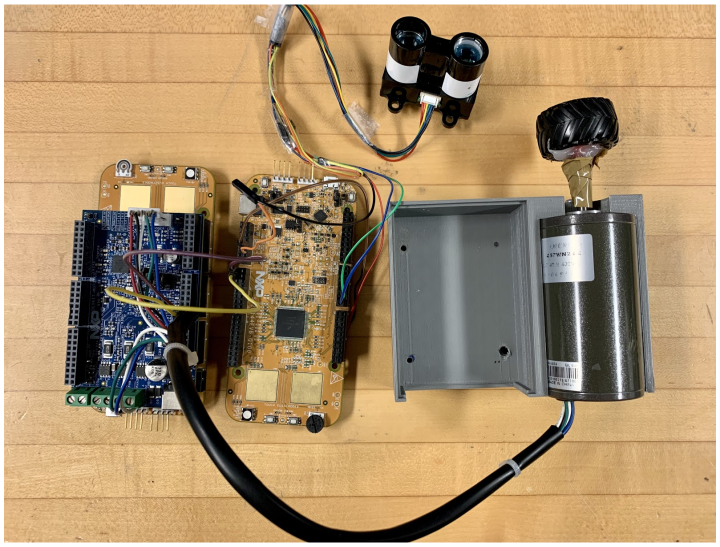


Figure 1.1: Prototype Photo

2.0 PROBLEM STATEMENT

2.1 Need

As stated in the introduction, the traditional cruise control system commonly used in road vehicles today operates by maintaining a constant speed regardless of traffic up ahead. This system cannot slow down or speed up the vehicle. Thus contributing to a substantial number of injuries and fatalities each year caused by rear-end collisions.

2.2 Objective

To enable the traditional cruise control system to automatically slow down or speed up the vehicle up to a speed limit set by the driver and maintain a safe distance with traffic up ahead. This will result in improved safety of vehicle and people in and around the road vehicle as well as improve driver’s comfort by maintaining a constant speed safely without keeping his/her foot on the gas or brake pedal.

2.3 Marketing Requirements

The marketing requirements are focused mainly on demonstrating how the Adaptive Cruise Control system will work using a single LIDAR sensor module. They are as follows:

1. Processing should be done in real-time and quickly (<1s).
2. Data from LIDAR module should automatically control the motor's speed.
3. Improve safety and ride comfort whilst maintaining good fuel economy.

The detailed engineering requirements derived from these marketing requirements are listed in section 3.3. Further discussion on system design will follow in the coming sections.

3.0 SYSTEM DESIGN

3.1 System Architecture

The overall architecture of the Adaptive Cruise Control (ACC) system has at its center: two NXP S32K144-EVB development boards featuring a 32-bit Arm® Cortex®-M4F S32K14 MCU running up to 112 MHz (HSRUN) and 80 MHz (Normal RUN), up to 2 MB code flash memory and 64 KB FlexMem, and up to 256 KB SRAM. The two MCUs are each assigned to a unique task. MCU #1 (I^2C Master) will receive data from Garmin LIDAR sensor (I^2C Slave) which will then be transferred to MCU #2 via UART. MCU #2 will then utilize this information to control the BLDC motor via the NXP DEVKIT-MOTORGD integrated motor control shield, which will simulate the ECU controlling the speed of the road vehicle. The functional architecture is drawn in the figure below:

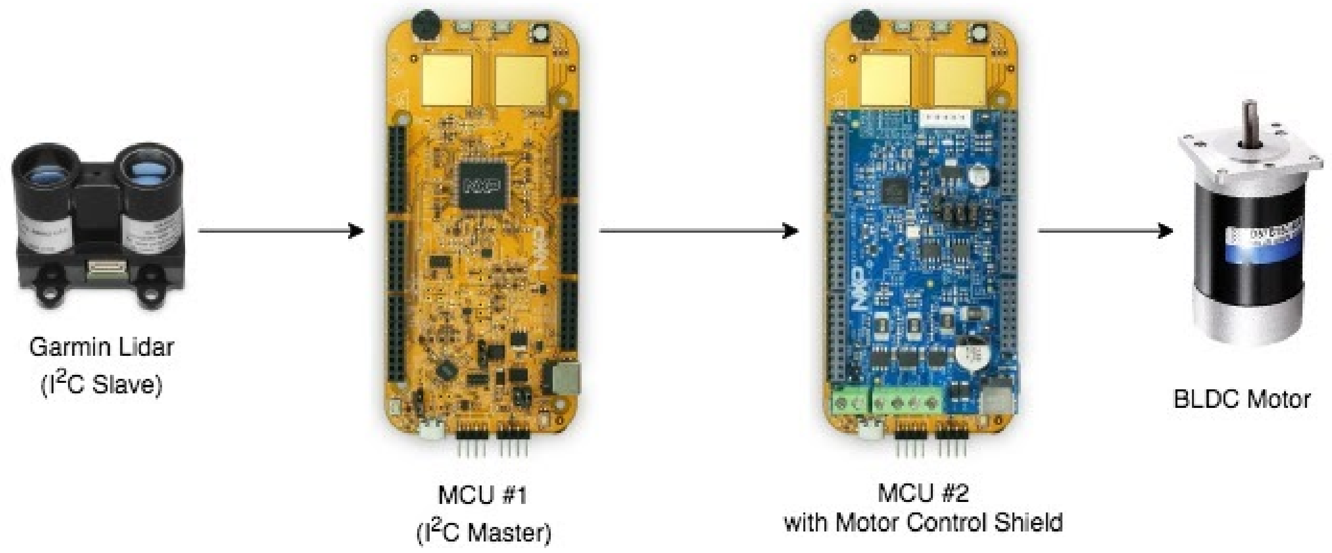


Figure 3.1.1: Adaptive Cruise Control System Architecture

3.2 Data Flow

The system consists of three stages: data acquisition via I^2C serial protocol, data transfer via UART and motor control. Data acquisition is done by LIDAR sensor module (I^2C Slave) which shines a small beam of light at a surface and measures the time it takes for this beam of light to bounce and return back to its source. This data is received by MCU #1 (I^2C Master) which then transfers it to MCU #2 via UART which is a physical circuit in a microcontroller that transmits and receives serial data, not communication protocol like I^2C. MCU #2 then transfers this data to NXP DEVKIT-MOTORGD integrated motor control shield which controls the speed of the BLDC motor accordingly.

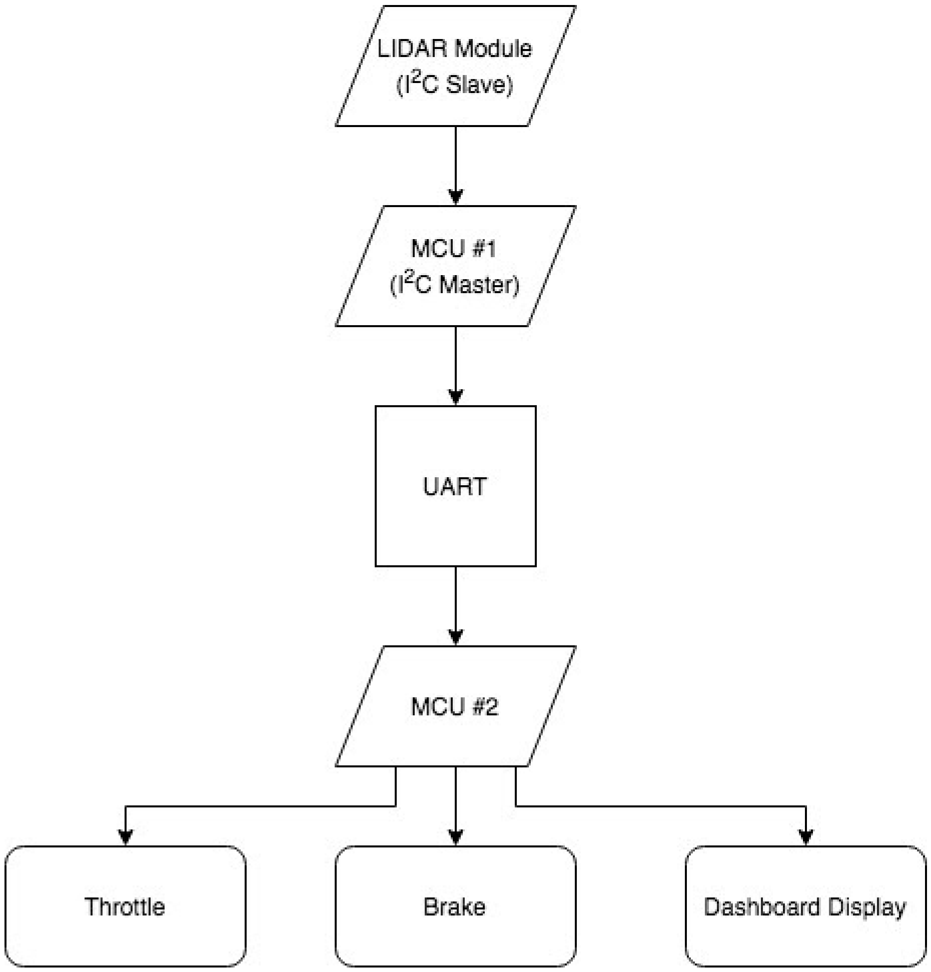


Figure 3.2.1: Data Flow Diagram for Adaptive Cruise Control System

3.3 Engineering Requirements

To satisfy the marketing requirements explained in section 2.3, we have set the engineering requirements listed below.

Marketing Require-ment	Engineering Requirement	Justification
1	Processing should be done in real-time and quickly (<1s).	Data acquired from LIDAR module should be processed in real-time so that a decision can be made to speed-up/maintain/slow the road vehicle, especially when the vehicle has to utilize full braking power in case of an emergency.
2	Data from LIDAR module should automatically control the motor's speed.	This will allow the MCUs to automatically control the speed of the road vehicle and allow it to either speed up, maintain or slow the vehicle based on traffic up ahead.
3	Improve safety and ride comfort whilst maintaining good fuel economy.	Only slow and brake when necessary and maintain a constant speed, avoiding fluctuations in fuel consumption and limiting input from the driver. Thus, increasing ride comfort.

Table 3.3.1: Engineering Requirements Justification Table

3.4 Validation

The following analyses explain how our design is expected to meet the engineering requirements. Each explanation is listed under the corresponding engineering requirement, reprinted below for convenience:

1. **Processing should be done in real-time and quickly (<1s).**

   Since NXP’s S32K144EVB is powered by a 32-bit ARM Cortex M4F core running up to 112MHz, the data should be processed with an acceptable delay.

2. **Data from LIDAR module should automatically control the motor's speed.**

   We propose to use NXP’s DEVKIT-MOTORGD integrated motor control shield in conjunction with NXP’s S32K144EVB MCU which will receive LIDAR data from NXP’s S32K144EVB MCU. This will enable the MCU to control BLDC motor’s speed per the predefined parameters.

3. **Improve safety and ride comfort whilst maintaining good fuel economy.**

   Since the size of components chosen for this system are relatively fast in processing data, this system can bring the road vehicle to a complete stop in case of an emergency. In regular scenarios, driver input is minimized. Thus improving safety, ride comfort whilst maintaining/improving fuel economy.

4.0 STANDARDS COMPLIANCE

4.1 Hardware Standards

All the components used in this system are bought off-shelf from the electronics store, which are manufactured in compliance with the following standards and hence, the overall system, when fully assembled, will be in compliance with the standards described below in detail:

A. Safety and Health Compliance Standards:

EN 60950-1:2006 + A1:2010 + A11:2009 + A12:2011: The safety standard EN 60950-1 applies to mains powered equipment such as the NXP S32K144EVB, NXP DEVKIT-MOTORGD integrated motor control shield, Garmin LIDAR-Lite v3 sensor module and Brushless DC electric motor. "Requirements within this standard address normal equipment operating conditions, likely and consequential faults, foreseeable misuse and external influences such as temperature, altitude, pollution, moisture and over voltages. In general, these requirements are intended to minimize the risk of fire, electric shock and injury for the operator, layman, service person and anyone else that may come into contact with the equipment during installation, operation and maintenance." [3][4][6]

B. Electromagnetic Compatibility (EMC) Compliance Standards:

EN 55022:2010: The safety standard EN 55022 applies to the Radio disturbance characteristics of the components in the system. All the components of the system have been tested by the manufacturer and found to comply with the limits for Class B Information Technology Equipment according to the European Standard except the touch screen display which complied with the limits for a Class A Information Technology Equipment according to the European Standard. [3][4][6]

C. Federal Communications Commission (FCC) Compliance Standards:

FCC Emissions Compliance Statement: All the components of the system have been tested by their respective manufacturers and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. All the components of the system comply with Part 15 of the FCC Rules. Operation is subject to the following two conditions:

a) This system might not cause harmful interference

b) This system must accept any interference received, including interference that may cause undesired operation. [3][4][6]

D. Restriction of Hazardous Substances Directive (RoHS) Compliance Standards:

RoHS Directive Compliance Statement: NXP S32K144EVB MCU board and other components of this system complies with the relevant provisions of the RoHS Directive for the European Union. In common with all Electrical and Electronic Equipment (EEE), the NXP S32K144EVB MCU or any components of the Adaptive Cruise Control system should not be disposed of as household waste. Alternative arrangements may apply in other jurisdictions. [3][4][6]

4.2 Software Standards

Since our project includes custom software developed in the C programming language in S32DS for Arm® Software Development Kit (SDK) which includes the S32 Design Studio Integrated Development Environment (IDE) and debugger, Arm C/C++ compiler, and essential middleware components, it is expected to adhere to Arm Holding's standards. The SDK is licensed under the permissive Apache 2.0 (Open-Source) license and so it can be used in both commercial and personal projects. For BLDC motor control system design, MATLAB-Simulink was used as a graphical block diagramming tool. A MATLAB educational license was obtained from our educational institution: Indiana University - Purdue University at Indianapolis for this project. The software in C programming language and graphical block diagram in MATLAB-Simulink mentioned in this report was developed by us for this project.

5.0 DETAILED DESIGN

5.1 Physical Design

Our prototype utilizes two NXP S32K144EVB microcontroller units, a Garmin LIDAR Lite v3 module which is compact and high-performance optical distance measurement sensor, and Brushless DC (BLDC) electric motor. We have setup a prototype of our system by utilizing a simple breadboard and connecting wires as shown below:

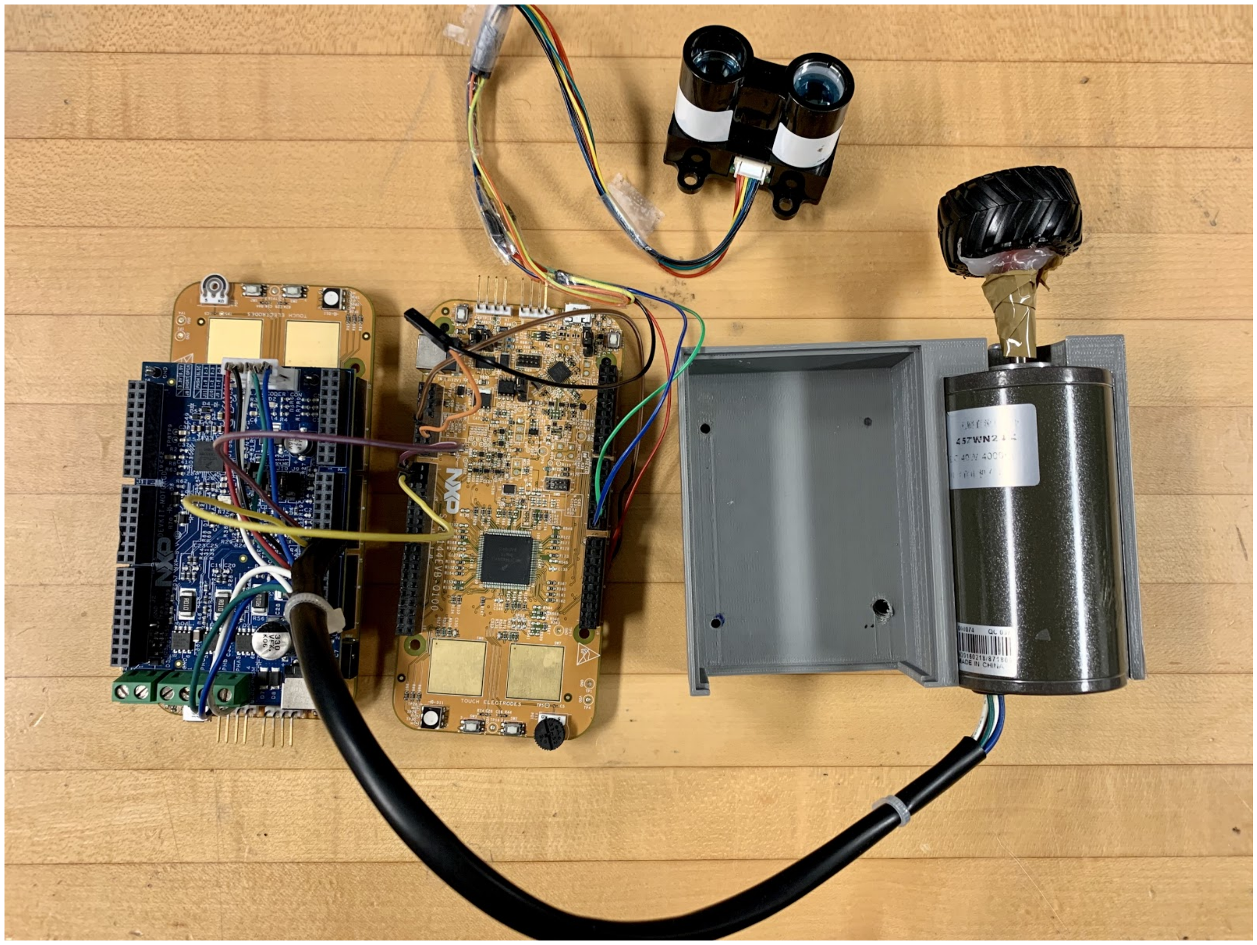


Figure 5.1.1: Adaptive Cruise Control Prototype.

5.2 Component Dimension

Measurements of the main components (excluding wires) are shown in Table 5.1.2 below:

Component	Height (mm)	Width (mm)	Depth (mm)
NXP S32K144EVB MCU	67.6	30	2.7
Motor Control Shield	51	30	2.5
Garmin LIDAR Lite v3	40	48	20
BLDC Motor	75	20	20

Table 5.1.2 Table for Component Dimensions.

5.3 Software Design

The software was developed in S32DS for Arm® SDK utilizing C Programming Language, which is one of the recommended programming languages. The software design consists of:

1. Starting up the code.
2. Checking C Library integration.
3. Checking Peripheral libraries.
4. Executing the code.

MVC stands for Model, View and Controller — the three main elements of an MVC design as follows:

- Model: The model is a representation of the data. In this case, the model is receiving data from the LIDAR sensor module (I^2C Slave).
- View: The view is represented by data output observed on TeraTerm's screen.
- Controller: The controller coordinates the model and the view. In this case, the Controller is S32K144EVB MCU #2 and Motor Control Shield.

Our software design contains the following steps:

1. LIDAR sensor module (I^2C Slave) to MCU #1 (I^2C Master): Data acquisition is done by LIDAR sensor module (I^2C Slave) which shines a small beam of light at a surface and measures the time it takes for this beam of light to bounce and return back to its source. The light photon travels at the speed of light and the sensor module calculates the distance using the following formula:

$$Distance = \frac{(Speed\ of\ Light * Time\ of\ Flight)}{2}$$

I^2C and UART are initialized and when I^2C bus and LIDAR sensor are idle, a Write 0x04 is sent to address 0x00 to initialize ranging. I^2C then reads 2 bytes from address 0x8F which belongs to sensor data register. MCU #1 (I^2C Master) then processes the received data and sends it to MCU #2 through UART.

Figure 5.3.1 below, provides a diagrammatic representation of this algorithm.

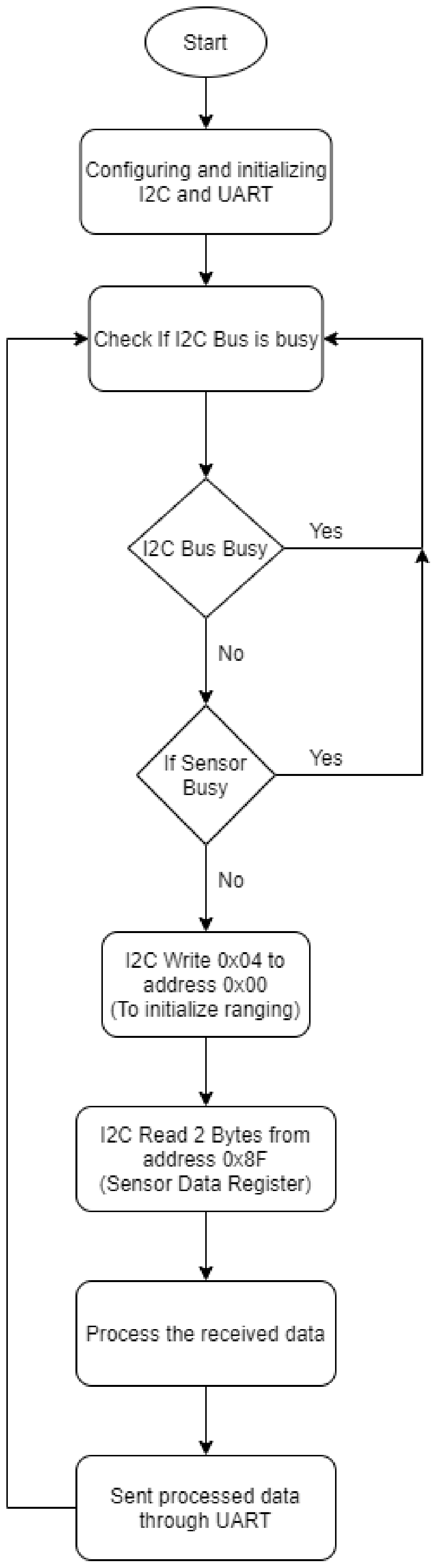

Start
Configuring and initializing I2C and UART
Check If I2C Bus is busy
I2C Bus Busy
Yes
No
If Sensor Busy
Yes
No
I2C Write 0x04 to address 0x00 (To initialize ranging)
I2C Read 2 Bytes from address 0x8F (Sensor Data Register)
Process the received data
Sent processed data through UART

Figure 5.3.1: Software Design Flow Chart 1.

2. MCU #1 (I^2C Master) to MCU #2 (Motor Control Unit): MCU #2 board receives data from MCU #1 board via UART. UART is a Universal Asynchronous Receiver-Transmitter is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable[7]. MCU #2 utilizes NXP's DEVKIT-MOTORGD integrated motor control shield to process the received data from MCU #1 (I^2C Master) to set the desired motor speed according to the pre-defined parameters. Figure 5.3.2 below, provides a diagrammatic representation of this algorithm.

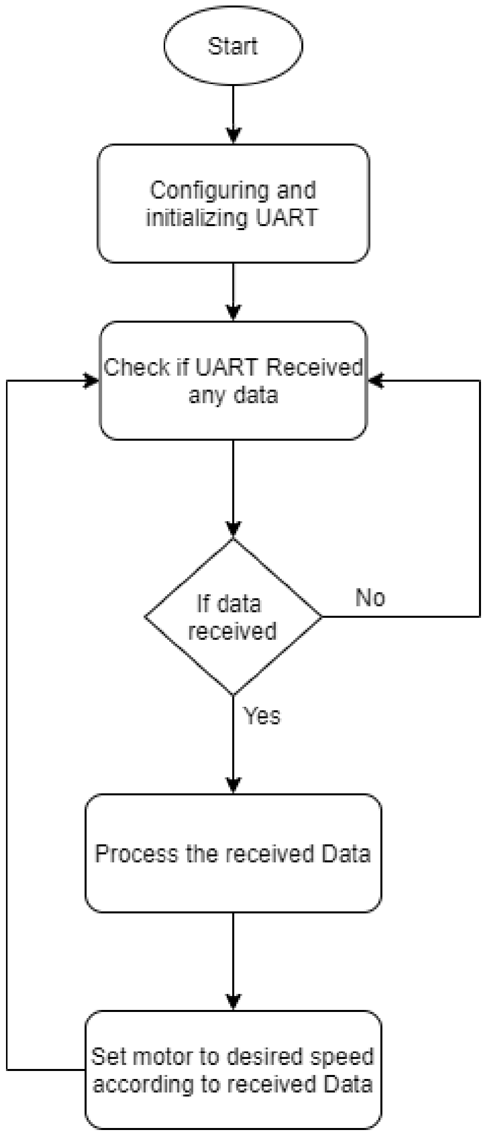

Start
Configuring and initializing UART
Check if UART Received any data
If data received
No
Yes
Process the received Data
Set motor to desired speed according to received Data

Figure 5.3.2: Software Design Flow Chart 2.

5.4 Hardware Design

Schematic for our prototype is shown in fig.5.4.1 below. It has been designed using Cadence OrCAD.

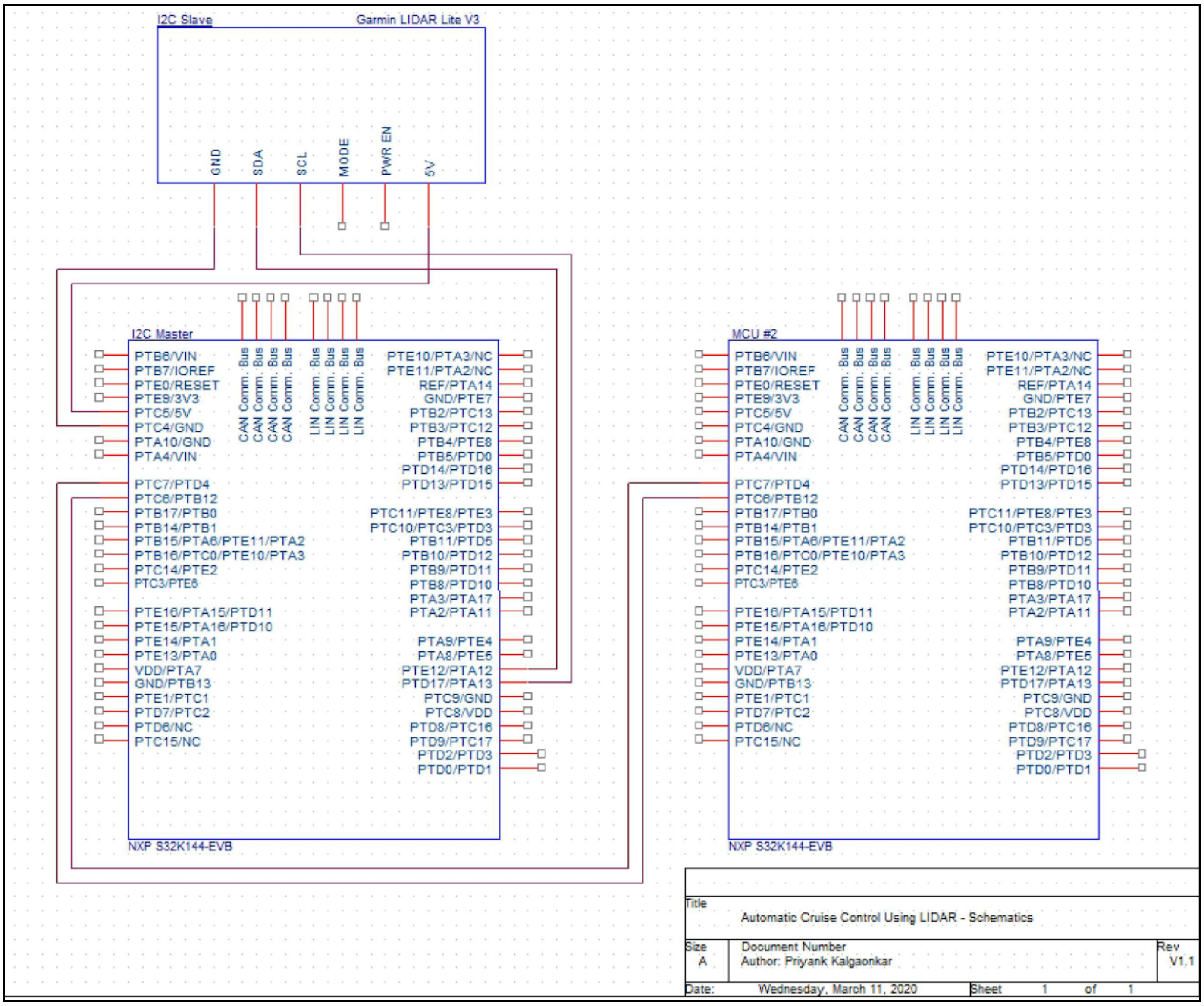


Figure 5.4.1: System Schematics

As seen the figure 5.4.1 above, following are detailed specifications of the components that we have used in our project:

1. **Garmin LIDAR-Lite v3:** This module is an ideal compact and high-performance optical distance measurement sensor. One LIDAR sensor module has been used in this project for cruise control automation. Data acquisition is done by this module which shines a small beam of light at a surface and measures the time it takes for this beam of light to bounce and return back to its source. Features [6] of Garmin LIDAR-Lite v3 sensor module are as follows:
    - Communication via I^2C and PWM.
    - Low power consumption (4.75-5 VDC; 6 V Max).
    - Accuracy: +/- 2.5 cm at distances greater than 1 meter.
    - Range: 5 cm to 40 meters.
    - Refresh rate: up to 500 Hz.
    - Operating temperature: -20°C to 60°C.
    - Beam divergence: 8 meter Radian.

2. **NXP S32K144-EVB Development Board:** It is a low cost, power efficient development board with 32-bit Arm® Cortex®-M4F S32K14 MCU. Following are the specifications of this development board[3]:
    - 512KB Flash memory, 64KB RAM with 112MHz clock.
    - Arduino™ UNO footprint-compatible with expansion shield support.
    - Flexible power supply options: microUSB or external 12V power supply.
    - Easy access to the MCU I/O header pins for prototyping.
    - On-chip connectivity for CAN, LIN, UART/SCI.
    - Potentiometer for precise voltage and analog measurement.
    - Two push-button switches (SW2 and SW3) and two touch electrodes.
    - Integrated open-standard serial and debug adapter (OpenSDA) with support for several industry-standard debug interfaces.

3. **NXP DEVKIT-MOTORGD integrated motor control shield:** This is an integrated motor control shield from NXP for the DEVKIT platform. It combines the functionality of the FRDM-GD3000EVB and FRDM-PWRSTG for a full automotive solution. DEVKIT-MOTORGD is compatible with the low-cost evaluation board of the S32K and MPC5744P, S32K144EVB-Q100 and DEVKIT-MPC5744P, respectively, and can drive a 3-phase BLDC/PMSM motor. The built-in Hall/Encoder interface means DEVKIT-MOTORGD can support both sensored and sensorless motor control applications [3].
4. **Brushless DC electric motor:** It is an electronically commutated motor synchronous DC motor powered by direct current (DC) electricity via an inverter or switching power supply which produces electricity in the form of alternating current (AC) to drive each phase of the motor via a closed loop controller. The controller provides pulses of current to the motor windings that controls the speed and torque of the motor[4].

6.0 INTEGRATION AND TESTING

6.1 Integration Plan

The items tested consist of the integration of the software code modules developed as well as the hardware components. For testing software code modules, we choose the middle-out approach whereas for testing the hardware components of the subsystem, we choose the top-down approach. The integration sequence is described below, while the individual tests are in sections 6.2 and 6.3.

6.1.1 Software Integration Sequence

Although we did not do any explicit unit tests on our software, we did incremental tests of the components as we built the software to ensure all of the modules were working together as they should. This helped us remove defects early and focus on future integrations.

6.1.2 Hardware Integration Sequence

The integration approach chosen for this system was a top-down approach. This means that the designer has an overall vision of what the final system must do, and the problem is partitioned into components, or subsystems that work together to achieve the overall goal of the system for this project. Then each subsystem is successively refined and partitioned as necessary.

In the case of our Adaptive Cruise Control Using LIDAR project, the overall objective was determined as aforementioned in the previous sections; the major components are defined such as the NXP S32K144-EVB development boards, Garmin LIDAR-Lite v3 sensor module, NXP DEVKIT-MOTORGD integrated motor control shield and BLDC electric motor.

Before we integrated the various components of the subsystem, we performed individual testing of each hardware component. This way the project was built down starting from the top level because this method was more suitable, where it is unlikely that bringing together pieces in an ad-hoc fashion would have successfully solved the problem. This integration test, as described, is at the component level in section 6.2.

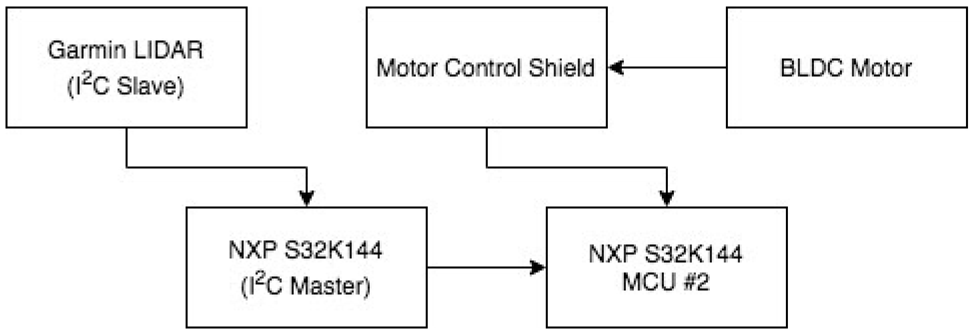


Figure 6.1.2: Hardware Integration Order.

6.1.3 Full Hardware-Software Integration Sequence

The integration approach chosen for this system was a top-down approach. This means that the designer has an overall vision of what the final system must do, and the problem is partitioned into components, or subsystems that work together to achieve the overall goal of the system for this project. Then each subsystem is successively refined and partitioned as necessary.

Before we integrated various software components as mentioned in section 6.1.1, we assembled all hardware components as per section 6.1.2 because integration of software build(s) depends on the hardware modules. In particular, all the hardware modules had to function properly before we could integrate software modules with hardware modules.

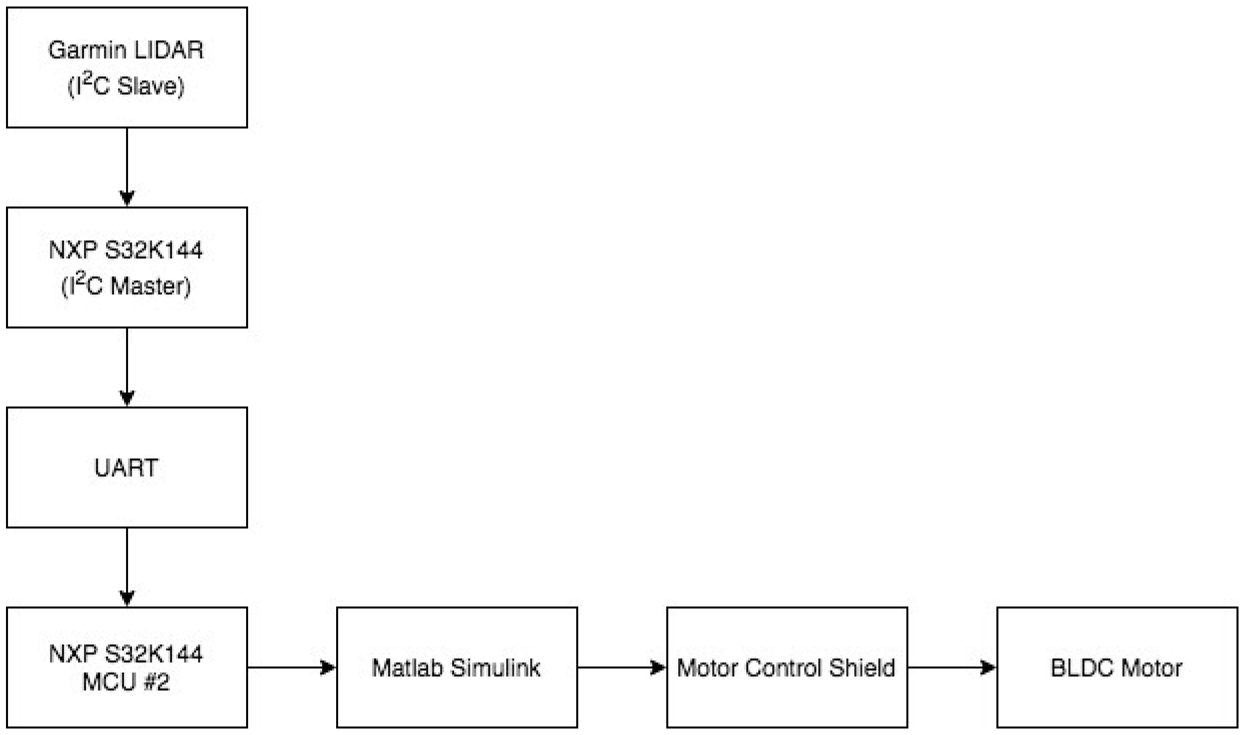


Figure 6.1.4: Full Hardware-Software Integration Order

6.2 Unit Testing - Hardware

Test Writer: Priyank Kalgaonkar.						
Test Case Name:		Test Garmin LIDAR-Lite v3 sensor module.		**Test ID #:**		SL-UT-01
Description:		This test shall be performed on the LIDAR-Lite v3 sensor module. S32K microcontroller and programming required.				
Tester Information						
Name of Tester:		Arjun and Kavya		**Date:**		03/02/2020
System Ver:		N/A		**Time:**		12:30 PM
Setup:		Connect the 5V pin and GND of the LIDAR module to the 5V and GND pin on S32K respectively. Connect SDA to PTA17 and SCL to PTA11. Connect S32K to a PC. Build and push the LIDAR test program to the board and verify output in TeraTerm.				
Step	**Action**	**Expected Result**	**Pass**	**Fail**	**N/A**	**Comments**
1	Verify output in TeraTerm.	Distance measurements should be seen in TeraTerm.	✓			LIDAR sensor module works as intended.
	Overall test result:		✓			Module has passed the unit test.

Table 6.2.1: Unit Test for LIDAR Module.

Test Writer: Priyank Kalgaonkar.						
Test Case Name:		Test Motor Control Shield and BLDC Motor.			**Test ID #:**	SL-UT-02
Description:		This test shall be performed on the S32K with motor shield attached.				
Tester Information						
Name of Tester:		Arjun and Kavya			**Date:**	03/02/2020
System Ver:		N/A			**Time:**	2:45 PM
Setup:		Connect the BLDC motor to the shield and provide an external 12V power supply. Now, connect this setup to your PC using a USB cable. Build and push a lab example (program) to the S32K board setup.				

Step	Action	Expected Result	Pass	Fail	N/A	Comments
1	Verify the functioning of the BLDC motor.	BLDC motor should spin.	✓			Motor was functioning properly.
	Overall test result:		✓			No issues were found.

Table 6.2.2: Unit Test for Motor Control Shield and BLDC Motor.

Test Writer: Priyank Kalgaonkar.						
Test Case Name:		Test NXP S32K144 Microcontroller.			**Test ID #:**	SL-UT-03
Description:		This test shall be performed to check the proper functioning of the S32k144 MCU development board.				
Tester Information						
Name of Tester:		Arjun and Kavya			**Date:**	03/02/2020
System Ver:		N/A			**Time:**	4:00 PM
Setup:		Connect the S32144 MCU to a PC using S32K144's OpenSDAv2 USB port.				

Step	Action	Expected Result	Pass	Fail	N/A	Comments
1	Hold the Reset button on S32K144 MCU while plugging in the cable.	Upon successful connection, the green LED should glow on the S32K144 MCU and should show as 'Bootloader' in Windows Explorer.	✓			Show's Bootloader in Windows Explorer. Green LED lights up.
	Overall test result:		✓			MCU Board powers on.

Table 6.2.3: Unit Test for NXP S32K144-EVB MCU Development Board.

6.3 Integration and Testing - System Acceptance Test

Test Writer: Priyank Kalgaonkar.						
Test Case Name:		Sound localization system final test			**Test ID #:**	SL-AT-01
Description:		Test the sound capture, sampling and filtering process on the final system.				
Tester Information						
Name of Tester:		Arjun, Kavya, Priyank			**Date:**	03/09/2020
System Ver:		ACC Ver. 1.1			**Time:**	01:05 PM
Setup:		This test shall be performed on the final system after the NXP S32K144 is powered on, software from S32 Design Studio is flashed to it and all external connections to LIDAR sensor and BLDC motor are properly implemented as planned. No special setup is required.				
Step	**Action**	**Expected Result**	**Pass**	**Fail**	**N/A**	**Comments**
1	Place an object in front of the LIDAR Sensor.	Distance measurement should be seen in TeraTerm/S32DS output terminal.	✓			Works as intended.
2	Now, slowly change the distance of the object. Then change the distance rapidly.	The change in distance should be seen immediately in the output terminal. Motor should change speed and spin accordingly.	✓			Works as intended.
	Overall test result:					No issues found during acceptance testing.

Table 6.3.1: System Acceptance Test

This test covers all of the functionality of the ACC system. Step 1 verifies that data acquisition is successful by S32K from the LIDAR module. Step 2 verifies the software algorithm designed by us in S32 Design Studio and MATLAB Simulink works as intended and the motor changes speed when either the distance between the object and LIDAR sensor module is increased or decreased.. No issues were found during system acceptance testing.

6.4 Final Functional Test

The final functional tests were completed after software was fully developed, all of the hardware had been assembled, and the two had been integrated together. The tests that were carried out are outlined as follows:

1. Action: **Place an object very close to the LIDAR sensor module.**

Expected Result: BLDC motor should not spin.

2. Action: **Remove object from front of the sensor module.**

Expected Result: BLDC motor should spin at full predetermined speed (rpm).

3. Action: **Test with varying speed by changing (increasing and decreasing) distance between the object and the sensor gradually.**

Expected Result: BLDC motor should change speed (rpm) and correspond to the change in distance.

These tests are intended to confirm that the system has been successfully integrated, and now is ready for full system test by engineering. Section 6.4 provides the test results of the integration and testing carried out as aforementioned.

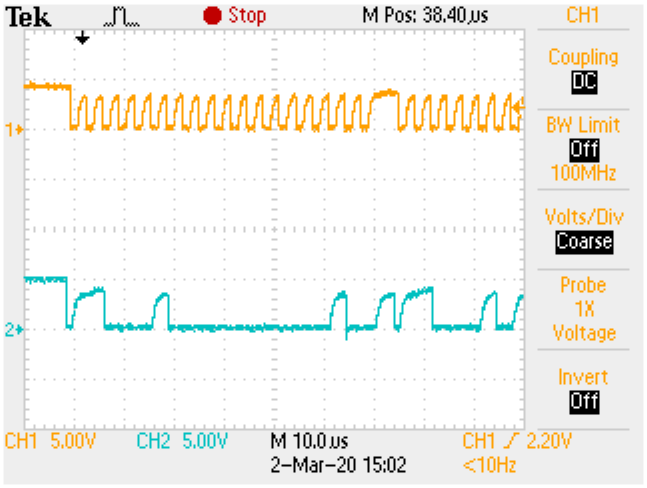


Figure 6.4.1: Oscilloscope Screenshot to Confirm I^2C Communication.

7.0 ECONOMICS

The system development life cycle of our Adaptive Cruise Control Using LIDAR and CAN Protocol project is composed of a number of clearly defined work phases, such as: brainstorming, design, implementation, testing and delivering the final product. Like anything that is prototyped, the system development life cycle aims to produce high quality systems to meet professional engineering standards, based on the need for an easy-to-use product within scheduled time frame and budget estimates.

7.1 Cost of Material

The budget includes costs for all hardware requirements for building the prototype including the NXP S32K144-EVB development boards, sensors and other related components. Following is a table of cost of individual components:

Component	Price/Component	Qty. Purchased	Total
NXP S32K144-EVB	$77.58	2	$155.16
NXP DEVKIT -MOTORGD	$71.50	1	$71.50
BLDC Motor	$49.00	1	$49.00
Garmin LIDAR-Lite	$129.00	1	$129.00
Total Billable Material		5	$404.66

Table 7.1.1: Cost of Billable Material

7.2 Engineering Labor Hours

In determining overall cost estimates, engineering labor hours also need to be taken into consideration. Table 7.2.1 gives an estimate on the labor hours for each activity below:

Activity	(In Hours)
Meeting 4 days/week (3 group members)	48
Critical Design Review	5
Software Development	25
Hardware	5
System Integration and Testing	12
Finalize Implementations	15
Total Hours	110

Table 7.2.1 Table of Engineering Labor (In Hours)

7.3 Specialized Facilities and Resources

Implementation of this project requires resources and a location where these resources are available. Table 9.4 shows the specialized facilities that were used for research and development of the project.

Facility	Location	Resources	Use
SL111: ECE595 Lab	Engineering Science & Technology (SL) Building at IUPUI.	Hardware such as voltmeters, oscilloscopes, soldering station, wires and breadboards.	Testing purposes.

Table 7.3.1 Table of Specialized Facilities and Resources

8.0 PROJECT MANAGEMENT

8.1 Pre-Defined Schedule (Gantt Chart)

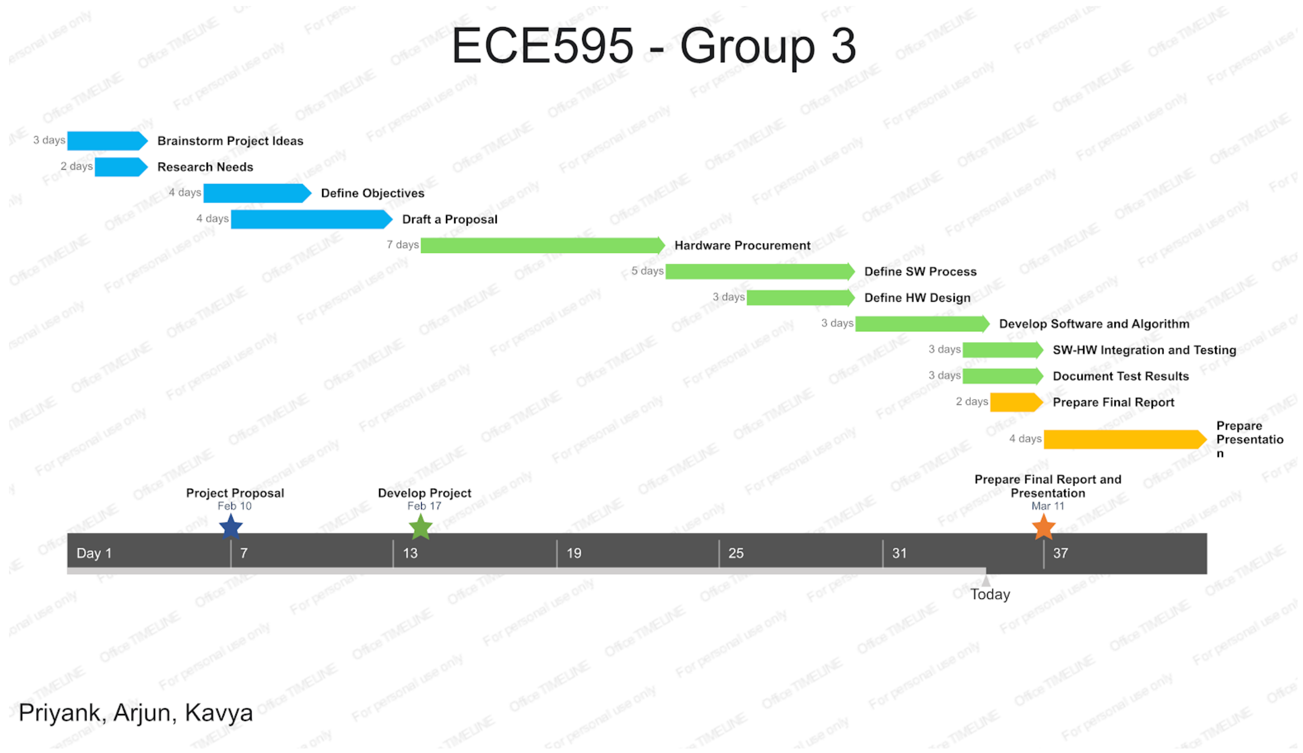


Figure 8.1.1: Project Schedule - Gantt Chart

8.2 Risk Management

There were several risks identified with our project. The first major risk was that various project deadlines such as development, integration, testing, and final building of the prototype would not be met. This was mitigated by creating a detailed plan including a Gantt chart as shown above, and setting up efficient communication channels between the team members to manage our time correctly and efficiently.

Another risk was not having the critical components available in a timely fashion. Shipments could have easily been delayed, and parts could get damaged during shipping. This risk was avoided by organizing and selecting all the components necessary for our design as early as possible. Once this was complete, all the components were ordered at once. This gave enough time for complications and troubleshooting, which included parts that did not work. We avoided real complications by using good time management.

Additionally, our group ran the risk of not complying with specific standards for hardware components. We were able to avoid this risk by purchasing all hardware components off the shelf.

9.0 CONCLUSION

The objective of the Adaptive Cruise Control Using LIDAR system is to enable the traditional manually operated cruise control system to automatically slow down or speed up the vehicle up to a speed limit set by the driver.

Our design of this system utilizes two microcontroller units, a Garmin LIDAR module which is compact and high-performance optical distance measurement sensor, and Brushless DC (BLDC) electric motor. The result is improved driver's comfort, and safety of the vehicle and other traffic on the road by minimizing rear-end collisions.

The Adaptive Cruise Control Using LIDAR prototype requires the system to process the data captured by the LIDAR sensor module in real-time and quickly (<1s), automatically control the motor's speed using data captured by LIDAR, and improve safety and ride comfort whilst maintaining or improving fuel economy. The requirement was partially fulfilled since S32K144 MCU is powered by a 32-bit ARM Cortex M4F core running up to 112MHz, the data should be processed with acceptable delay. The two MCUs, in conjunction, were able to process data and control the speed (rpm) of the BLDC motor in real-time as intended.

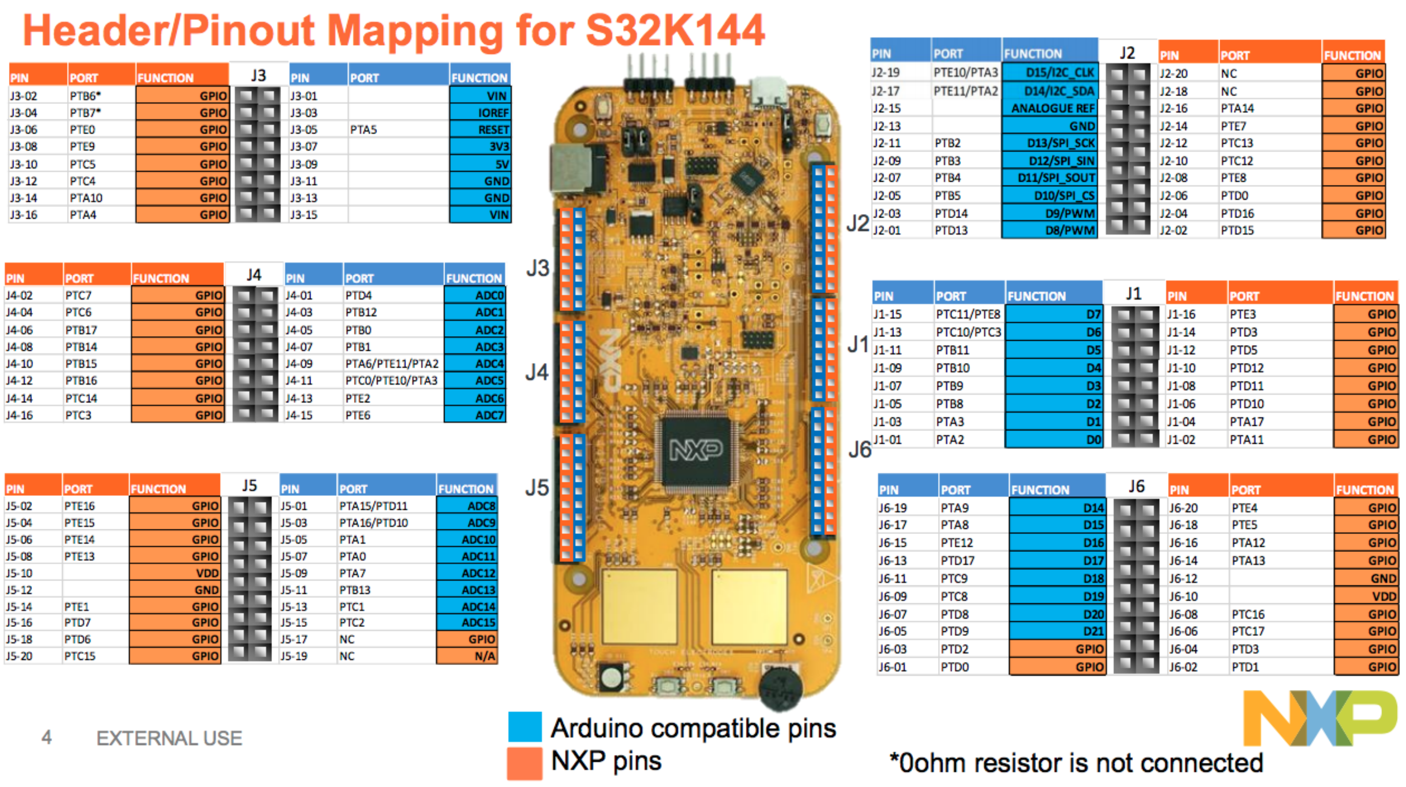
In conclusion, our design of Adaptive Cruise Control Using LIDAR prototype utilizes hardware and multiple-access protocol such as I^2C communication protocol and UART to control the speed of the BLDC motor in real-time with minimal delay. For future work, we recommend using CAN communication protocol combining our system with cameras, multiple sensors and computer vision processing to pave way for this system into the autonomous world of trucks and automobiles.

10.0 REFERENCES

1. Rear-End Collisions Are the Most Frequent Type of Collision. (July 11, 2019). Retrieved from https://www.daveabels.com/rear-end-collisions-frequent-type-collision/.
2. Garmin, & Garmin Ltd. (n.d.). Garmin LIDAR-Lite v3: GPS Sensors. Retrieved from https://buy.garmin.com/en-US/US/p/557294#specs
3. NXP Semiconductors N.V. (n.d.). S32K144 Evb Quick Start Guide [4.3]. Retrieved from https://www.nxp.com/docs/en/quick-reference-guide/S32K144EVB-QSG.pdf
4. DEVKIT-MOTORGD: Low-Cost Motor Control Solution for DEVKIT Platform. (n.d.). Retrieved from https://www.nxp.com/design/development-boards/automotive-development-platforms/hardware-tools-accessories/low-cost-motor-control-solution-for-devkit-platform:DEVKIT-MOTORGD
5. Wikipedia contributors. (2020, March 3). Brushless DC electric motor. In *Wikipedia, The Free Encyclopedia*. Retrieved 23:25, March 9, 2020, from https://en.wikipedia.org/w/index.php?title=Brushless_DC_electric_motor&oldid=943711971
6. Garmin Ltd. (2016). Lidar Lite v3 Operational Manual and Technical Specifications. Retrieved from http://static.garmin.com/pumac/LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf
7. Wikipedia contributors. (2020, March 06). Universal asynchronous receiver-transmitter. In Wikipedia, The Free Encyclopedia. Retrieved 04:14, March 06, 2020, from https://en.wikipedia.org/w/index.php?title=Universal_asynchronous_receiver-transmitter&oldid=944849875

11.0 APPENDICES

11.1 Appendix A - Component Pinouts



Header/Pinout Mapping for S32K144

PIN	PORT	FUNCTION	J3	PIN	PORT	FUNCTION
J3-02	PTB6*	GPIO		J3-01		VIN
J3-04	PTB7*	GPIO		J3-03		IOREF
J3-06	PTE0	GPIO		J3-05	PTA5	RESET
J3-08	PTE9	GPIO		J3-07		3V3
J3-10	PTC5	GPIO		J3-09		5V
J3-12	PTC4	GPIO		J3-11		GND
J3-14	PTA10	GPIO		J3-13		GND
J3-16	PTA4	GPIO		J3-15		VIN

PIN	PORT	FUNCTION	J4	PIN	PORT	FUNCTION
J4-02	PTC7	GPIO		J4-01	PTD4	ADC0
J4-04	PTC6	GPIO		J4-03	PTB12	ADC1
J4-06	PTB17	GPIO		J4-05	PTB0	ADC2
J4-08	PTB14	GPIO		J4-07	PTB1	ADC3
J4-10	PTB15	GPIO		J4-09	PTA6/PTE11/PTA2	ADC4
J4-12	PTB16	GPIO		J4-11	PTC0/PTE10/PTA3	ADC5
J4-14	PTC14	GPIO		J4-13	PTE2	ADC6
J4-16	PTC3	GPIO		J4-15	PTE6	ADC7

PIN	PORT	FUNCTION	J5	PIN	PORT	FUNCTION
J5-02	PTE16	GPIO		J5-01	PTA15/PTD11	ADC8
J5-04	PTE15	GPIO		J5-03	PTA16/PTD10	ADC9
J5-06	PTE14	GPIO		J5-05	PTA1	ADC10
J5-08	PTE13	GPIO		J5-07	PTA0	ADC11
J5-10		VDD		J5-09	PTA7	ADC12
J5-12		GND		J5-11	PTB13	ADC13
J5-14	PTE1	GPIO		J5-13	PTC1	ADC14
J5-16	PTD7	GPIO		J5-15	PTC2	ADC15
J5-18	PTD6	GPIO		J5-17	NC	GPIO
J5-20	PTC15	GPIO		J5-19	NC	N/A

PIN	PORT	FUNCTION	J2	PIN	PORT	FUNCTION
J2-19	PTE10/PTA3	D15/I2C_CLK		J2-20	NC	GPIO
J2-17	PTE11/PTA2	D14/I2C_SDA		J2-18	NC	GPIO
J2-15		ANALOGUE REF		J2-16	PTA14	GPIO
J2-13		GND		J2-14	PTE7	GPIO
J2-11	PTB2	D13/SPI_SCK		J2-12	PTC13	GPIO
J2-09	PTB3	D12/SPI_SIN		J2-10	PTC12	GPIO
J2-07	PTB4	D11/SPI_SOUT		J2-08	PTE8	GPIO
J2-05	PTB5	D10/SPI_CS		J2-06	PTD0	GPIO
J2-03	PTD14	D9/PWM		J2-04	PTD16	GPIO
J2-01	PTD13	D8/PWM		J2-02	PTD15	GPIO

PIN	PORT	FUNCTION	J1	PIN	PORT	FUNCTION
J1-15	PTC11/PTE8	D7		J1-16	PTE3	GPIO
J1-13	PTC10/PTC3	D6		J1-14	PTD3	GPIO
J1-11	PTB11	D5		J1-12	PTD5	GPIO
J1-09	PTB10	D4		J1-10	PTD12	GPIO
J1-07	PTB9	D3		J1-08	PTD11	GPIO
J1-05	PTB8	D2		J1-06	PTD10	GPIO
J1-03	PTA3	D1		J1-04	PTA17	GPIO
J1-01	PTA2	D0		J1-02	PTA11	GPIO

PIN	PORT	FUNCTION	J6	PIN	PORT	FUNCTION
J6-19	PTA9	D14		J6-20	PTE4	GPIO
J6-17	PTA8	D15		J6-18	PTE5	GPIO
J6-15	PTE12	D16		J6-16	PTA12	GPIO
J6-13	PTD17	D17		J6-14	PTA13	GPIO
J6-11	PTC9	D18		J6-12		GND
J6-09	PTC8	D19		J6-10		VDD
J6-07	PTD8	D20		J6-08	PTC16	GPIO
J6-05	PTD9	D21		J6-06	PTC17	GPIO
J6-03	PTD2	GPIO		J6-04	PTD3	GPIO
J6-01	PTD0	GPIO		J6-02	PTD1	GPIO

🟦 Arduino compatible pins
🟧 NXP pins

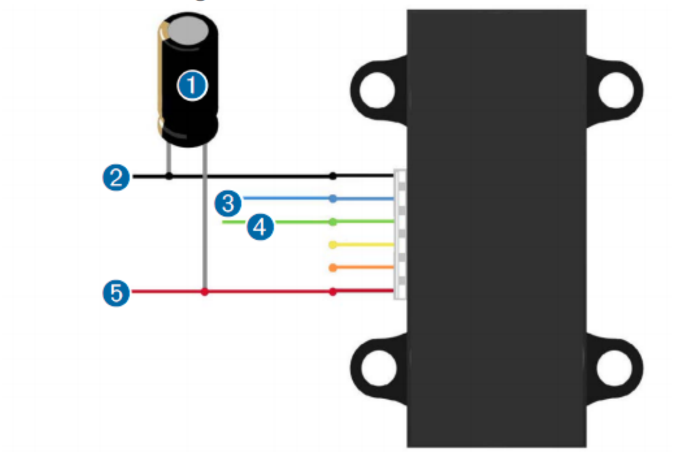
*0ohm resistor is not connected

4 EXTERNAL USE

Figure 11.1.1: NXP FRDM K64F Header Pinout Diagram[3]

I2C Connection Diagrams

Standard I2C Wiring



Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	Power ground (-) connection	Black wire
3	I2C SDA connection	Blue wire
4	I2C SCA connection	Green wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

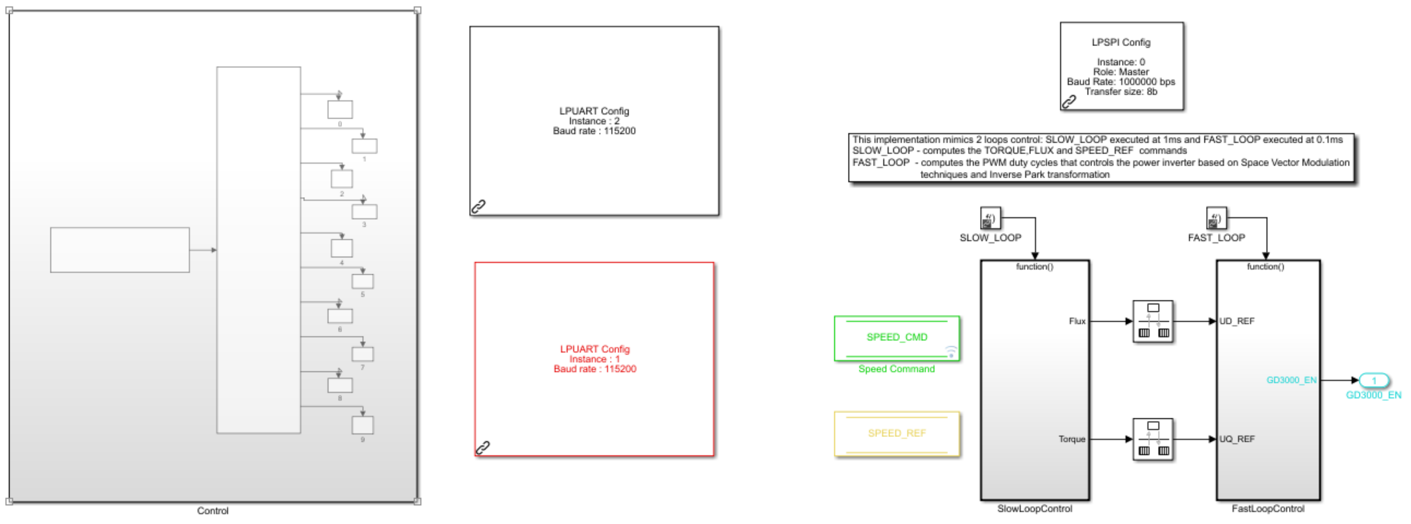
Figure 11.1.2: Garmin LIDAR-Lite v3 Pinout[6]

11.2 Appendix B - S32 Design Studio C Programming Code

```
/*****************************************************************************
Project Name: Automatic Cruise Control System Using LIDAR.
Project By: Priyank Kalgaonkar, Kavyashree Prasad S P, Arjun Narukkanchira
Anilkumar.
Course: ECE595 - Autonomous Embedded Systems - Spring 2020
*****************************************************************************/
#include "Cpu.h"
#include "pin_mux.h"
#include "osif1.h"
#include "dmaController1.h"
#include "clockMan1.h"
#include "lpuart1.h"
#include "S32K144.h"
#include "MPL3115A2.h"
#include "I2C.h"
#include <string.h>
#include <stdint.h>
#include <stdbool.h>
/*****************************************************************************
Local function prototypes
*****************************************************************************/
void SIRC_div(void);
void PORT_conf(void);
/****************************************************************************
Global variables
*****************************************************************************/
uint8_t  ready;
uint8_t  err_0, err_1, err_2, err_3, err_4, err_5, err_6;
uint8_t  status[1],altitude[2];
uint16_t distance_lidar;
volatile int exit_code = 0;

int main(void)
{
  SIRC_div();
  PORT_conf();
  LPI2C0_clock();
  LPI2C0_IRQs();
  LPI2C0_init();

  bool strReceived = false;
  uint8_t    buffer [255]   ={0,};
  uint8_t  i = 0;
  uint32_t bytesRemaining;
#ifdef PEX_RTOS_INIT
```

```
  PEX_RTOS_INIT();
#endif

  CLOCK_SYS_Init(g_clockManConfigsArr, FSL_CLOCK_MANAGER_CONFIG_CNT,
                 g_clockManCallbacksArr, FSL_CLOCK_MANAGER_CALLBACK_CNT);
  CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_FORCIBLE);

  PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);
  PORT_conf();

  /* Initialize LPUART instance */
  LPUART_DRV_Init(FSL_LPUART1, &lpuart1_State, &lpuart1_InitConfig0);

  while(LPUART_DRV_GetTransmitStatus(FSL_LPUART1, &bytesRemaining) !=
  LPUART_STAT_SUCCESS);

   /* Infinite loop Implemented Here. Loop is as follows:
   *    - Receive data from user
   *    - Echo the received data back
   */

  SIRC_div();
  while (1)
    {
        err_0 = LPI2C0_read(MPL3115A2_R, DR_STATUS_REG, status, 1);
        distance_lidar =0;
        if(!(status[0] & 0x01))
        {   err_2 = LPI2C0_write(MPL3115A2_W, 0x00, 0x04);
            if(!(err_5 = LPI2C0_read(MPL3115A2_R, OUT_P_MSB_REG, altitude,
            2)))
            {
                distance_lidar     |= (altitude[0] << 8);
                distance_lidar     |= (altitude[1]);
                sprintf(buffer, "%u\r\n", distance_lidar);
                i = strlen((char *)buffer);
                LPUART_DRV_SendData(FSL_LPUART1, buffer, i);
            }
        }
      buffer[i] = 0;
      while(LPUART_DRV_GetTransmitStatus(FSL_LPUART1, &bytesRemaining) !=
      LPUART_STAT_SUCCESS);
      strReceived = false;
      i = 0;
    }
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();
/* Startup of the selected RTOS. Macro is defined by the RTOS component. */
  #endif
  for(;;) {
    if(exit_code != 0) {
      break;
    }
  }
  return exit_code;
}

/******************************************************************************
Function Name : SIRC_div_8MHz
Notes         : SIRCDIV2 divides by 1 (8MHz)
                SIRCDIV1 divides by 1 (8MHz)
******************************************************************************/
void SIRC_div(void)
{

    SCG->SIRCCSR &= ~ (1 << 24);
    SCG->SIRCCSR &= ~ (1 << 0);
    SCG->SIRCDIV |= 0x0101;
    SCG->SIRCCSR |= (1 << 0);
    while((SCG->SIRCCSR & (1 << 24)) == 0);
}

/******************************************************************************
Function Name : init_port
******************************************************************************/
void PORT_conf(void)
{
    //Peripheral Clock Controller (PCC)
    PCC-> PCCn[PCC_PORTA_INDEX] = PCC_PCCn_CGC_MASK;

    PORTA->PCR[2] |= PORT_PCR_MUX(3);
    PORTA->PCR[2] |= PORT_PCR_PS(1);
    PORTA->PCR[2] |= PORT_PCR_PE(1);
    PORTA->PCR[3] |= PORT_PCR_MUX(3);
    PORTA->PCR[3] |= PORT_PCR_PS(1);
    PORTA->PCR[3] |= PORT_PCR_PE(1);

    PCC-> PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;

    PORTD->PCR[0] = 0x00000100;
    PTD-> PSOR |= (1 << 0);
    PTD->PDDR |= (1 << 0);
}
```

11.3 Appendix C - MATLAB Simulink Graphical Block Program



This implementation mimics 2 loops control: SLOW_LOOP executed at 1ms and FAST_LOOP executed at 0.1ms
SLOW_LOOP - computes the TORQUE,FLUX and SPEED_REF commands
FAST_LOOP - computes the PWM duty cycles that controls the power inverter based on Space Vector Modulation techniques and Inverse Park transformation

Figure 11.1.3: MATLAB Simulink Graphical Block Program.