IMPLEMENTATION OF CAN BUS INTERFACE FOR THE MULTIPURPOSE BOARD

A PROJECT REPORT

Submitted by

P. SAI SAMYUKTA (1602-17-734-036)

Under the Guidance of

Mr. Madhav Tenneti

Chief Product Architect, T-Works

Along with

Dr. K. Ravi Kumar

Professor

submitted in partial fulfilment for the award of the degree of

Bachelor of Engineering

IN

ELECTRICAL AND ELECTRONICS ENGINEERING

Vasavi College Of Engineering

(Affliated to Osmania University)

Ibrahimbagh, Hyderabad-500031

July, 2021

TO WHOMSOEVER IT MAY CONCERN

Palle Sai Samyukta,
Flat no. 33, Lavender,
L & T Serene County, Telecom Nagar,
Gachibowli, Hyderabad,
Telangana – 500032
Contact: 8374707834

Regarding Internship Completion

We are pleased to confirm that P. Sai Samyukta (Roll no: 1602-17-734-036), student of Vasavi College of Engineering pursuing degree in B.E Electrical and Electronics Engineering, has completed an internship with T Works foundation. During the course of the internship, Sai Samyukta worked on the project titled – "**Implementation of CAN Bus Interface for the multi-purpose board**" and reported to Mr. Madhav Tenneti, Chief Product Architect and Project Manager.

Details of Internship:

Type of Assignment: IoT Internship
Start date of the Internship: **18/03/2021**
End date of the internship: **21/06/2021**
Brief of the Project / Assignment: Sai Samyukta worked on a project aimed at testing Electric Vehicles. As part of the project, she has worked on the implementation of MCP2515-CAN, the creation of a register-based driver for ADS1115, understanding the wiring of Master-Slave architecture in RS-485 interface.

During the internship, Sai Samyukta was dedicated, diligent and performed at a satisfactory standard, and we believe she made a valuable contribution to the team.

We wish Sai Samyukta all the best in the future endeavors.

From T-Works Foundation:

Sanjay Gajjala
Director, Operations

CERTIFICATE

This is to certify that this project report entitled "**IMPLEMENTATION OF CAN BUS INTERFACE FOR THE MULTI-PURPOSE BOARD**" *by* **P. SAI SAMYUKTA (1602-17-734-036),** submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Electrical and Electronics Engineering of the Vasavi College of Engineering, Hyderabad, during the academic year 2020-21, is a bonafide record of work carried out under my guidance and supervision.

Dr. K. Ravi Kumar,

Professor, Dept of EEE,

Vasavi College of Engineering, Hyderabad

ACKNOWLEDGEMENT

I want to express my sincere acknowledgment to my project guide, Mr. Madhav Tenneti, Chief Product Architect, T-works, and even Dr. K. Ravi Kumar, Professor, Department of Electrical and Electronics Engineering. I am grateful for their kind cooperation and encouragement that helped in the completion of this project. It would have been difficult to complete this project without the enthusiastic support, encouragement, and inciteful advice given by them.

I am also grateful to T-Works Foundation for giving me this opportunity and constantly helping me throughout the project.

It is my privilege to thank all project review committee members for their continuous guidance and monitoring.

I am thankful to Dr. Ch. V. S. S. Sailaja, Associate professor of Electrical and Electronics Engineering Dept., and Dr. M. Chakravarthy, Professor & Head of Department of Electrical and Electronics Engineering Dept., Vasavi College of Engineering, for their support and valuable suggestions.

I would also like to express my sincere gratitude to the management of Vasavi College of Engineering for their encouragement, facilities provided, and support.

Finally, I want to mention all my family members and friends for their help and encouragement during the project work.

P. Sai Samyukta (1602-17-734-036)

ABSTRACT

In a modern automotive system, there's a minimum of 30+ Electronic Control Units (ECUs) and a few luxury vehicles claim to have 90+ ECUs. Hence, there is rapid growth for communications between all these ECUs which is crucial for the overall implementation of the vehicle's features. This thesis will give a brief on the real-time In-Vehicle Communication protocols like Local Interconnected Network, FlexRay, and Controller Area Network. Then, will give a extensive information about CAN, followed with how to set up communication network using CAN.

Even though, this thesis talks only about the implementation on CAN in a car, CAN is actually being used in railways, aircrafts, telescopes and in medical fields too.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. CHAPTER I - INTRODUCTION

1.1. SIGNIFICANCE

A few decades ago, vehicles essentially consisted of hydraulic or mechanical components. However, the rise of electronic components resulted in vehicles shifting to electronic systems. Since there were a number of advantages that came with electronic systems like an increase in reliability and the chance to implement complex functions.

At the initial stages of automotive electronics, each function was a standalone system, which is known as Electronic Control Unit. But, for a vehicle to function properly each function must be interconnected to each other. For example, a vehicle's speed estimated by the engine controller is necessary to understand how much should be the steering effort, so as to control the suspension. This gave rise to point-to-point links for data exchange. However, this approach was not efficient, because, for an 'n' node system, n^2 number of links were required, which increased weight, complexity, and cost.

Hence, there was a need for multiplexed communication protocols. One such protocol which is not only reliable but also considerably of less cost is Controller Area Network.

1.2. OBJECTIVE

The main aim of the project is to set up the in-vehicle communication by implementing Controller Area Network (CAN) Bus protocol. And some of the other objectives are –

- It should be able to do data logging with precision;
- There should be a negligible delay in the transmission of data;
- While, receiving the data it should be able to recognise the main and error messages;

- It must be a reliable network, with less execution time and storage space, because even safety system will be based on this communication.

1.3. OUTLINE OF THE THESIS

This section's purpose is to help the reader understand the flow of the content.

- It starts with understanding the theoretical topics, which is covered in chapter 2– "**Theoretical Concepts**". Here, the reader will be able to understand more about communication protocols, and what are the protocols being used in real-time. It also explains on why CAN protocol is being implemented in this project.
- Next, chapter – "**Hardware Implementation**" is all about the hardware components used. It will discuss every aspect of those components required for doing this project. Further, it will talk about the Block Diagram and Circuit Diagram.
- After that, "**Software Implementation**" chapter is for knowing the software being used and also on how to interface them along with a few examples to get more clarity.
- Following chapter, "**Flowchart and Code**" is to understand the code which is executing the project.
- Lastly, there is a chapter named "**Outcomes**", here the reader will get to know the results and future scope.

2. CHAPTER II – THEORETICAL CONCEPTS

The most popularly used In-Vehicle communication protocols are Controlled Area Network (CAN), Local Interconnected Network (LIN), and FlexRay. Since all of these are Serial Communication protocols, let's first understand about Serial Communication.

2.1. SERIAL COMMUNICATION

Serial communication is a widely used method for transferring information between computing devices and peripheral devices. Most devices in everyday life such as laptops, PCs, cell phones run on the serial protocol. First of all, the protocol is a set of rules used to establish secure and reliable communication. In serial communication, data is sent bit-by-bit over two wires, known as sender and receiver.

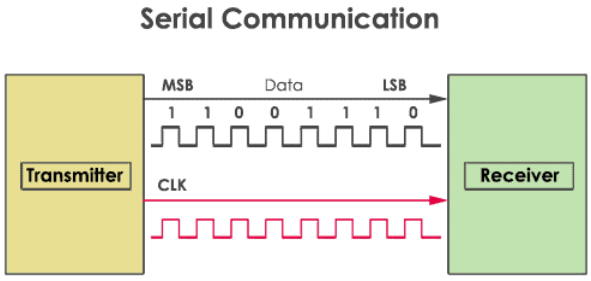

Figure 2.1: Serial Communication

There is another type of communication called parallel communication, in which a chunk of data is transferred at once. In this method, to transmit 'n' data bits, 'n' lines are required. Therefore, although data can be transferred faster as compared with serial communication, there is a problem that the installation cost is high, and it is not feasible for long-distance communication.

A number of the commonly used interfaces are LIN, CAN, I2C, etc. Now, let's look into some these commonly used protocols and then understand why in this project, CAN is being employed.

2.2. LOCAL INTERCONNECTED NETWORK (LIN)

2.2.1. Overview

It is a serial bus that is mainly used for mechatronics nodes of automotive applications. This protocol is relatively less expensive for serial communication which supports in-vehicle data transfer. It has a single master and multi-slaves bus architecture network. LIN has a feature mechanism that allows the devices to enter sleep mode when they are in idle mode. Hence, power conservation is achieved potentially. The number of LIN nodes that are connected through a physical cable is called the LIN cluster. A single cluster can have only one master node and multiple slave nodes (max up to 16 nodes). Today LIN being a safe, cheap, and efficient protocol it is present in almost all automobiles. Though LIN is not the best communication protocol, it is still a good alternative where bandwidth and speed are not the major concern.

2.2.2. Pros

- Simple, Low-cost interface
- Single-wires based wiring method hence reduces the cost and implementation complexity.

2.2.3. Cons

- Low speed, hence not ideal for safety or other important application systems in the vehicle.
- Communication gets initiated by the master, so if the master fails then the whole bus gets failed.

2.2.4. Applications in Vehicle

- Trunk Switches
- Electric Windows
- Door Locks
- Lights
- Power Seat Controllers

2.3. FLEXRAY

2.3.1. Introduction

It is the most reliable communication protocol within the automotive field. The Flex ray bus works with a time signal which is of two parts - dynamic and static segments. The Flex ray nodes are interconnected by using the twisted pair of wires. The FlexRay protocol is a unique time-triggered protocol that provides options for deterministic data arriving in a predictable timeframe (down to a microsecond). FlexRay manages multiple nodes employing a scheme known as Time Division Multiple Access (TDMA). Each FlexRay node is synchronized to a clock and needs to wait for its turn to write on the bus. Because the timing is consistent in a TDMA scheme, FlexRay can guarantee the determinism or the consistency of data to nodes within the network. Hence, the low priority data waits in the queue".

2.3.2. Advantages

- Highly reliable
- Can handle any type of network configuration
- Relatively faster than the other three protocols

2.3.3. Disadvantages

The only disadvantage is its expensive

2.3.4. Applications in Vehicle

- Safety Components
- Active Suspension System
- High-Performance Transmissions
- Traction Control Systems

2.4. CONTROLLER AREA NETWORK (CAN) BUS

2.4.1. Introduction

CAN is a serial communication protocol designed for automotive and industrial applications. It enables in-vehicle communication between microcontrollers and devices without a host computer, to control and acquire data. These devices are also known as Electronic Control Units (ECU). There are up to 70 ECUs in an automotive CAN BUS system like an energy control unit, airbag, audio-video system, etc. Hence, CAN BUS consists of several linked ECUs within a vehicle that communicates with each other based on a broadcast. Every ECU intercepts every broadcast but singly decides whether or not to react to it.

In comparison with other communication protocols like UART, SPI, and I2C, the CAN BUS communication protocol gives much more reliability. Since we need automotive communication protocols that are used to transfer vital data, for example, throttle position in a vehicle, failure of communication or loss of data can lead to critical failures.

2.4.2. Role of CAN

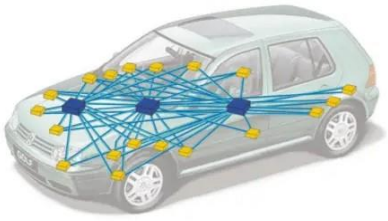

Figure 2.2: Car with 3 control units using point-to-point network

Without CAN-BUS protocol, the in-vehicle communication happens using direct, point-to-point analog signal lines (as discussed in Significance of this thesis). However, this method is not only time-consuming but also is messy with all the additional wiring as shown in the picture above. And the other disadvantages are - unreliable communication between devices and an increase in cost.

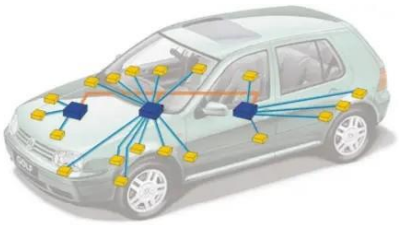

Figure 2.3: Car with 3 control units using CAN communication

The CAN-BUS protocol eliminates the need for all of this complex wiring because electronic devices can instead communicate with one another employing a single multiplex cable that connects every node on the network to the dashboard, as shown in the figure above. Thereby ensuring each electronic module that is part of the vehicle, to receive data from all sensors and actuators. Moreover, since an ECU can use data from another ECU there is no necessity for installing the same sensors in multiple devices. Thus, a vehicle can have a simple design framework to convey more information.

2.4.3. Wiring

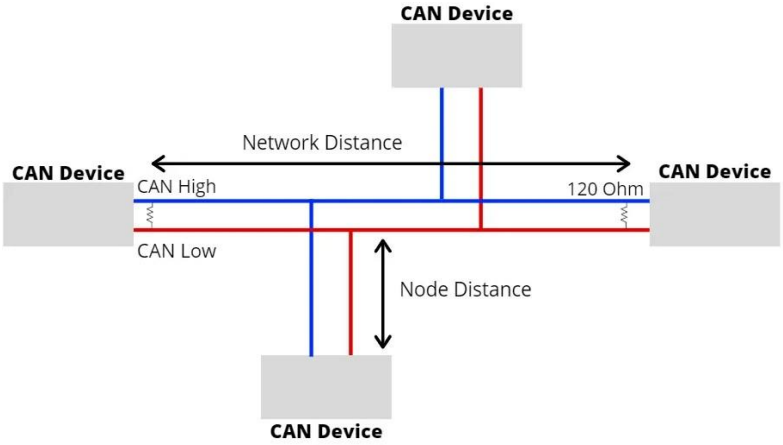

Figure 2.4: CAN Bus Wiring

The CAN-BUS protocol is used for bi-directional data transmission, which is possible via the two wires - CAN_H (CAN High) & CAN_L (CAN Low). These wires act as a differential line, i.e., the CAN signal value is either 0 or 1, which is based upon the potential difference between the two wires. For example, for a negative potential difference, it is 0. For CAN termination a single 120 Ohm resistance is usually used at the ends of the CAN network which is called node termination resistance.

2.4.4. Speed and Range

The CAN-BUS protocol's communication rates**,** range from 10kpbs to 1Mbps, depend**s** on the length of cable utilised. **For** faster transmission, the **length must be** shorter, and vice versa. For example, at 50 metres, the speed is around 1Mbps, whereas at 1000 metres, it is approximately 50kpbs.

2.4.5. CAN Message

Let's have a look at the frames that were sent over the network. The CAN message is divided into two parts: the identifier and the data, which aid in the transmission of messages via the CAN BUS. The identifier will be used to identify CAN devices in the CAN network, whereas data is the sensor or control data that must be transmitted from one device to another. The identifier or CAN ID length depends on the kind of CAN protocol used, i.e., for Standard CAN it is 11 bits, while for Extended CAN it is 29 bits. And, the data can be anywhere from 0 to 8 bytes.

2.4.6. Advantages

- **Low Cost -** As discussed before, CAN implements multiplex wiring rather than using direct analog lines, which helps in reducing errors, weight, and costs.
- **Centralized -** Because CAN BUS offers centralised control over linked electrical devices, it enables centralised fault detection and configuration across all ECUs. The CAN protocol also includes error management, so nodes can check for faults in transmission while keeping an error counter. The protocol, for example, provides various error detection capabilities such as ack error, bit error, CRC error, form error, and so on.
- **Flexible -** When adding or removing equipment, it can be simply operated without any major programming overhead, and there is no need to do large-scale system transformation, which saves a considerable amount of manpower. And, because each CAN-connected ECU receives all sent signals, it can determine whether or not they are significant and respond appropriately.

- **Robust -** When selecting a communication protocol, durability and dependability are critical. You would like the communication protocol to be self-sustaining and resilient for an extended length of time without the need for maintenance. The CAN BUS protocol is resistant to electric disturbances and electromagnetic interference, making it an excellent protocol for automobiles.

- **Efficient -** CAN message frames are prioritised by ID, with the highest priority receiving bus access while making sure that the frames are not interrupted. It also saves time and requires less and simpler wiring thanks to flash programming.

2.4.7. Applications in Vehicle

- Airbags
- Anti-lock Braking System,
- Electric power steering,
- Audio-Video Systems,
- Suspension Systems, etc.

3. CHAPTER III – HARDWARE IMPLEMENTATION

3.1. COMPONENTS

3.1.1. Arduino UNO

Overview –

It is a microcontroller developed by Arduino.cc. The board is built on an 8-bit ATmega328P microcontroller that comes with 2KB SRAM, 32KB of flash memory, and 1KB of EEPROM. Also, it contains other components like crystal oscillator, ATmega16U2, voltage regulator, etc. And, the software used is Integrated Development Environment, generally known as IDE, in which programming can be done using C and C++ language.

Pin Description –

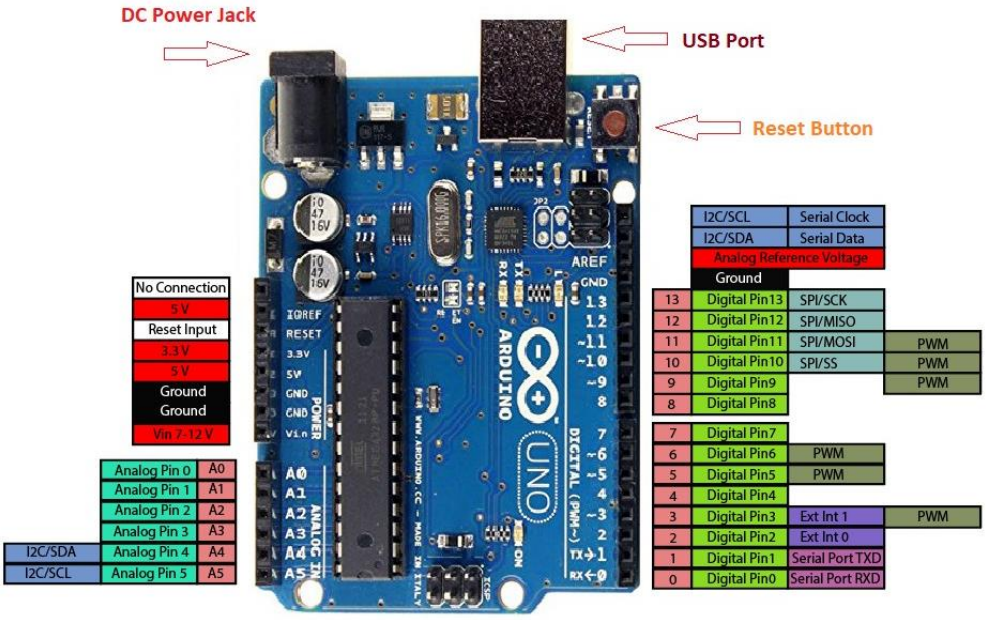

Figure 3.1: Arduino UNO Pin Description

Pin Category	Pin Name	Details
Power	Vin, 3.3V, 5V, and GND	Vin: Input voltage provided through a power jack.

		5V: Provides a output regulated power supply. It can be utilised to power the microcontroller and other board components. 3.3V: The on-board voltage regulator generates a 3.3V supply. The maximum current that can be drawn is 50mA. GND: There are three ground pins provided on board. Note: Vin and Power Jack supports voltage between 7V and 20V
Analog Pins	A0 – A5	These six pins come with a resolution of 10 bits, and supply analog input in the range of 0-5V. However, they can be configured to higher range using analogReference() and AREF pin.
Digital Pins	0 – 13	There are 14 digital pins which are available as input or output pins. These can be connected to external devices.
Pulse Width Modulation (PWM)	3, 5, 6, 9, and 11	Six of the digital pins are configured to provide 8-bit PWM output.
Serial	0 (Rx) and 1 (Tx)	To carry out Serial Communication, Rx receives and Tx transmits TTL serial data.
Serial Peripheral Interface (SPI)	10 (SS), 11 (MOSI), 12	SPI communication is accessed through SPI library; SS - Slave Select

	(MISO), and 13 (SCK)	MOSI - Master Out Slave In MISO - Master In Slave Out SCK - Serial Clock
Two – Wire Interface (TWI)	A4 (SDA) and A5 (SCA)	Used for TWI communication with the help of Wire Library.
External Interrupts	2 and 3	To trigger an interrupt, i.e., by providing LOW value or changing value.
Inbuilt LED	13	A HIGH value on the 13th pin will turn on the built-in LED, while a LOW value will turn it off.
AREF	AREF	Analog Reference pin, used to provide reference voltage for input voltage and analog inputs.
Reset	Reset	Resets the microcontroller, this resetting feature can also be done through programming

Table 3.1: Arduino UNO Pin Description

Technical and Physical Specifications –

Category	**Specifications**
Operating Voltage	5 V
Recommended Input Voltage	7 – 12 V
Input Voltage Limits	6 – 20 V
DC Current on I/O Pins	40 mA
DC Current on 3.3V Pin	50 mA
Communication	I2C, SPI and USART

Maximum Length and Width	2.7 and 2.1 inches.

Table 3.2: Arduino UNO Specifications

ATmega328 Microcontroller –

It has counters, timers, interrupts, PWM, CPU, I/O pins, and relies on 16MHz clock. Below is an Arduino UNO to ATMega328 Pin Mapping, it is useful to know for low-level programming, i.e., coding with registers.

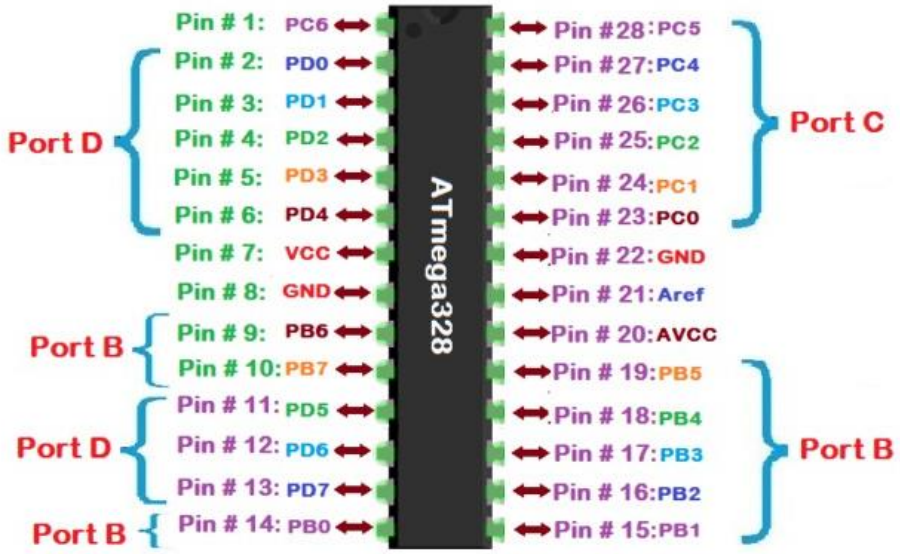

Figure 3.2: ATmega328

Communication –

Arduino can communicate with a computer, another Arduino board, or other microcontrollers. The ATmega328P microcontroller supports UART TTL (5V) serial communication using the Serial pins mentioned in the above Technical Specifications table. An ATmega16U2 on the board communicates serially through USB, which appears as a virtual com port to software on the laptop. Because the ATmega16U2 firmware utilises standard USB COM drivers, no additional driver is required. On Windows, however, a.inf file is necessary. The Arduino Integrated Development Environment (IDE) has a serial monitor for sending and receiving basic textual data to and from the Arduino board The Arduino Board also features two RX and TX LEDs that illuminate when data is transferred via the USB-to-serial chip and USB connection to the computer (not for serial communication on pins 0 and 1).

Serial communication on any of Uno's digital pins is possible using a SoftwareSerial library. Other than Serial Communication, I2C (TWI) and SPI communication is also supported by the ATmega328 microcontroller. Furthermore, this microcontroller comes with a built-in bootloader.

Applications –

UNO has a wide range of applications, some of them are - Home Automation, Digital Electronics & Robotics, Traffic Light Count Down Timer, and Embedded Systems. They are generally used for prototyping Electronic Products and Systems and also for DIY projects.

3.1.2. Arduino Nano

It is also a microcontroller designed by Arduino.cc and based on an 8-bit ATmega328P microcontroller. Hence, it is very much similar to Arduino UNO.

And, here is the **Pin Configuration** of Arduino Nano –

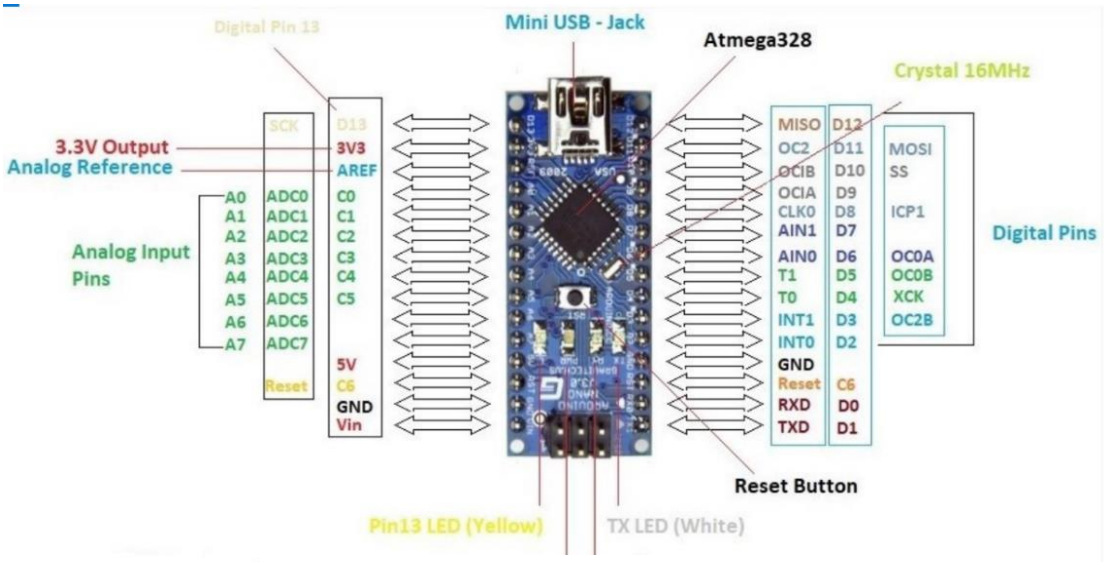

Figure 3.3: Arduino Nano Pin

3.1.3. MCP2515 CAN Module

Overview –

The MCP2515 CAN Bus Controller is a basic Module supporting the CAN Protocol, which is discussed in the previous chapter, version 2.0B and can be

used for communication at 1Mbps. To set up a communication system, there is a need of minimum two CAN Bus Module, but since CAN is a multi-drop network usually there are more than two modules.

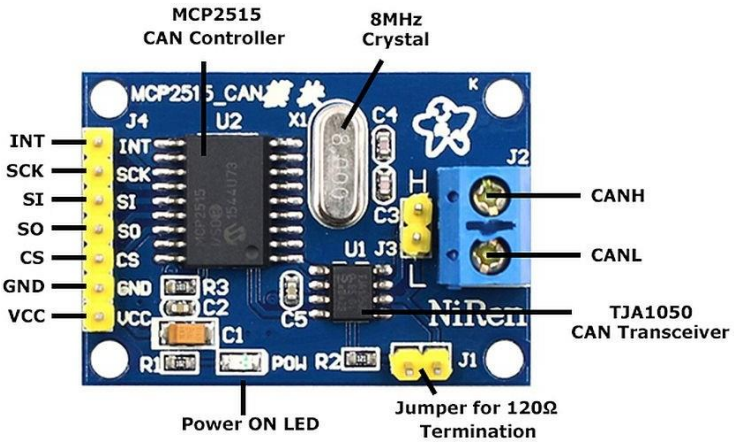

Figure 3.4: MCP2515 CAN Module

MCP2515 CAN Controller IC and TJA1050 CAN Transceiver IC are based on this particular module. The MCP2515 IC includes an inbuilt SPI interface for communicating with microcontrollers, and is an individual CAN Controller.

Features –

- Uses High-speed CAN transceiver TJA1050
- SPI control for expansion of Multi CAN bus interface
- 120Ω terminal resistance
- Has independent key, LED, and Power indicator
- Supports 1 Mbps CAN operation
- Low current standby operation
- Up to 112 nodes can be connected

Specifications –

Category	**Values**
Temperature: Range	-40 to 125 C
Interrupt Outputs	1

Crystal Frequency	8 MHz
Dimensions: L x W (PCB)	41 x 39 mm
Datasheets	MCP2515, TJA1050

Table 3.3: MCP2515 CAN Module Specifications

Schematic –

Let's learn a little more about both the ICs, MCP2515 and TJA1050, before diving into the schematic. The main controller, the MCP2515 IC, is made up of three main subcomponents:

- **CAN Module** is in charge of sending and receiving messages over the CAN Bus;
- **Control Logic** interfaces all of the blocks and manages the setup and operation of the MCP2515
- **SPI Block** is in charge of the SPI communication interface.

The **TJA1050 IC** serves as an interface between the MCP2515 CAN Controller and the physical CAN Bus, collecting data from the controller and relaying it on the bus. The MCP2515 CAN Module's schematic shows how the MCP2515 and TJA1050 ICs are coupled on the module.

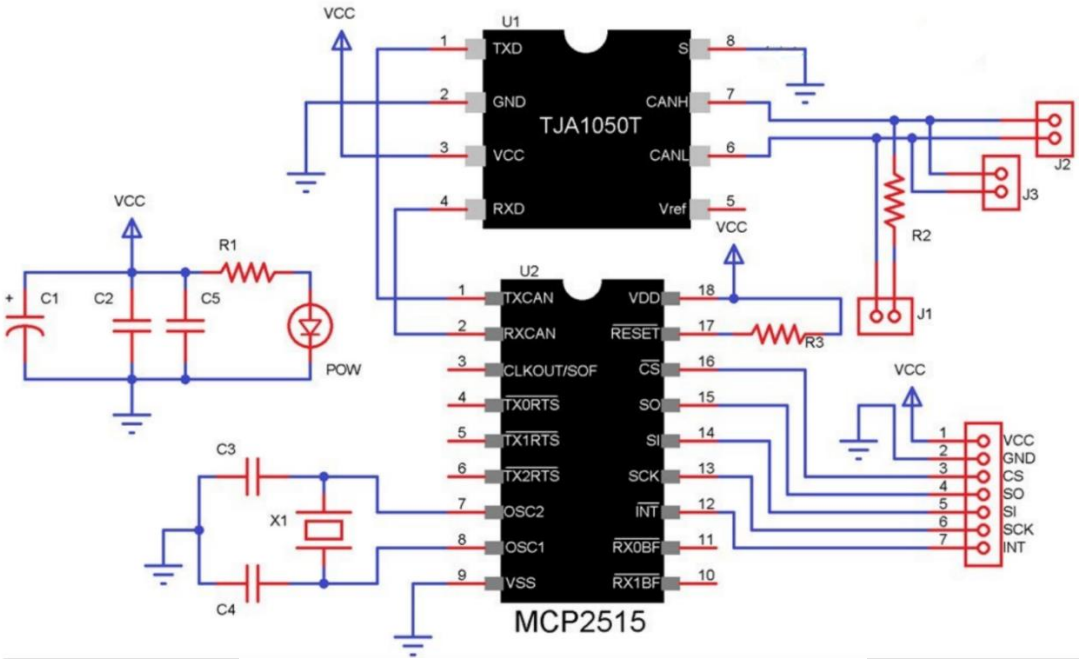

Figure 3.5: MCP2515 CAN Module

Module Connections –

MCP2515 CAN Module	Microcontroller (ATmega328)
INT	Interrupt Pins (INT0 or INT1)
SCK	SPI SCK
SI	SPI MOSI
SO	SPI MISO
CS	SPI SS
GND	Ground
VCC	5 V

Table 3.4: MCP2515 CAN Module Connections

Connect H and L to H and L respectively of other CAN module. And at Header J1 add jumper to include a 120-ohm termination resistor at each end of the CAN Bus

The module has 4 holes for mounting.

MCP2515 Modes of Operation –

There are five modes of operation:

- **Configuration mode:** On powering up the module, this mode is automatically selected. In this mode, all error counters are cleared and it is the only way to change CNF1, CNF2, CNF3 and TXRTSCTRL, Filter registers and Mask registers.
- **Normal mode:** This is the device's default operating mode, in which it monitors all bus messages and generates acknowledge bits, error frames, and other information. This is also the only mode in which the MCP2515 module can send messages over the CAN bus.
- **Sleep mode:** This mode is used to reduce device’s current consumption. Even when MCP2515 is in sleep mode, the SPI interface remains active, thus allowing access to all registers.

- **Listen only mode:** All messages including error messages are received in this mode by MCP2515. It can be used for baud rate detection in hot plugging or bus monitoring applications. And this mode is a silent mode, so during this mode no messages are transmitted (including error flags or acknowledge signals).
- **Loopback mode:** Allows internal communication from the transmission buffers to the received buffers without actually sending messages on the CAN bus. This mode is usually used to design and test the system.

3.1.4. ADS1115

Overview –

The ADS1115 is a precise 16-bit Analog to Digital Converter with four multiplexed inputs. These inputs can be used by themselves, or in pairs for differential measurements. Its high accuracy comes from the internal calibrated reference. And the power supply range of ADS1115 is 2.0V to 5.5V

Figure 3.6: ADS1115 Board

Block Diagram –

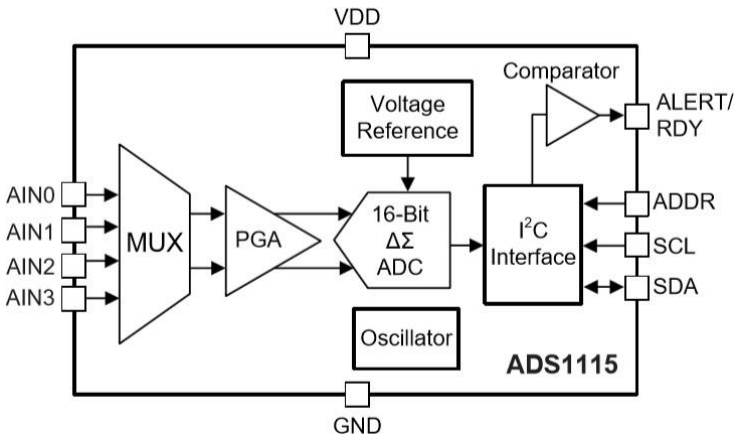

Figure 3.7: ADS1115 Block Diagram

Pinout –

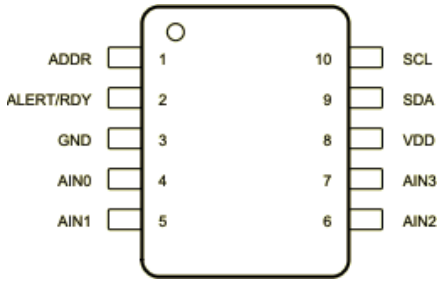

Figure 3.8: ADS1115 Pinout

Pin Description –

Pin Name	Type	Description
ADDR	Digital Input	I2C slave address select
AIN0	Analog Input	Analog input 0
AIN1	Analog Input	Analog input 1
AIN2	Analog Input	Analog input 2
AIN3	Analog Input	Analog input 3
ALERT / RDY	Digital Input	Comparator output or conversion ready
GND	Analog	Ground
NC	-	Not connected
SCL	Digital Input	Serial clock input - locks data on SDA
SDA	Digital I/O	Serial data - Transmits and receives data
VDD	Analog	Power Supply

Table 3.5: ADS1115 Pin Description

Assembly –

All surface-mount components are pre-soldered on the board. The included header-strip should be soldered on for breadboard use. The steps are as follows:

- **Prepare the header strip:** To hold the breadboard for soldering, place the supplied header strip long-pins-down into the breadboard.

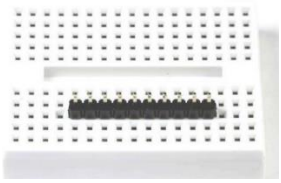

Figure 3.9: ADS1115 Assembly – Header Strip

- **Place the board:** Position breakout board on the pins of the header.

Figure 3.10: ADS1115 Assembly – Board

- **Solder:** For a good electrical connection, carefully solder each pin.

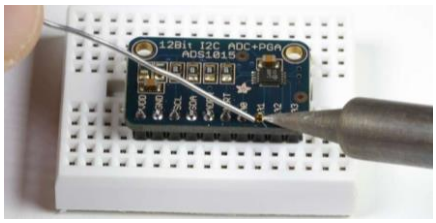

Figure 3.11: ADS1115 Assembly – Solder

I2C Addressing –

To set up the ADS1115, there are four addresses, so a maximum of four ADS1115 chips can be placed on one I2C bus, these addresses are 0x48, 0x49, 0x4a & 0x4b. Normally to switch between 4 addresses, there is a need for two inputs, but ADS1115 16-bit ADC has only one input pin known as address control pin. When GND, VDD, SDA, or SCL are connected to the single

address input, addresses from 0x48, 0x49, 0x4a, and 0x4b are set, respectively.

The below diagram shows one board addressed to 0x48:

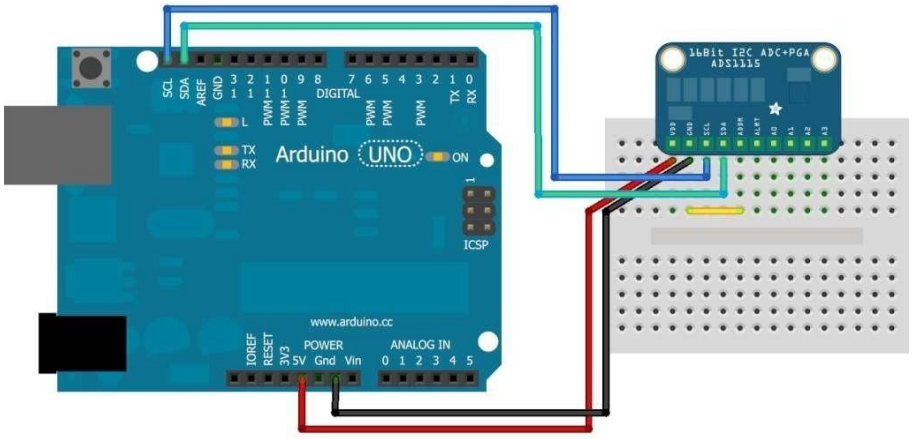

Figure 3.12: Connections for one ADS1115

Now, let's also see how all 4 addresses are used to connect 4 boards using a single I2C bus:

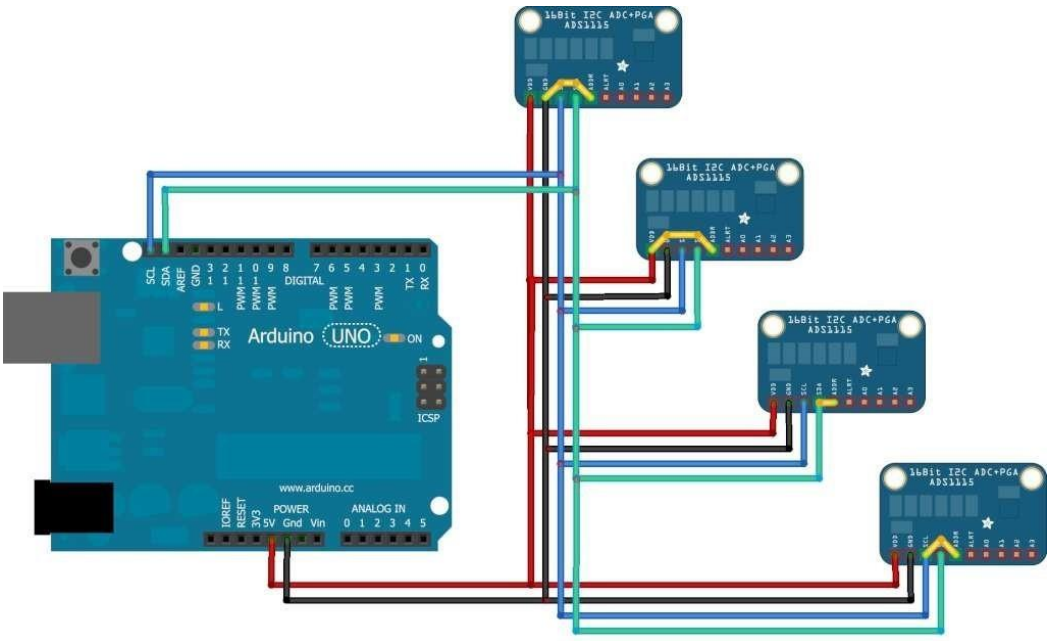

Figure 3.13: Connections for four ADS1115

Register Map –

The Address Pointer register on the ADS1115 provides access to four registers via the I2C interface. They are as follows:

- **Conversion Register** - This register holds the result of the most recent conversion.
- **Config Register** - This register is used to change the device's operating modes and query its status.
- The other two are **Lo thresh** and **Hi thresh**, which set the comparator function's threshold values..

7	6	5	4	3	2	1	0
0	0	0	0	0	0	P[1:0]	
W-0h	W-0h	W-0h	W-0h	W-0h	W-0h	W-0h	

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -n = value after reset

Figure 3.14: ADS1115 – Address Pointer Register

Bit	Field	Type	Reset	Description
7:2	Reserved	W	0h	Always write 0h
1:0	P[1:0]	W	0h	**Register address pointer** 00 : Conversion register 01 : Config register 10 : Lo_thresh register 11 : Hi_thresh register

Table 3.6: ADS1115 – Address Pointer Register Field

Let's know more about config register –

15	14	13	12	11	10	9	8
OS	MUX[2:0]			PGA[2:0]			MODE
R/W-1h	R/W-0h			R/W-2h			R/W-1h
7	6	5	4	3	2	1	0
DR[2:0]			COMP_MODE	COMP_POL	COMP_LAT	COMP_QUE[1:0]	
R/W-4h			R/W-0h	R/W-0h	R/W-0h	R/W-3h	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

Figure 3.15: ADS1115 – Config Register

Bit	Field	Type	Reset	Description
15	OS	R/W	1h	**Operational status or single-shot conversion start** This bit determines the operational status of the device. OS can only be written when in power-down state and has no effect when a conversion is ongoing. When writing: 0 : No effect 1 : Start a single conversion (when in power-down state) When reading: 0 : Device is currently performing a conversion 1 : Device is not currently performing a conversion
14:12	MUX[2:0]	R/W	0h	**Input multiplexer configuration (ADS1115 only)** These bits configure the input multiplexer. These bits serve no function on the ADS1113 and ADS1114. 000 : AIN_P = AIN0 and AIN_N = AIN1 (default) 001 : AIN_P = AIN0 and AIN_N = AIN3 010 : AIN_P = AIN1 and AIN_N = AIN3 011 : AIN_P = AIN2 and AIN_N = AIN3 100 : AIN_P = AIN0 and AIN_N = GND 101 : AIN_P = AIN1 and AIN_N = GND 110 : AIN_P = AIN2 and AIN_N = GND 111 : AIN_P = AIN3 and AIN_N = GND

11:9	PGA[2:0]	R/W	2h	**Programmable gain amplifier configuration** These bits set the FSR of the programmable gain amplifier. These bits serve no function on the ADS1113. 000 : FSR = ±6.144 V(1) 001 : FSR = ±4.096 V(1) 010 : FSR = ±2.048 V (default) 011 : FSR = ±1.024 V 100 : FSR = ±0.512 V 101 : FSR = ±0.256 V 110 : FSR = ±0.256 V 111 : FSR = ±0.256 V
8	MODE	R/W	1h	**Device operating mode** This bit controls the operating mode. 0 : Continuous-conversion mode 1 : Single-shot mode or power-down state (default)
7:5	DR[2:0]	R/W	4h	**Data rate** These bits control the data rate setting. 000 : 8 SPS 001 : 16 SPS 010 : 32 SPS 011 : 64 SPS 100 : 128 SPS (default) 101 : 250 SPS 110 : 475 SPS 111 : 860 SPS
4	COMP_MODE	R/W	0h	**Comparator mode (ADS1114 and ADS1115 only)** This bit configures the comparator operating mode. This bit serves no function on the ADS1113. 0 : Traditional comparator (default) 1 : Window comparator
3	COMP_POL	R/W	0h	**Comparator polarity (ADS1114 and ADS1115 only)** This bit controls the polarity of the ALERT/RDY pin. This bit serves no function on the ADS1113. 0 : Active low (default) 1 : Active high
2	COMP_LAT	R/W	0h	**Latching comparator (ADS1114 and ADS1115 only)** This bit controls whether the ALERT/RDY pin latches after being asserted or clears after conversions are within the margin of the upper and lower threshold values. This bit serves no function on the ADS1113. 0 : Nonlatching comparator . The ALERT/RDY pin does not latch when asserted (default). 1 : Latching comparator. The asserted ALERT/RDY pin remains latched until conversion data are read by the master or an appropriate SMBus alert response is sent by the master. The device responds with its address, and it is the lowest address currently asserting the ALERT/RDY bus line.
1:0	COMP_QUE[1:0]	R/W	3h	**Comparator queue and disable (ADS1114 and ADS1115 only)** These bits perform two functions. When set to 11, the comparator is disabled and the ALERT/RDY pin is set to a high-impedance state. When set to any other value, the ALERT/RDY pin and the comparator function are enabled, and the set value determines the number of successive conversions exceeding the upper or lower threshold required before asserting the ALERT/RDY pin. These bits serve no function on the ADS1113. 00 : Assert after one conversion 01 : Assert after two conversions 10 : Assert after four conversions 11 : Disable comparator and set ALERT/RDY pin to high-impedance (default)

Table 3.7: ADS1115 – Config Register Field Description

Knowing the above tables will be useful for low-level programming which is discussed in the following chapter.

3.1.5. 20 x 4 LCD Module

Overview –

Liquid Crystal Display, commonly known as LCD is a type of electronic display module used in a variety of circuits and devices such as mobile phones, calculators, computers, television sets, and so on. These displays are selected for multi-segment seven segments and light-emitting diodes. The

main benefits of using this module are that it is simple to program, and is inexpensive.

Figure 3.16: Liquid Crystal Display (LCD)

Features –

- Has an interface IC, such as the HD44780, that receives commands and data from the microcontroller and processes it to display information on the LCD screen.
- The operating voltage is between 4.7 and 5.3 volts.
- It has four rows, each of which can generate 20 characters.
- Each character can be constructed in a single 5x58 pixel box.
- The display can operate in two modes: 4-bit and 8-bit.

Pin Diagram –

Pin Name	**Pin Number**	**Description**
VSS (GND)	1	Connected to GND terminal of microcontroller unit or power source
VDD (+5V)	2	Voltage supply pin used to connect to the supply pin of power source to power the LCD
VEE (Contrast)	3	Determines the level of contrast, and is usually connected to a variable POT that can supply 0 to 5V. For maximum contrast, connect to GND

Register Select (RS)	4	Pin connects to microcontroller and switches between command and data register. It obtains either 0 i.e., data mode or 1 i.e., command mode.
Read / Write	5	Used to read or write data, which is decided by the value obtained from microcontroller unit pin. (1 refers to Read Operation, and 0 to Write Operation)
Enable (E)	6	Must be high in order to execute the Read/Write process.
Data Pins	7 – 14	To read data from or write data to the display
LED positive	15	Connected to +5V
LED negative	16	Connected to the GND

Table 3.8: LCD Pin Description

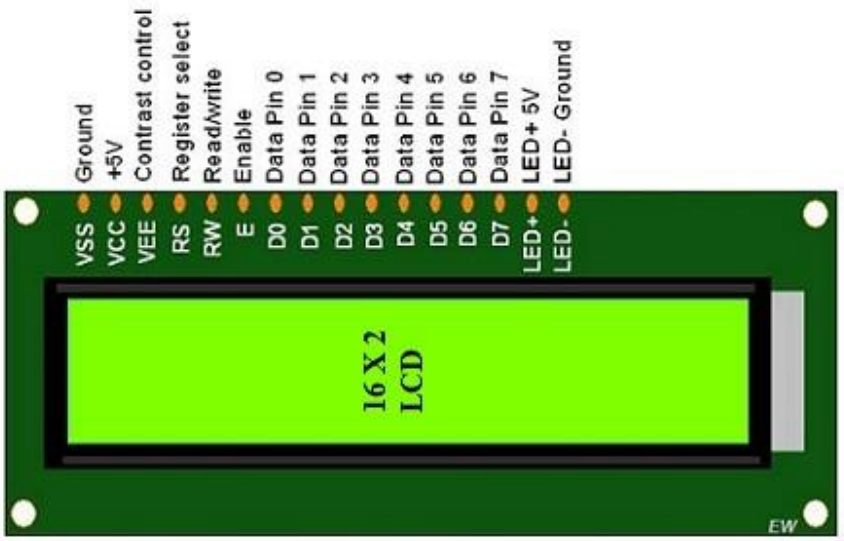

Figure 3.17: LCD Pins

Registers of LCD –

A 20×4 LCD has two registers, which is fixed based on RS pin, they are –

- **Command Register** - The main function is to store the instructions of command which are given to the display. In order to let predefined tasks like clearing the display, initialising, setting the cursor position, and controlling the display to be completed. Here commands processing can occur within the register.
- **Data Register** - The data register's primary function is to store the information that will be displayed on the LCD screen. The ASCII value of the character is the information that will be displayed on the LCD screen in this case. When information is sent to the LCD, it transmits to the data register, and the process begins there.

3.1.6. Resistors

A resistor is a passive two-terminal element that creates a certain level of resistance into the electrical circuit. The primary role of a resistor in any given electrical circuit is to reduce current flow, adjust signal levels, divide voltages, bias active elements, and terminate transmission lines. The fundament electrical law, known as Ohms Law, states that the voltage across a resistor is proportional to the current flowing through it, where R remains as the constant of proportionality.

$$R = \frac{V}{I}$$

where, V refers to voltage, I to current, and R to resistance.

Figure 3.18: Resistor

3.1.7. Jumper wires

Jumper wires are just wires with connector pins on both ends that can be used to connect two points without the need for soldering. Jumper wires are

commonly used with breadboards and other prototyping tools to allow for easy circuit changes.

Figure 3.19: Jumper Wires

There are three types of jumper wires:

- male – to – male,
- male – to – female, and
- female – to – female.

The endpoint of the wire is what distinguishes each of them. Female ends do not have a pin protruding from them and are used to plug things into it, whereas male ends do.

Figure 3.20: Different Types of Jumper Wires

3.1.8. Breadboard

A breadboard is a solderless device that is used to prototype circuit designs and test them. The terminals of most electronic components in electronic circuits can be placed into holes, and then wires can be used to connect them according to the requirements. The breadboard is made up of metal strips that run underneath the board and connect to holes on top. In the picture below, the top and bottom rows of the holes are linked horizontally and split in the centre, while the rest of the holes are linked vertically.

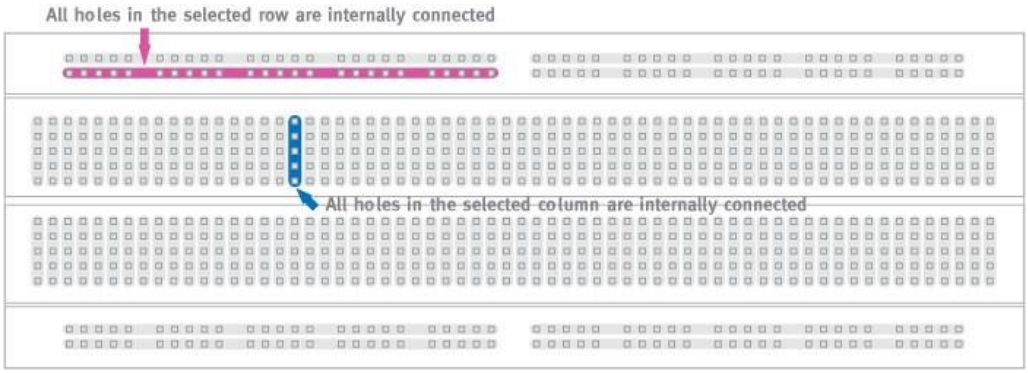

Figure 3.21: Breadboard

3.2. BLOCK DIAGRAM

The purpose of a Transmitter here is Data Logging while the purpose of the Receiver is Displaying the data.

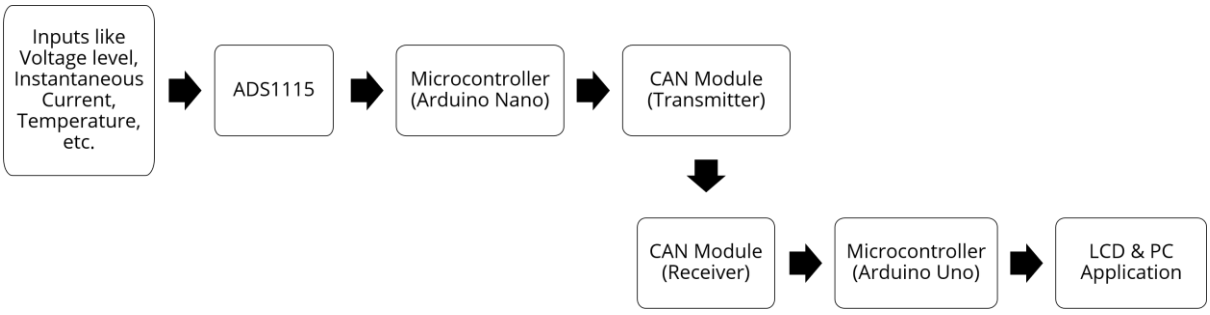

Figure 3.22: Block Diagram

The project works the following way -

1. Using sensors like voltage sensor, current sensor, pulse sensor we will get voltage, current and distance travelled values respectively

2. We read this data using ADS1115
3. Then, to process the information, we have a microcontroller
4. After that we set up the communication using CAN Module, so we transmit this data. Since a vehicle has a lot of nodes from which it transmits data, we usually have more than two nodes, but for the sake of simplicity let's assume we have only one transmitter and one receiver.
5. Again, we have a microcontroller that processes the data to understand which parameter's value is being read.
6. Then, we display these parameters' values using LCD and a PC Application for validation purposes.

3.3.CIRCUIT DIAGRAM

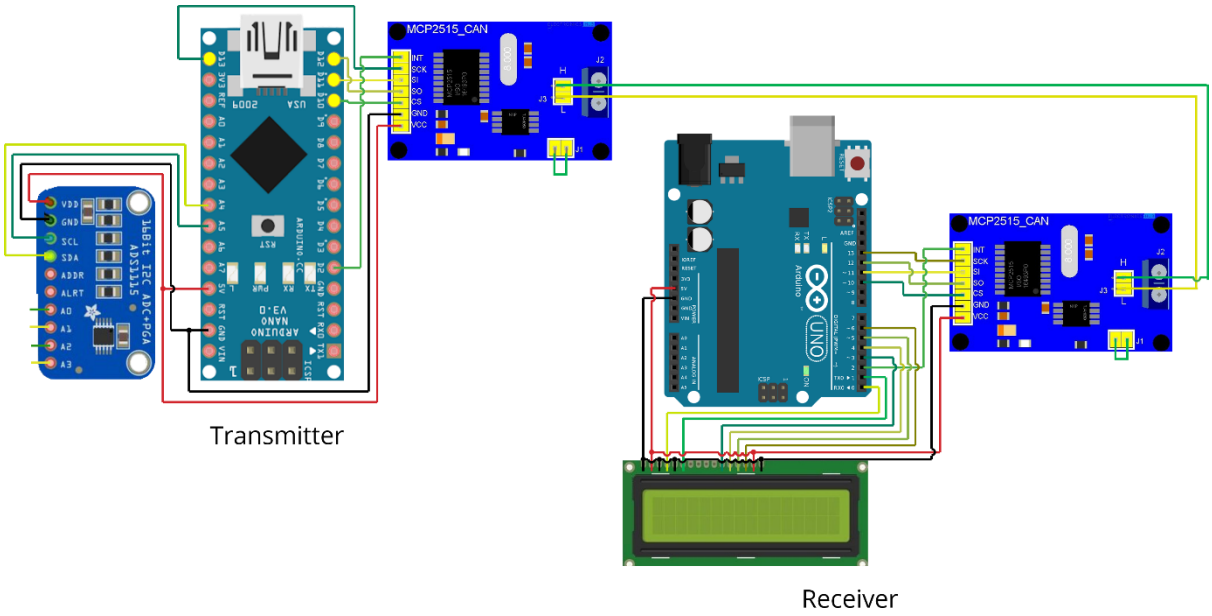

Figure 3.23: Circuit Diagram

The above circuit can be used to read 8 parameters, but using I2C Addressing of ADS1115 (explained in the components section), we can read up to 32 by using 8 ADS1115 instead of 2.

For checking whether the communication is set up or not, we can create a a voltage divider, to end terminals connect a positive and negative terminal of a battery and to the output of voltage divider connect AIN1 of ADS1115. After finding voltage from

AIN1 divide it by resistance to get current. Then, for voltage, connect a battery to it. In this way we can check the working of voltage and instantaneous current. The real project looks like -

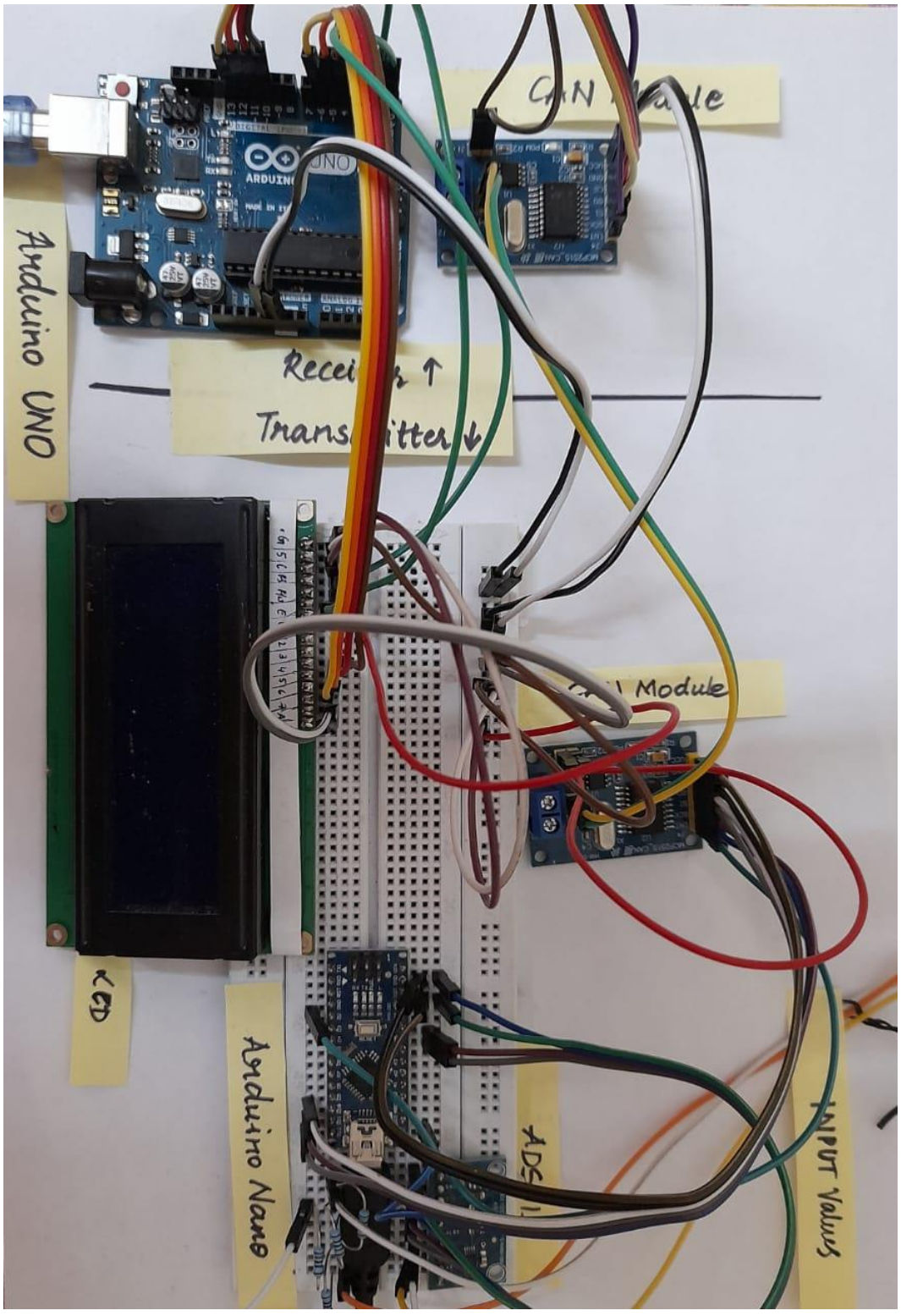

Figure 3.24: Real Circuit

4. CHAPTER IV – SOFTWARE IMPLEMENTATION

4.1. ARDUINO IDE

4.1.1. Introduction

Open-source software for writing and compiling code in various Arduino modules like Arduino Uno, Arduino Mega, Arduino Leonardo, and many more is the Arduino Integrated development environment (IDE). These modules have an onboard microcontroller that can be written to and accepts the information as a code.

Arduino IDE is a cross-platform application, i.e., it is available for MAC, Windows, Linux, and runs on the Java Platform and is equipped with integrated functions and commands that play a crucial role in debugging, editing, and compiling the code in the surroundings. Both C and C++ languages are supported in this environment. IDE mainly contains two parts –

- Editor - Used for writing the code,
- Compiler - Used in the Arduino Module for the compilation and upload.

In the end, IDE's main code, also called a sketch, creates a Hex file that is transferred and uploaded to a board microcontroller.

4.1.2. How to Download

On Arduino.cc website, go to the Software tab to download Arduino IDE based on your operating system. Before downloading check the technical requirements to understand whether the IDE is compatible with your operating system. For instance, the Windows version requires Windows 8.1 or later, as the application is not compatible with Windows 7 or earlier versions.

4.1.3. Sections

The IDE environment has the following sections –

- **Menu Bar -** It contains File, Edit, Sketch, Tools & Help, all of them have drop-down lists. File, Edit & Tools have the general options, so

let's discuss few options available in Sketch and Tools.

- Upload using Programmer - This is used to override the onboard Bootloader. This allows us to make full use of the Flash memory. To do so, use the Tools-> Burn Bootloader option to recover the Bootloader and upload it to the USB serial port.
- Include Library - This contains a collection of Arduino libraries. Libraries are added at the beginning of the code with the '# include' command. The Manage Libraries option, which is also present in the Tools drop-down list, can be used to download external libraries.
- Serial Monitor - It permits data to be exchanged with a board connected to the port. When it starts up, it refreshes the operating system. It can also be accessed through the Toolbar.
- Board - This is used to pick the board we connected from the list of available boards.
- Processor – It is displayed based on the selected board. Every time we pick the board, it refreshes.
- Port - This field contains a list of all virtual and physical serial devices on our machine.
- Burn Bootloader - There is a bootloader on the board. If we buy a microcontroller without a bootloader, this option comes in handy.

- **Toolbar Button - Verify**, **Upload**, **New**, **Open**, and **Save** are the icons on the toolbar. It also includes a Serial Monitor button in the right corner, which activates the Serial Monitor.
- **Text Editor -** Here the codes goes, and
- **Output Pane -** When we click on verify button, the output pane shows the uploading status and the error messages. After we correct the errors, if any, we can upload the code to the board.

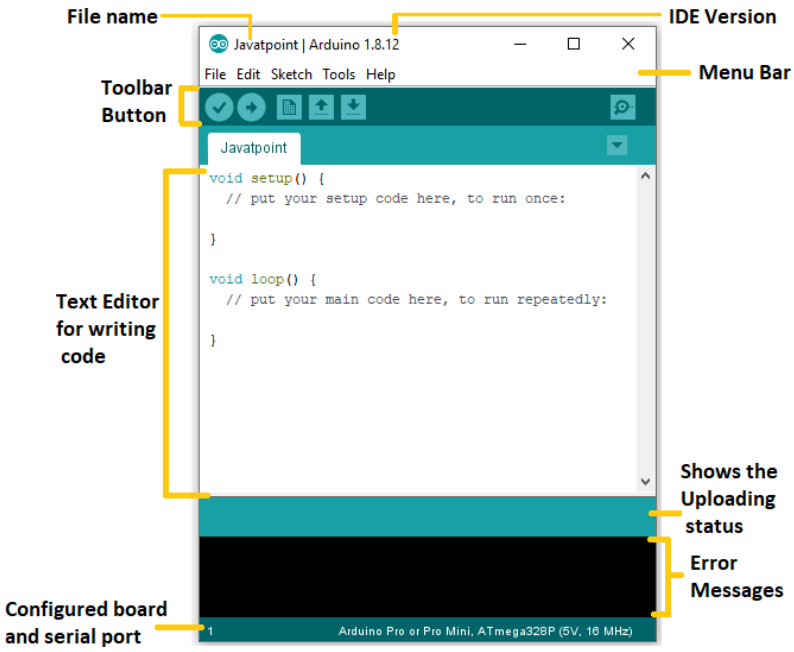

Figure 4.1: Arduino IDE Sections

4.1.4. Example

The steps are –

- Install Arduino IDE on the computer.
- Using a USB cable, connect the board to the laptop
- Then open the IDE.
- Let’s say you're using an Arduino UNO board, go to Tools>Boards>Arduino UNO to choose the board.
- After that, choose the port by going to Tools>Port.
- Now use the below example code which will blink the built-in LED, which is available at the 13th pin as discussed in chapter 3.
- Click on the ‘verify’ button, then on the 'upload' button.
- Once it gets uploaded, you will see the Arduino’s built-in LED blink.

```
#define LEDPin 13

void setup() {
  // put your setup code here, to run once:
  pinMode(LEDPin, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LEDPin, HIGH);
  delay(1000);
  digitalWrite(LEDPin, LOW);
  delay(1000);
}
```

Figure 4.2: LED Blink Code

This takes 924 bytes of memory. Let's try looking at whether this same application can be coded such that it uses less memory.

While we are talking about Arduino IDE, let's also look at a more efficient way to code when compared to using pre-determined libraries, i.e., coding by using registers.

4.1.5. Low-level Programming

Overview –

A low-level programming language offers little or no abstraction from a computer's instruction set architecture—commands or functions in a language map that is structurally equivalent to the processor's instructions. It's commonly referred to as either machine code or assembly language. Without the need for a compiler or interpreter, low-level languages can be converted to machine code and run directly on the processor.

Features –

When compared to high-level languages, low-level programming has the following features –

- Relatively non-portable, because it is optimized for specific system architecture.
- Execution time is less

- Has a small memory footprint
- Simple, but is considered difficult to handle due to numerous details that a programmer has to consider.

For low-level programming, we need to understand the registers available in a particular component, which is available in the datasheet of every component.

Now, let's see the proof of how low-level programming uses less execution time and takes less memory space.

For this here is a code written using registers for the same application as above. The code is as follows -

```
void setup(){
  // Using DDRB to set 13th pin as output
  DDRB = B00100000;
}

void loop(){
  PORTB = B00100000; // Turn on LED
  delay(1000);
  PORTB = B00000000; // Turn off LED
  delay(1000);
}
```

Figure 4.3: LED Blink Code using registers

To get more understanding, read the registers of Arduino UNO.

Writing code by low-level programming occupies 646 bytes of memory. And, since we are directly assigning to ports, microcontrollers don't need work on understanding the syntaxes like digitalWrite or pinMode. Hence, it uses lesser space & reduces execution time.

4.2. XOJO PLATFORM

4.2.1. Introduction

Xojo is an integrated development environment (IDE), used to design and build other software applications. It uses a programming language which is also known as Xojo. In simple, you as the programmer or developer enter your Xojo code into the Xojo IDE, which then compiles the code into a native app

that can be run, independent of Xojo, on a computer. It builds high-quality, native apps for the Desktop (macOS, Windows, Linux), iOS (iPad/iPhone), Web, and Raspberry Pi.

4.2.2. Features

- Native Controls - There are over 40 user interface controls built-in, so it's easy to make an application look good
- Beginner-Friendly - It is a simple platform to use
- Drag & drop UI
- Advanced Features - It supports Object Oriented concepts like inheritance, interfaces, polymorphism and has many more advanced features
- Easy Deployment – using Xojo Cloud which is a simple, safe, and maintenance-free web application hosting.

4.2.3. Basic Steps

- Download - Create an account in Xojo & download from xojo.com.
- Develop - Build application's front end using drag-and-drop, after that give functionality using Xojo language.
- Launch - Once the application is built, compile and then launch to test.

4.2.4. Getting Around

1. Begin by opening the Xojo application. Xojo launches with a Getting Started window.
2. Click on 'start using Xojo', it will direct to the Project Chooser window.

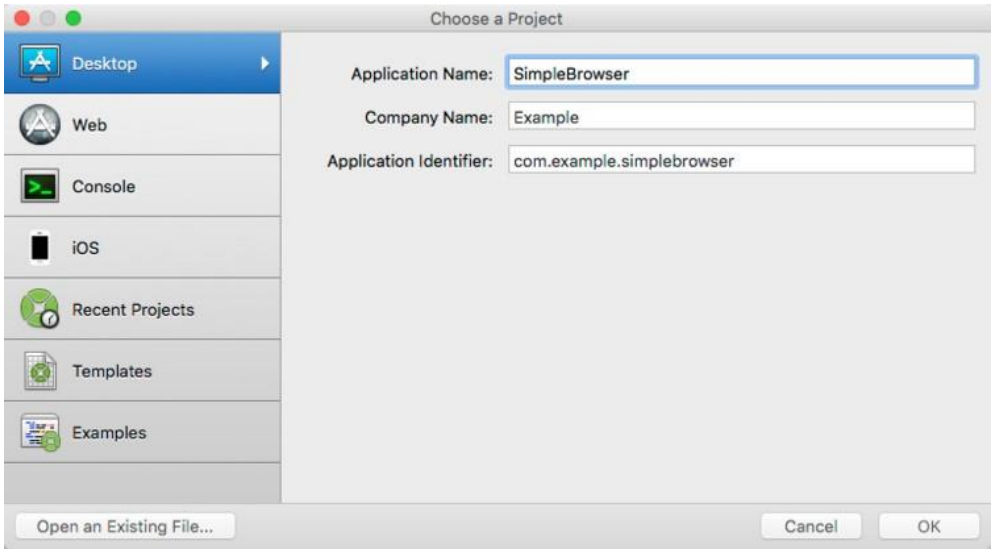

Figure 4.4: Xojo – Project Chooser Window

3. This will prompt you to choose a template for a new project. Let's build a 'Desktop' app, so for that select “Desktop”.
4. Give an application name which is the name of your app. Then give a company name, which can be left blank.
5. Application Identifier is a unique identifier for this application which will automatically populate based on Application and Company Names, but you can also change it to something else.
6. Press the OK button.

4.2.5. IDE Workspace

Workspace is the main window; this is the place where you can design your application.

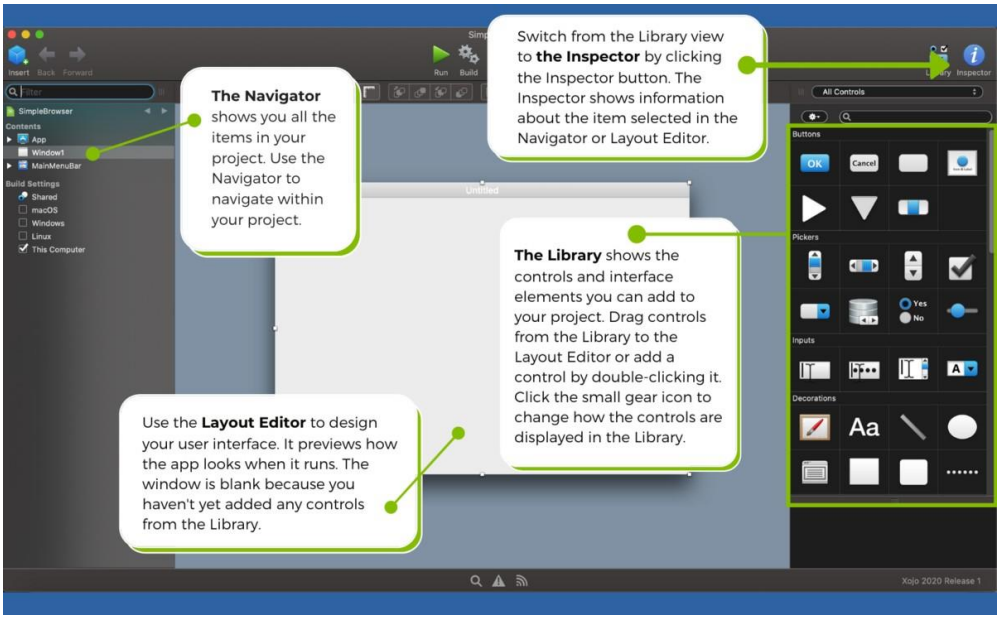

Figure 4.5: Xojo – Main window

4.2.6. Example

Let's make an addition operation application, where there is two input fields in which the user can enter some values and a button which on click adds the two numbers given, after that it prints on some other text field. To implement the above application, follow the steps below –

1. Create a new file and give the Application name as 'Addition'. It opens a default empty project. Go to file as Save the project.
2. Go to the Inspector tab at the right-side corner. In the Frame section change title to 'Addition'. The widow will look like -

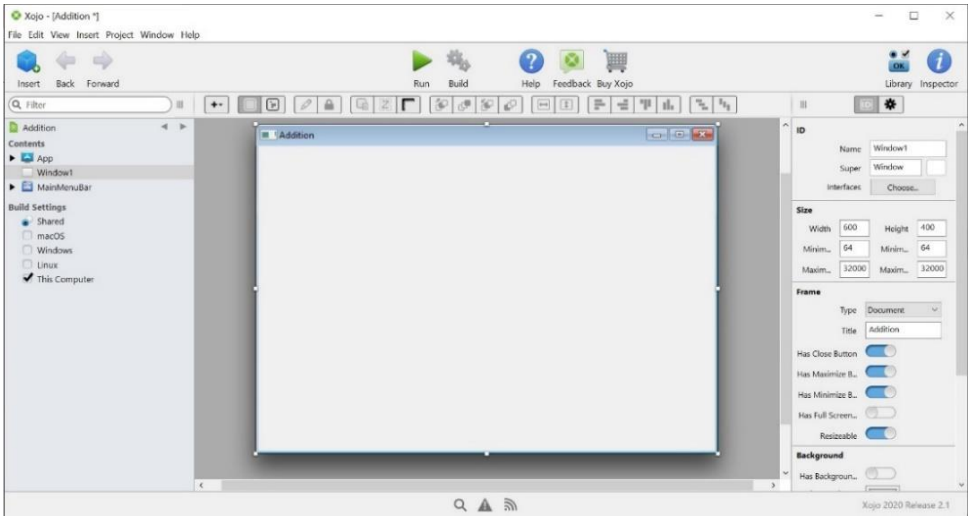

Figure 4.6: Xojo Window

3. Now go to the Library tab, from the 'Inputs' section drag and drop two text fields. Now, click on one text field go to inspector tap, change Name from TextField1 to txtNumber1. Similarly, from the other text field name it as txtNumber2.

4. After that, from the Library tab drag-and-drop a button, from the 'button' section, and a Label, from the 'Decorations' section. By going to the Inspector tab name them - btnAdd and lblResult respectively.

5. Now double-click on the button in the Layout Editor, event handler window opens. Click on 'Action' the Ok.

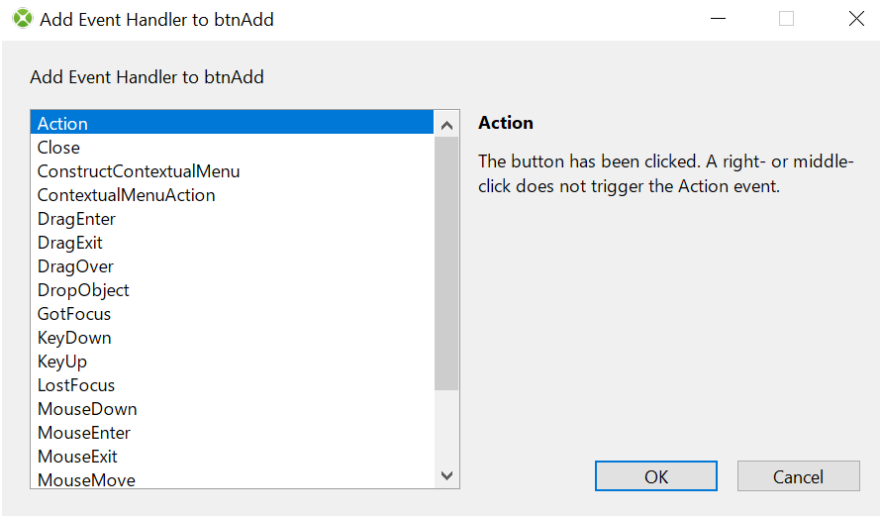

Figure 4.7: Xojo – Event Handler

6. Write the following code in the window which gets open –

```
Var Result As Integer = 0
Result = (txtNumber1.Text.ToInteger + txtNumber2.Text.ToInteger)
lblResult.Text = str(Result)
```

7. Now, similarly double-click on Label and click on 'Open' event, then ok. Write the following code there –

```
Var initial As Integer = 0
lblResult.Text = str(initial)
```

8. Click on the Run symbol to test. The application opens which will look something like the picture below –

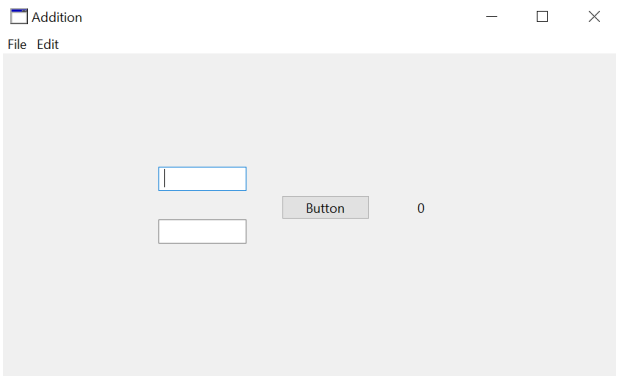

Figure 4.8: Xojo Addition Operation Application

You can try typing two numbers and then click on the button, as soon as you click, you will see the result. Hence, our simple application is ready.

9. To Deploy this application, in the Navigator section click on all the platforms in which you want to build the application. And then click on the Build button in the toolbar (or choose Project ↠ Build Application from the menu). For each selected platform, Xojo creates a standalone application.

Using SerialConnection control of the Xojo platform, which is available in the library, we can perform serial communications with serial devices. And this is how we interface Xojo with Arduino.

5. CHAPTER V – FLOWCHART AND CODE

5.1. TRANSMITTER SIDE

5.1.1. Flowchart

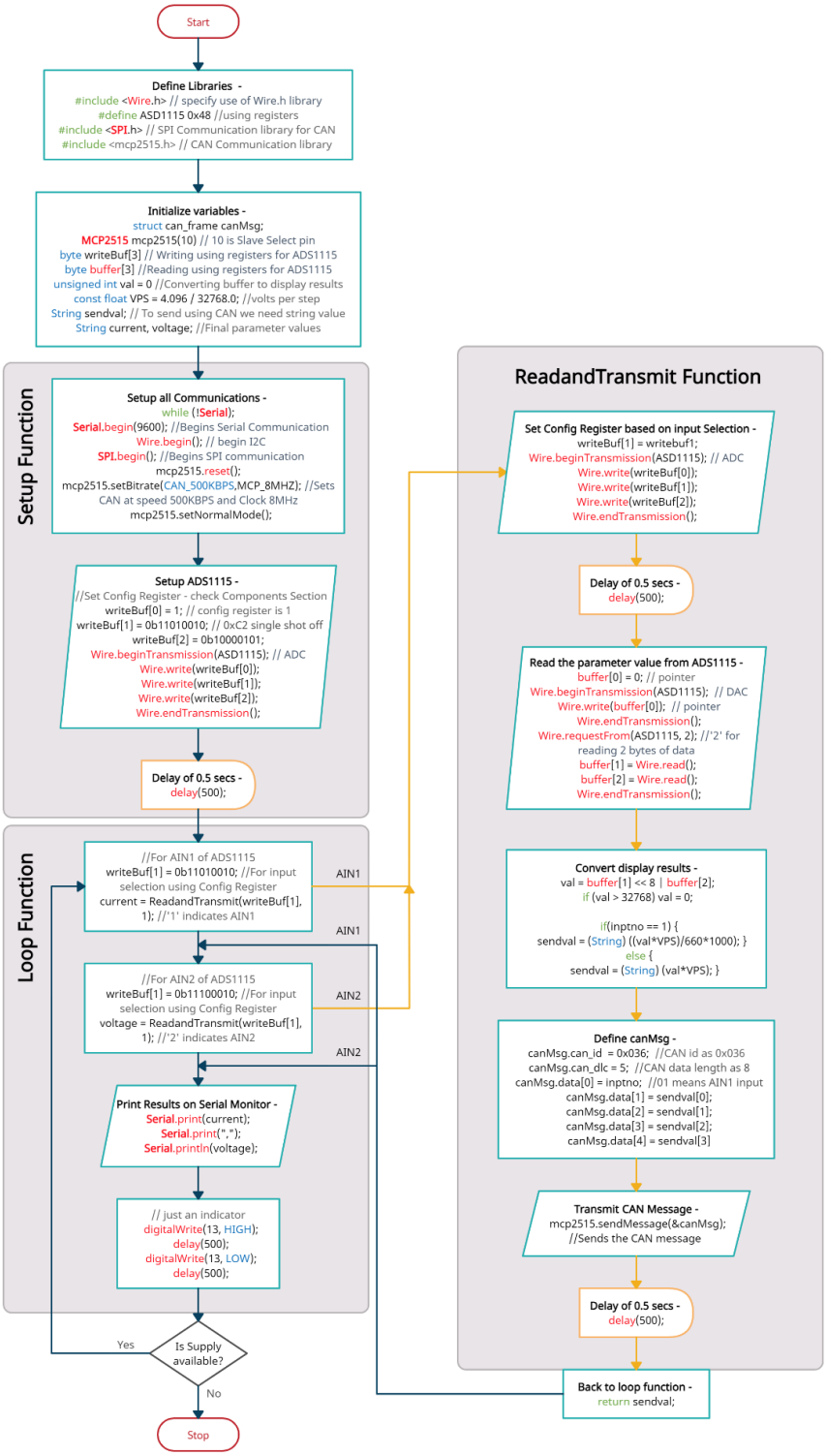

Figure 5.1: Transmitter Flowchart

In Loop Function, for every input i.e., AIN1, AIN2, etc., it has to follow all the steps of the ReadandTransmit Function. It is shown in the above flowchart for 2 inputs, for AIN1 it is following all ReadandTransmit Function steps, and as soon as returns, again for AIN2 it follows all ReadandTransmit Function steps. Using one ADS1115, like this four inputs can also be read.

5.1.2. Code

```
#include <Wire.h> // specify use of Wire.h library
#define ASD1115 0x48
#include <SPI.h>          // Library for using SPI Communication
#include <mcp2515.h>      // Library for using CAN Communication

struct can_frame canMsg;
MCP2515 mcp2515(10);
String sendval;
unsigned int val = 0;
byte writeBuf[3];
byte buffer[3];
const float VPS = 4.096 / 32768.0; // volts per step
String current;
String voltage;

void setup() {
  while (!Serial);
  Serial.begin(9600);

  Wire.begin(); // begin I2C
  SPI.begin(); // Begins SPI communication

  // ASD1115

  writeBuf[0] = 1; // config register is 1
  writeBuf[1] = 0b11010010; // 0xC2 single shot off
  // bit 15 flag bit for single shot
  // Bits 14-12 input selection:
  // 100 refers to ANC0; 101 to ANC1; 110 to ANC2; 111 to ANC3
  // Bits 11-9 Amp gain. Default to 010 here 001 P19
  // Bit 8 Operational mode of the ADS1115.
  // 0 : Continuous conversion mode
  // 1 : Power-down single-shot mode (default)

  writeBuf[2] = 0b10000101; // bits 7-0  0x85
  // Bits 7-5 data rate default to 100 for 128 Samples Per Second (SPS)
  // Bits 4-0  comparator functions check components section .

  // setup ADS1115
  Wire.beginTransmission(ASD1115);  // ADC
  Wire.write(writeBuf[0]);
  Wire.write(writeBuf[1]);
  Wire.write(writeBuf[2]);
  Wire.endTransmission();

  delay(500);
```

```
  //setup MCP2515
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ);
  //Sets CAN at speed 500KBPS and Clock at 8MHz
  mcp2515.setNormalMode();
}

void loop() {
  //For AIN1
  writeBuf[1] = 0b11010010;
  current = ReadandTransmit(writeBuf[1], 1);
  //For AIN2
  writeBuf[1] = 0b11100010;
  voltage = ReadandTransmit(writeBuf[1], 2);

  Serial.print(current);
  Serial.print(",");
  Serial.println(voltage);

  // just an indicator
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
  delay(500);
}

String ReadandTransmit(byte writebuf1, int inptno) {
  writeBuf[1] = writebuf1;
  Wire.beginTransmission(ASD1115); // ADC
  Wire.write(writeBuf[0]);
  Wire.write(writeBuf[1]);
  Wire.write(writeBuf[2]);
  Wire.endTransmission();
  delay(500);

  buffer[0] = 0; // pointer
  Wire.beginTransmission(ASD1115); // DAC
  Wire.write(buffer[0]); // pointer
  Wire.endTransmission();
  Wire.requestFrom(ASD1115, 2);

  buffer[1] = Wire.read();
  buffer[2] = Wire.read();
  Wire.endTransmission();

  // convert display results
  val = buffer[1] << 8 | buffer[2];
  if (val > 32768) val = 0;
  //Serial.println(val * VPS);
  if(inptno == 1){
    sendval = (String) ((val*VPS)/660*1000);
  }


  canMsg.can_id  = 0x036; //CAN id as 0x036
  canMsg.can_dlc = 5; //CAN data length as 5
  canMsg.data[0] = inptno; //00 means AIN0 input
  canMsg.data[1] = sendval[0];
  canMsg.data[2] = sendval[1];
  canMsg.data[3] = sendval[2];
  canMsg.data[4] = sendval[3];
```

```
  mcp2515.sendMessage(&canMsg); //Sends the CAN message
  delay(1000);
  return sendval;
}
```

Figure 5.2: Transmitter Code

5.2. RECEIVER SIDE

5.2.1. Flowchart

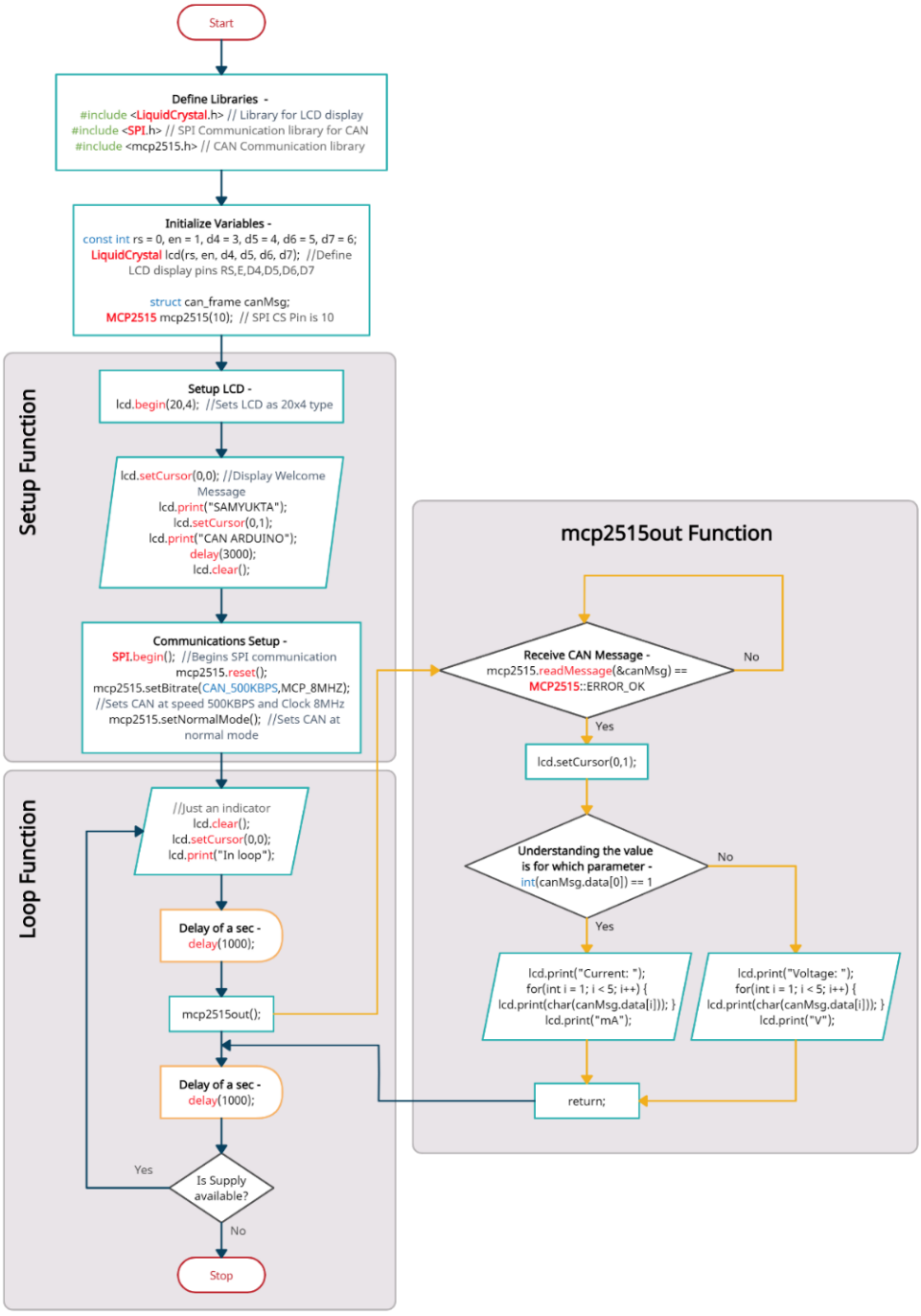

Figure 5.3: Receiver Flowchart

5.2.2. Code

```
#include <SPI.h>              //Library for using SPI Communication
#include <mcp2515.h>          //Library for using CAN Communication
#include <LiquidCrystal.h>    //Library for using LCD display

const int rs = 0, en = 1, d4 = 3, d5 = 4, d6 = 5, d7 = 6;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);  //Define LCD display pins RS,E,D4,D5,D6,D7

struct can_frame canMsg;
MCP2515 mcp2515(10);                // SPI CS Pin 10

void setup(){
  lcd.begin(20,4);                  //Sets LCD as 20x4 type
  lcd.setCursor(0,0);               //Display Welcome Message
  lcd.print("SAMYUKTA");
  lcd.setCursor(0,1);
  lcd.print("CAN ARDUINO");
  delay(3000);
  lcd.clear();

  SPI.begin();                      //Begins SPI communication

  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS,MCP_8MHZ); //Sets CAN at speed 500KBPS and Clock at 8MHz
  mcp2515.setNormalMode();                  //Sets CAN at normal mode
}

void mcp2515out(){
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) // To receive data (Poll Read)
  {
     lcd.setCursor(0,1);
     if(int(canMsg.data[0]) == 1){
        lcd.print("AIN1: ");
        for(int i = 1; i < 5; i++){
            lcd.print(char(canMsg.data[i]));
         }
         lcd.print("mA");
     }
     else{
        lcd.print("AIN2: ");
        for(int i = 1; i < 5; i++){
            lcd.print(char(canMsg.data[i]));
         }
         lcd.print("V");
     }
  }

  return;
}

void loop() {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("In loop");
    delay(1000);
    mcp2515out();
    delay(1000);
}
```

Figure 5.4: Receiver Code

5.3. Xojo Coding – A Validation Tool

Steps for creating the application using Xojo Platform –

i. Open a new project, name it Display. Drag-and-drop the following controls from Library tool and place them as shown in below figure–

- Five Labels from Decorations section; three of them are fixed labels, i.e., they don't change based on real-time situations. Hence, change their name and text from Inspector pane to lblSerial, lblA, and lblV respectively. The rest two change the values based on the real-time situations, hence just change their name to lblVoltage and lblCurrent respectively;
- One pop-up menu from Pickers section;
- One button from Button section;
- Lastly, DeviceListUpdator and SerialConnection1 from Controllers section.

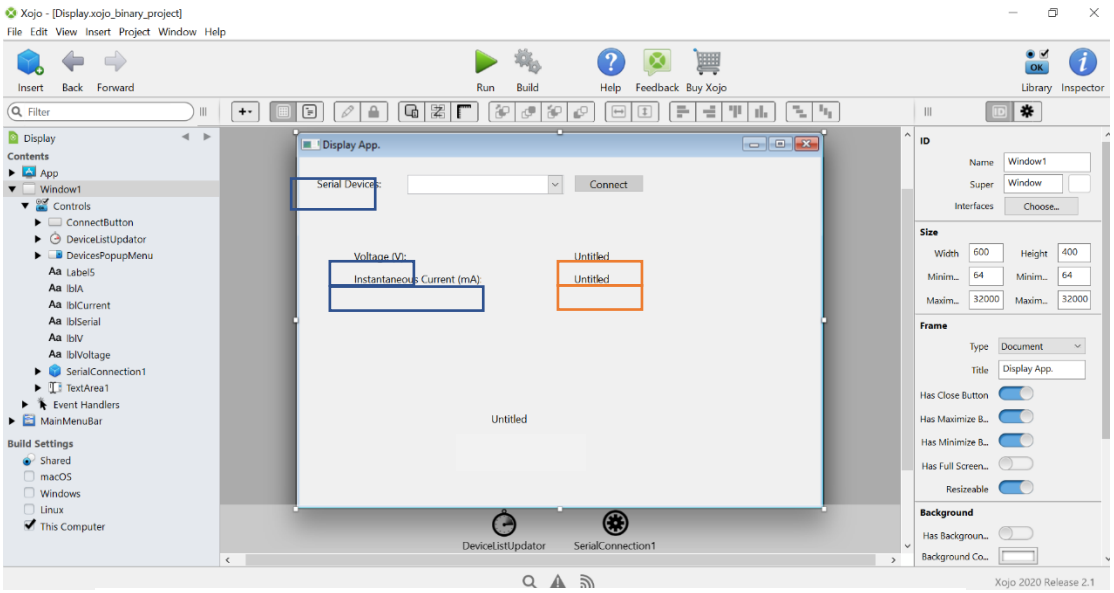

Figure 5.4: PC Application – Main Window

ii. Double-click on SerialConnection1, select DataReceived Event and write the following code –

```
DataReceived
  Var data As String
  data = Me.LookAhead(Encodings.ASCII)
  If data.IndexOf(EndOfLine.Windows) > -1 Then
    TextArea1.Value = TextArea1.Value + Me.ReadAll(Encodings.ASCII)
  End If
  if(lblVoltage.Text <> data.NthField("," , 2)) Then
    lblVoltage.Text = data.NthField("," , 2)
  End If
  if(lblCurrent.Text <> data.NthField("," , 1)) Then
    lblCurrent.Text = data.NthField("," , 1)
  End If…
```

Figure 5.4: PC Application – DataReceived Code

iii. Double-click on DeviceListUpdator, select Action event and write –

```
Action
    Var count As Integer = DevicesPopupMenu.RowCount

  If SerialDevice.Count <> count Then
    // The number of serial devices has changed so update the menu
    DevicesPopupMenu.RemoveAllRows
    For i As Integer = 0 To SerialDevice.LastIndex
      DevicesPopupMenu.AddRow(SerialDevice.At(i).Name)
    Next
    If SerialDevice.Count < count Then // a device has been removed
      DevicesPopupMenu.SelectedRowIndex = 0
    Else // one has been added so select the new device
      DevicesPopupMenu.SelectedRowIndex = DevicesPopupMenu.LastRowIndex
    End If
  End If
```

Figure 5.5: PC Application – DeviceListUpdator Code

iv. Double-click on Connect button, select Action event, then write –

```
Action
  If Me.Caption = "Disconnect" Then // Disconnect from the serial device
    SerialConnection1.Close
    Me.Caption = "Connect"
    DevicesPopupMenu.Enabled = True
    DeviceListUpdator.RunMode = Timer.RunModes.Multiple 'turn it on
  Else // Connect to the serial device
    // Set the serial device to the index of the one chosen in the popup menu
    SerialConnection1.Device = SerialDevice.At(DevicesPopupMenu.SelectedRowIndex)
    Try
      SerialConnection1.Connect
      DevicesPopupMenu.Enabled = False
      DeviceListUpdator.RunMode = Timer.RunModes.Off
      Me.Caption = "Disconnect"
    Catch error As IOException
      Beep
      MessageBox("The selected serial device could not be opened.")
    End Try
  End If
```

Figure 5.6: PC Application – Connect Button Code

v. Now, click on Window in the Navigator, then press “plus” symbol from tool bar, under the options which are available select Event Handler. Then, choose open event, after that write the following code –

```
Open
    var c as Integer
    c = 0
    lblVoltage.text = c.ToString
    lblCurrent.Text = c.ToString
```

Figure 5.7: PC Application – Open Event Code

vi. Now, click on Build. Select the port connected to Transmitter port from pop-up menu. Then, click on connect button.

vii. The Voltage and Current values change based on change in the input value being read by the ADS1115.

6. CHAPTER VI – OUTCOMES

6.1. CONCLUSION

The project is able to set up the in-vehicle communication, log the data precisely, and transmit and receive the information without errors.

Using the same connections, a PCB can be created and the same code can be uploaded. The below table shows the values read by transmitter, receiver and PC Application. As its seen all of them are the same

S. No.	Transmitter		Receiver		PC Application	
	Voltage (V)	Current (mA)	Voltage (V)	Current (mA)	Voltage (V)	Current (mA)
1	3.78	2.33	3.78	2.33	3.78	2.33
2	3.78	1.79	3.78	1.79	3.78	1.79
3	3.79	0.89	3.79	0.89	3.79	0.89
4	3.78	4.73	3.78	4.73	3.78	4.73

Table 6.1: Results

6.2. FUTURE SCOPE

A few enhancements which can be done to the project are listed below –

- The receiver code can be made specific to the system its being used, i.e., decide whether the information being received is useful for the present system of the vehicle.
- Control of sensors can be implemented with the use of data being distributed to all the systems
- A mobile application can be created which will display the parameter values. A few parameters which will come handy is GPS location, audio-video systems, etc.

BIBLIOGRAPHY

- https://www.codrey.com/embedded-systems/serial-communication-basics/
- https://www.researchgate.net/publication/341311712_Comparison_of_CAN_LIN_FLEX_RAY_and_MOST_In-vehicle_bus_protocols
- https://skill-lync.com/blogs/auto-communication-protocols
- https://www.ni.com/en-in/innovations/white-papers/06/flexray-automotive-communication-bus-overview.html
- https://www.enigmatos.com/2018/03/06/flexray-protocol-overview-blog/
- https://www.seeedstudio.com/blog/2019/11/27/introduction-to-can-bus-and-how-to-use-it-with-arduino/
- https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-uno.html
- https://components101.com/microcontrollers/arduino-uno
- https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-nano.html
- https://www.polytechnichub.com/applications-controller-area-network-can-bus/
- https://components101.com/microcontrollers/arduino-nano
- https://en.wikipedia.org/wiki/List_of_Arduino_boards_and_compatible_systems
- https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/
- https://circuitdigest.com/microcontroller-projects/arduino-can-tutorial-interfacing-mcp2515-can-bus-module-with-arduino
- https://protosupplies.com/product/mcp2515-can-bus-interface-module/
- https://www.sparkfun.com/datasheets/DevTools/Arduino/MCP2515.pdf
- https://www.best-microcontroller-projects.com/ads1115.html
- https://learn.adafruit.com/adafruit-4-channel-adc-breakouts/assembly-and-wiring
- https://www.ti.com/lit/ds/symlink/ads1115.pdf?ts=1623574042104&ref_url=https%3A%2F%2Fwww.ti.com%2Fproduct%2FADS1115
- https://components101.com/displays/16x2-lcd-pinout-datasheet
- https://www.elprocus.com/lcd-16x2-pin-configuration-and-its-working/
- https://en.wikipedia.org/wiki/Resistor
- http://blog.sparkfuneducation.com/what-is-jumper-wire
- http://wiring.org.co/learning/tutorials/breadboard/
- https://www.theengineeringprojects.com/2018/10/introduction-to-arduino-ide.html
- https://www.javatpoint.com/arduino-ide]
- https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
- https://en.wikipedia.org/wiki/Low-level_programming_language
- https://e-radionica.com/en/blog/blinking-a-led-using-assembler-in-arduino/
- https://docs.xojo.com/GettingStarted:Desktop_Tutorial