SEATBELT WARNING AND ALERT SYSTEM FOR PASSENGER CARS

MAY 2025

KLS Gogte Institute of Technology, Belagavi

Atharva Dharmatti (2GI21EC034)

Jyoti Karagi (2GI21EC057)

Kunal Sindhi (2GI21EC061)

Sonal Poojary (2GI21EC144)

Table of Contents

Abstract

In recent years, vehicle safety has become a top priority in the automotive industry, with a growing emphasis on integrating smart systems that ensure both driver and passenger protection. One of the most basic yet crucial safety measures is the use of seat belts. However, human error, negligence, and lack of awareness often result in non-compliance, especially when monitoring systems are either absent or easily bypassed. This project proposes the development of a Seat Belt Alert System utilizing Arduino Uno microcontrollers and CAN (Controller Area Network) modules to ensure seat belt compliance in a reliable and efficient manner. The system uses a force sensor to detect seat occupancy and a relay switch to simulate seat belt engagement. Two Arduino boards communicate using MCP2515 CAN modules, mimicking real-world automotive communication systems. A buzzer and an LCD display provide immediate audio-visual alerts if the seat is occupied but the seat belt is not fastened. This modular and scalable solution not only offers a cost-effective alternative to commercial systems but also serves as a foundational step toward smarter, safer vehicles for the future.

Introduction

As vehicle technology evolves, safety has emerged as a critical aspect of both vehicle design and user experience. Governments and manufacturers alike have recognized the importance of reducing fatalities and serious injuries due to road accidents. Among the many safety features introduced, the seat belt remains one of the most effective and affordable solutions. Despite its proven benefits, seat belt usage remains inconsistent, especially in developing countries where awareness, enforcement, and proper in-vehicle systems are lacking.

Modern automotive systems are increasingly embedded with electronics and smart features that monitor and respond to real-time conditions. The adoption of embedded systems, microcontrollers, and inter-device communication protocols like the CAN bus has revolutionized vehicle safety, allowing the implementation of dynamic features such as airbag deployment systems, anti-lock braking systems (ABS), lane-keeping assist, and driver attention monitoring.

This project contributes to that revolution by designing a Seat Belt Alert System using low-cost components, including Arduino microcontrollers, force sensors, relay switches, and CAN modules. The system is capable of detecting seat occupancy and verifying whether the seat belt has been fastened. If the seat is occupied and the seat belt is not engaged, a buzzer and LCD screen alert the driver or passengers immediately. This approach not only reinforces safety but also demonstrates the feasibility of integrating smart monitoring in low-cost vehicles.

Market Research

The global automotive safety market has grown significantly over the past decade, driven by consumer awareness, governmental regulations, and technological innovation. According to MarketsandMarkets, the automotive safety system market is projected to reach USD 180 billion by 2030, growing at a CAGR of 10.5% from 2023. A significant portion of this growth is due to the adoption of intelligent systems, including collision detection, lane departure warnings, adaptive cruise control, and seat belt reminder systems.

Countries like Germany, Japan, USA, and India have implemented regulations requiring all new vehicles to feature seat belt reminder systems, at least for front seats. However, these systems are often only available in mid- to high-range models due to the cost of integration and proprietary systems used by car manufacturers.

In countries like India, where road traffic accidents account for over 150,000 deaths annually, awareness and enforcement of seat belt usage are critical. A study conducted by the Indian Ministry of Road Transport and Highways revealed that more than 75% of rear-seat passengers do not wear seat belts, mainly due to the lack of alert systems or any immediate consequence. Additionally, existing systems in economy vehicles are often basic, hardwired, or sensorless, making them easy to ignore or override.

Open-source platforms such as Arduino, Raspberry Pi, and ESP32 have made it possible to create cost-effective, customizable safety systems. Combined with CAN communication, these platforms enable the creation of modular systems that can scale across multiple vehicle nodes (seats, wheels, lights, etc.).

Our system provides a proof-of-concept that caters to a wide market segment, from automotive startups to educational institutes and even companies that retrofit vehicles with safety modules. By keeping costs low and using commonly available components, this system can pave the way for mass adoption in budget vehicles while maintaining compliance with safety regulations.

Problem Statement

Despite advancements in vehicle technology and the implementation of traffic safety regulations, there continues to be a significant gap in seat belt usage among drivers and passengers. Existing seat belt alert systems are either limited in functionality—monitoring only the driver's seat—or are built using proprietary technology that increases the overall cost of the vehicle. These systems often do not account for actual seat occupancy, resulting in false alerts or complete failure to detect passenger negligence.

The absence of a cost-effective, modular, and intelligent alert mechanism leads to preventable injuries and fatalities. Moreover, in many vehicles, especially in low-income markets, no proper detection system is installed at all. There is a pressing need for a system that not only detects the presence of a passenger but also actively monitors whether the seat belt is fastened, and alerts the user in real-time through multiple communication methods.

This project addresses this gap by developing a dual-node seat belt alert system using CAN bus communication, simulating real-world automotive electronics while remaining within a limited budget and using open-source technologies.

Business Level Requirement Analysis

To ensure that the project is aligned with both practical utility and industry expectations, we have defined several high-level requirements:

Functionality: The system must detect seat occupancy and seat belt status. It must generate alerts only when necessary (i.e., when the seat is occupied and the seat belt is not fastened).

Scalability: The system should be extendable to multiple seats in a vehicle, each functioning as a separate node in the CAN network.

Affordability: Components must be low-cost and readily available, suitable for mass production or educational deployment.

Ease of Integration: The design should allow easy integration into existing vehicle dashboards and electrical systems.

Real-Time Monitoring: The system must provide immediate feedback through audio (buzzer) and visual (LCD) signals.

Reliability: The alert system must be consistent in performance and must not generate false positives/negatives due to vibrations or electronic noise.

These requirements guide the design choices, such as the use of Arduino Uno, force sensors, relay switches, LCD modules, and MCP2515 CAN interfaces. The

combination provides a robust system capable of real-world application with future enhancements such as Bluetooth logging, mobile alerts, or cloud-based reporting.

Feature 1: Assist the Driver by Turning the Direction of the Lights Along with the Road Using Input from the Steering Wheel

Driving at night or in low-visibility conditions poses significant risks, especially when turning on sharp curves. A key innovation in modern vehicles is adaptive headlights that rotate in the direction of the steering to illuminate the road ahead more effectively.

Our system can incorporate this feature using a rotary encoder or potentiometer connected to the vehicle's steering column. As the steering wheel is turned, the angle is read by the encoder and sent to the microcontroller , which in turn adjusts servo motors mounted on the headlights. This allows the beam to pivot left or right, following the steering movement.

By synchronizing this feature with CAN communication, multiple lighting modules can respond to changes in steering input, thereby mimicking intelligent lighting systems found in luxury vehicles, but at a fraction of the cost.

Feature 2: Automatically Turn the Headlights ON/OFF Based on Ambient Light Outside the Vehicle

Headlight control based on ambient light is another safety feature increasingly adopted in modern cars. Using an LDR (Light Dependent Resistor) or a digital ambient light sensor, the system continuously measures the surrounding light intensity.

When the sensor detects low-light conditions (such as nightfall, tunnels, or heavy rain), it automatically activates the vehicle's headlights. Once sufficient lighting is restored, the headlights are turned off. This feature reduces human error, ensures proper visibility, and improves fuel efficiency in conventional cars by reducing battery load.

This automation can be implemented on the same Arduino-CAN architecture by transmitting ambient light data to the lighting module in the CAN network.

Feature 3: Turn Off LEDs of the Matrix Headlights So That Light Beam Doesn't Fall Directly on the Vehicles Ahead

Matrix headlights represent the cutting-edge of vehicle lighting systems. These are made up of multiple LED segments that can be individually controlled to shape the light beam dynamically. A common application is to prevent the beam from dazzling oncoming drivers while still providing maximum illumination elsewhere.

In our concept, ultrasonic sensors or IR sensors placed at the front of the vehicle can detect the presence and distance of oncoming vehicles. Based on this data, certain LEDs in the matrix array are turned off or dimmed, creating a 'shadow' in the beam that avoids shining directly into the driver's eyes.

This feature enhances nighttime driving safety and demonstrates how even a low-cost embedded system can replicate premium car technology using basic components and intelligent programming.

System Architecture Overview

Design Paradigm: Master-Slave Architecture

In this system, communication and responsibilities are split between two microcontroller nodes using the CAN (Controller Area Network) protocol:

A. Slave Node — Arduino 1 (Sensor Node)

Primary Responsibilities:

1. Sensor Interface:

   Reads input from two key sensors:

   Seat Occupancy Sensor (analog or digital input)

   Seat Belt Sensor (push button or switch)

2. Data Processing:

   Converts sensor readings into a simplified data format.

3. CAN Communication (Transmission):

   Sends the formatted data over the CAN bus to the Master Node using the MCP2515 CAN module.

4. Components Connected:

   Seat Occupancy Sensor → A0 (analog input)

   Seat Belt Sensor → Digital Pin 7

   MCP2515 CAN Module:

   VCC → 5V

   GND → GND

   CS →Digital Pin 10

   INT →Digital Pin 2

SPI: SCK → 13, MOSI → 11, MISO → 12

B. Master Node — Arduino 2 (Controller Node)

Primary Responsibilities:

- CAN Communication (Reception):

  Listens for incoming messages from the Sensor Node via CAN bus.

- Decision Logic:

  Analyzes received seat and belt data.

- User Interface:

  Displays messages on an I2C LCD (seat/belt status).

  Activates buzzer when seat is occupied but belt is not fastened.

- Components Connected:

  MCP2515 CAN Module (same wiring as above)

  Buzzer → Digital Pin 6

  LCD (I2C) → SDA → A4, SCL → A5

CAN Message Protocol

Byte Index	Parameter	Possible Values	Description
0	Seat Status	0x00 → Empty0x01 → Occupied	Indicates whether someone is sitting on the seat.
1	Belt Status	0x00 → Unfastened0x01 → Fastened	Indicates whether the seat belt is fastened.

Message Flow Example

Scenario	CAN Payload [Byte 0, Byte 1]	Master Node Action
Seat Vacant	[0x00, 0x00]	Display "Seat is Vacant", buzzer OFF
Seat Occupied + Belt Fastened	[0x01, 0x01]	Display "Seat Occupied / Belt Fastened", buzzer OFF
Seat Occupied + Belt Unfastened	[0x01, 0x00]	Display "Belt Unfastened!", activate buzzer

Project Plan

Phase	Duration	Deliverables	Description
Requirement Analysis	2 Days	Finalized Feature Requirements	- Identify stakeholder expectations and define key system features (e.g., seat detection, belt status, alert system).- Break down functional (FR) and non-functional requirements.- Decide on communication protocol (CAN), message format, and system architecture (master-slave).- Ensure compliance with safety use-case (in-vehicle scenario simulation).
Component Setup	1 Day	Connected Hardware and Sensor Inputs	- Physically assemble hardware components: ▪ Arduino UNO x2 ▪ MCP2515 CAN Modules x2 ▪ Seat Occupancy Sensor (IR sensor / analog input) ▪ Belt Sensor (Push button or toggle) ▪ LCD 16x2 with I2C module ▪ Buzzer ▪ Wiring for CANH ↔ CANH and CANL ↔ CANL.- Validate connections using multimeter or continuity check.

CAN Communication Setup	2 Days	Working CAN link between nodes	- Initialize MCP2515 modules on both Arduinos.- Test SPI communication.- Send and receive sample messages between nodes.- Debug baud rate mismatches, grounding issues, and terminations if needed.- Ensure real-time, reliable, error-free CAN message exchange.
Code Development	3 Days	Sensor Node and Controller Node Code	- Develop Arduino code for Sensor Node: ▪ Read sensors (seat and belt) ▪ Construct CAN messages with ID and payload ▪ Transmit using MCP2515. - Develop Arduino code for Controller Node: ▪ Receive and parse CAN messages ▪ Display status on LCD ▪ Control buzzer logic. - Modularize code for maintainability.
Testing & Validation	2 Days	Verified Scenarios using Test Cases	- Simulate multiple conditions: ▪ Seat vacant ▪ Seat occupied + belt fastened ▪ Seat occupied + belt unfastened.- Validate buzzer logic.- Monitor LCD outputs.- Use Serial Monitor for debugging.- Perform regression testing after each update.- Map test results to

			Stakeholder Requirements (traceability).
Documentation	1 Day	Project Report, Diagrams, Test Log	- Prepare a detailed report: ▪ System Overview & Architecture ▪ Wiring Diagram & Photos ▪ Source Code Snapshots ▪ Test Cases and Results ▪ Traceability Matrix ▪ Challenges & Solutions.- Include class diagram, block diagram, and timeline Gantt chart if possible.

Task Deadlines

Phase	Start Date	End Date	Duration	Milestone/Outcome
Requirement Analysis	2025-03-26	2025-03-27	2 Days	Finalized feature list, architecture outline
Component Setup	2025-03-28	2025-03-28	1 Day	Assembled and connected all hardware components

CAN Communication Setup	2025-03-29	2025-03-30	2 Days	CAN modules communicate between Arduinos
Code Development	2025-03-31	2025-04-02	3 Days	Fully functional Arduino code for both nodes
Testing & Validation	2025-04-03	2025-04-04	2 Days	All scenarios tested and logged successfully
Initial Review & Feedback	2025-04-05	2025-04-05	1 Day	Feedback received from guide and peers
Final Fixes & Optimization	2025-04-06	2025-04-07	2 Days	Applied feedback-based improvements
Documentation	2025-04-08	2025-04-14	7 Days	Created diagrams, report, and test summary
Report Review & Approval	2025-04-15	2025-04-16	2 Days	Submitted for internal guide/mentor review
Final Presentation Prep	2025-04-17	2025-04-20	4 Days	Created slides, rehearsed demo

Final Submission	2025-05-01	2025-05-01	1 Day	Submitted complete project and presentation

Task Assignment

Team Member	Role	Assigned Tasks
Sonal	Hardware Lead	Wiring sensors and actuators, MCP2515 setup, breadboard/circuit layout
Jyoti	Software Developer (Slave Node)	Writing code for sensor input and CAN transmission on Arduino 1
Atharva	Software Developer (Master Node)	LCD, buzzer logic, CAN data parsing on Arduino 2
Kunal	Tester & Documentation Lead	Test case execution, result validation, final report, and diagram preparation

Components

S. No.	Component Name	Quantity	Description
1	Arduino UNO	2	Microcontroller boards used for processing sensor data and controlling output
2	MCP2515 CAN Module	2	Used for CAN communication between the two Arduino boards
3	Force Sensor (FSR)	1	Detects if the seat is occupied based on pressure
4	Push Button	1	Simulates seatbelt latch – pressed means belt is fastened
5	16x2 LCD (I2C)	1	Displays seatbelt status messages
6	Buzzer Module	1	Provides audible alert when belt is not buckled
7	10kΩ Resistor	2	Used as pull-down resistors for FSR and push button
8	Jumper Wires	20	Used for making all the electrical connections

Software Design

Class Diagram

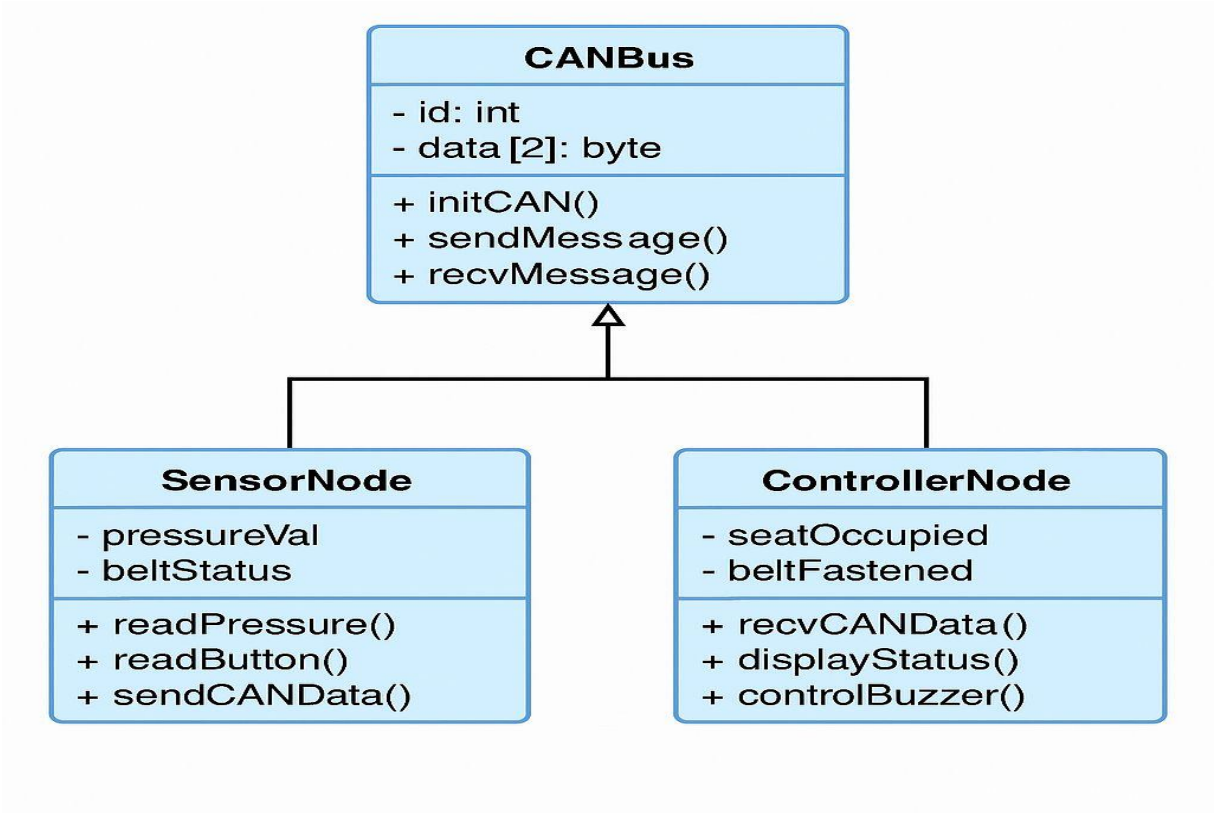

Fig 1. Class Diagram

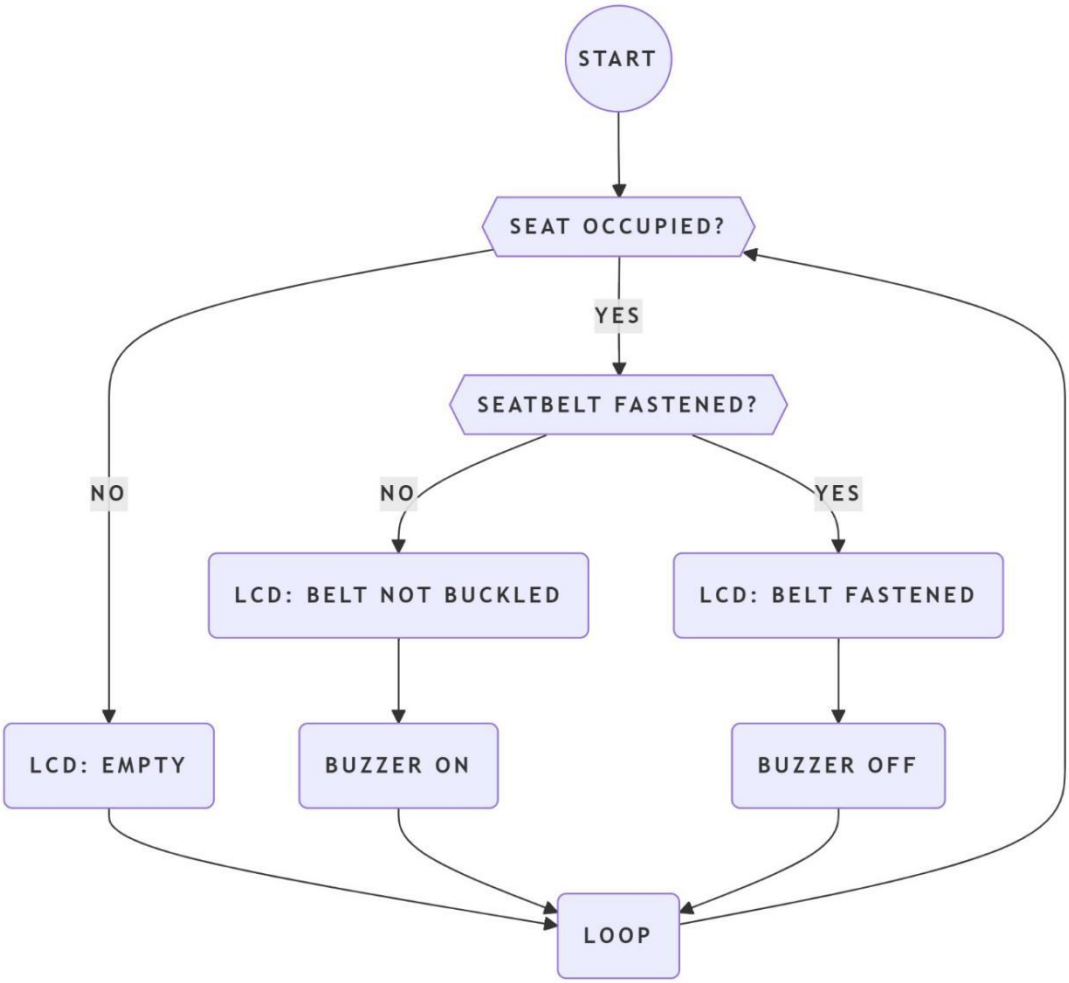

Fig 2. Flow Chart

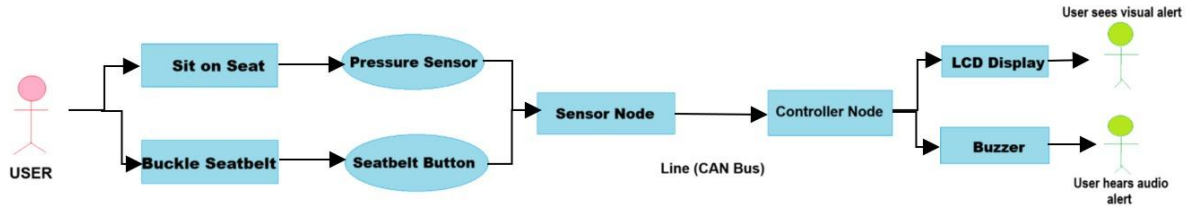

Fig 3. User Diagram

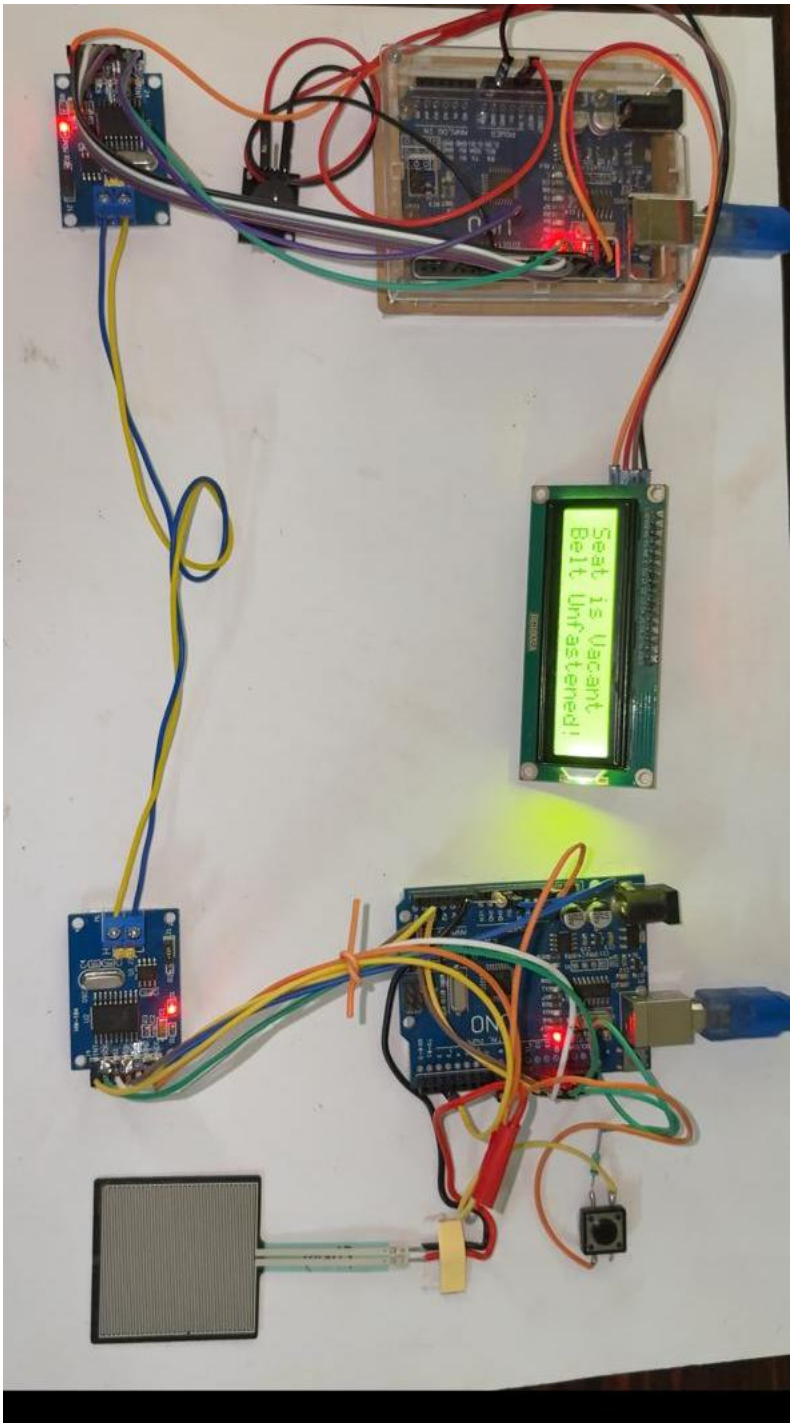

Fig 1. Circuit

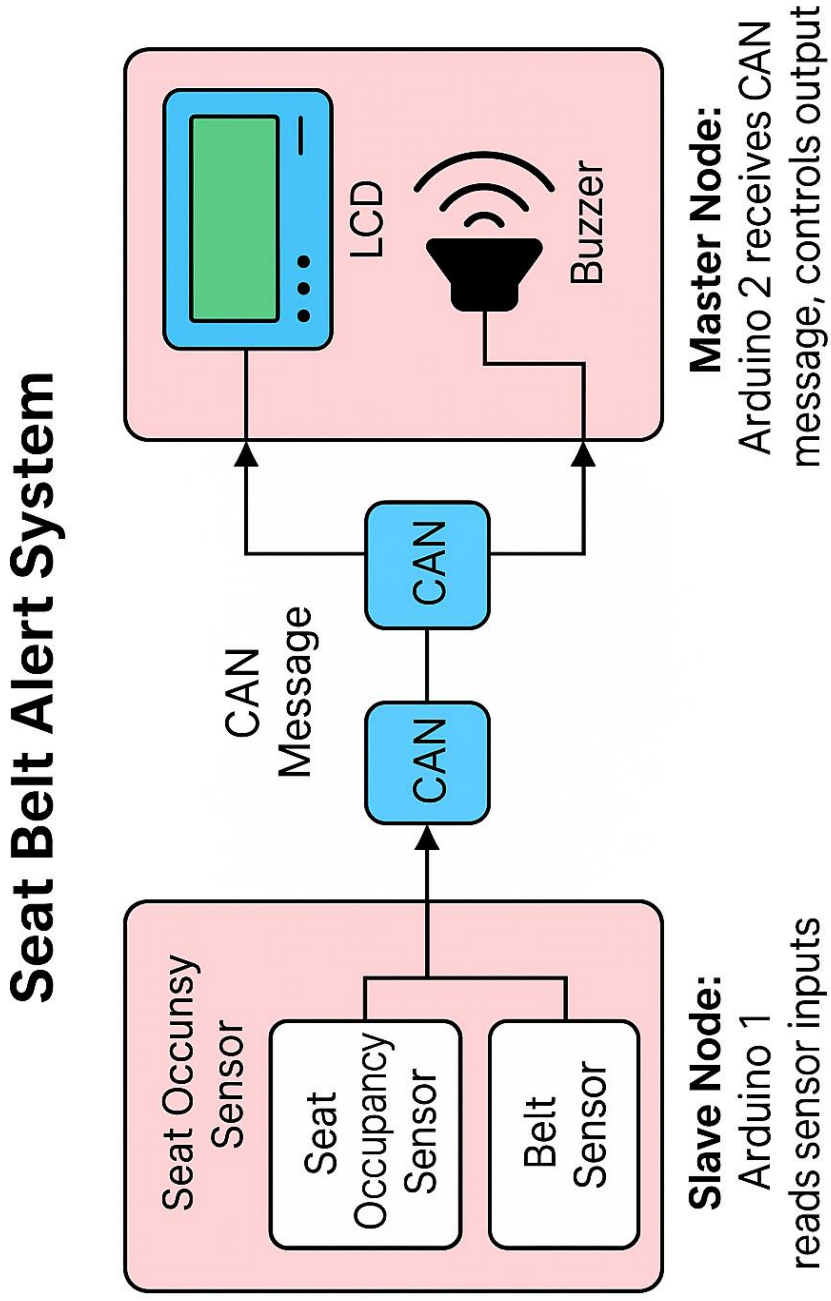

Fig 1. Hardware Diagram

Test Cases

Test Case No.	Test Scenario	Expected Output	Observed Output
TC01	Seat vacant, belt fastened	LCD: Seat is Vacant, Belt Fastened; Buzzer OFF	As expected
TC02	Seat vacant, belt unfastened	LCD: Seat is Vacant, Belt Unfastened!; Buzzer OFF	As expected
TC03	Seat occupied, belt unfastened	LCD: Seat Occupied, Belt Unfastened!; Buzzer ON	As expected
TC04	Seat occupied, belt fastened	LCD: Seat Occupied, Belt Fastened; Buzzer OFF	As expected
TC05	No data received	No change on LCD; Buzzer status unchanged	As expected
TC06	CAN disconnected	No data; LCD doesn't update; Buzzer status unchanged	As expected
TC07	Analog sensor failure (constant 0)	Always displays Seat is Vacant	As expected
TC08	Button shorted (always HIGH)	Always displays Belt Fastened	As expected

CAN Transmitter Code

```
#include <SPI.h>                    //Library for using SPI Communication

#include <mcp2515.h>                //Library for using CAN Communication (https://github.com/autowp/arduino-mcp2515/)

struct can_frame canMsg;

MCP2515 mcp2515(10);

int seat = 0;

#define buttonpin 7

void setup()

{

   pinMode(buttonpin, INPUT);

   while (!Serial);

   Serial.begin(9600);

   SPI.begin();                          //Begins SPI communication

   mcp2515.reset();

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

   mcp2515.setNormalMode();

}

void loop()

{

   int sensorValue = analogRead(A0);
```

```
  int belt = digitalRead(buttonpin);
  if(sensorValue > 20)
  seat = 1;
  else
  seat = 0;
  canMsg.can_id    = 0x036;                  //CAN id as 0x036
  canMsg.can_dlc = 8;                        //CAN data length as 8
  canMsg.data[0] = seat;                     //Update humidity value in [0]
  canMsg.data[1] = belt;                     //Update temperature value in [1]
  canMsg.data[2] = 0x00;                    //Rest all with 0
  canMsg.data[3] = 0x00;
  canMsg.data[4] = 0x00;
  canMsg.data[5] = 0x00;
  canMsg.data[6] = 0x00;
  canMsg.data[7] = 0x00;
  mcp2515.sendMessage(&canMsg);         //Sends the CAN message
  delay(500);
}
```

CAN Receiver Code

```
#include <SPI.h>                    //Library for using SPI Communication

#include <mcp2515.h>                //Library for using CAN Communication (https://github.com/autowp/arduino-mcp2515/)

#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2);    // set the LCD address to 0x3F for a 16 chars and 2 line display

struct can_frame canMsg;

MCP2515 mcp2515(10);                        // SPI CS Pin 10

#define buz 6

int seat,belt;

  void setup()

{ pinMode(buz, OUTPUT);

   digitalWrite(buz, HIGH);

   Serial.begin(9600);                       //Begins Serial Communication at 9600 baudrate

   SPI.begin();                              //Begins SPI communication

   lcd.init();

   lcd.clear();

   lcd.backlight();          // Make sure backlight is on

   lcd.setCursor(0, 0);

   lcd.print("      Seat Belt      ");

   lcd.setCursor(0, 1);
```

```
  lcd.print(" Alert    System ");
  delay(3000);
  lcd.clear();
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS, MCP_8MHZ); //Sets CAN at speed 500KBPS and Clock 8MHz
  mcp2515.setNormalMode();                   //Sets CAN at normal mode
}
void loop()
{
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) // To receive data (Poll Read)
  {
      seat = canMsg.data[0];
      belt = canMsg.data[1];
    if(seat == 1)
    {
      lcd.setCursor(0, 0);
      lcd.print("Seat Occupied      ");
    }
    else
  {
```

```
  lcd.setCursor(0, 0);
        lcd.print("Seat is Vacant    ");
     }
     if(belt == 1)
     {
        lcd.setCursor(0, 1);
        lcd.print("Belt Fastened      ");
     }
     else
     {
        lcd.setCursor(0, 1);
        lcd.print("Belt Unfastened!");
     }
  }
  if(seat == 1 && belt == 0)
  {
    digitalWrite(buz, LOW);
  }
  else
  digitalWrite(buz, HIGH);
}
```

Reference

1. MCP2515 Arduino CAN Library - https://github.com/autowp/arduino-mcp2515

2. Arduino SPI Library - https://www.arduino.cc/en/Reference/SPI

3. LiquidCrystal_I2C Library for I2C LCD - https://github.com/johnrickman/LiquidCrystal_I2C

4. CAN Protocol Overview - https://www.ni.com/en-us/innovations/can-protocol.html