SIMULATE ECU'S COMMUNICATION IN AN AUTOMOBILE

Thesis submitted in partial fulfilment of the

requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

In

TCS Academic Interface Program (AIP) during academic year 2018-2019.

By

A.M.P.UJWALA	15241A0401
M.TEJASWI	15241A0425
M.HARISH	15241A0426
MURALI ADABALA	15241A0431
D.JASWANTHI	15241A04P4

Under the Esteemed guidance of

Mr.Kishore P

Department of

Electronics and Communication Engineering

GOKARAJU RANGARAJU

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

Bachupally, Kukatpally, Hyderabad- 500 090

2019

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING

GOKARAJU RANGARAJU

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

Bachupally, Kukatpally, Hyderabad- 500 090

CERTIFICATE

This is to certify that the project report entitled **'SIMULATE ECU'S COMMUNICATION IN AN AUTOMOBILE'** is submitted by A.M.P.Ujwala (15241A0401), M.Tejaswi (15241A0425), M.Harish (15241A0426), Murali Adabala (15241A0431), D.Jaswanthi (15241A04P4) in the TCS Academic Interface Program (AIP) during academic year 2018-2019 under the guidance of Mr.Kishore P.

Tcs Mentor

Mr. Kishore P

DECLARATION

We hereby declare that this project report entitled **SIMULATE ECU'S COMMUNICATION IN AN AUTOMOBILE** is the work done during the period from **7 February 2019 to 25 March 2019** and is submitted in partial fulfilment of the requirements for the TCS Academic Interface Program (AIP) during academic year 2018-2019 under the guidance of Mr.Kishore P by the students of Bachelor of Technology in **Electronics And Communication Engineering** from Gokaraju Rangaraju Institute of Engineering and Technology (Autonomous). The results embodied in this project have not been submitted to any other university or Institution for the award of any Degree or Diploma.

A.M.P.UJWALA	15241A0401
M.TEJASWI	15241A0425
M.HARISH	15241A0426
MURALI ADABALA	15241A0431
D.JASWANTHI	15241A04P4

ABSTRACT

CAN (Controller Area Network) is a kind of real time field bus and has been widely used in the automatic control and testing field. Compared to traditional communication methods such as RS-485, it has great advantages in the data translating speed and reliability. Data communication in ECU (Electronic Control Unit) testing system has strict time requirement which is difficult for traditional communication system to achieve. According to the communication characteristics of ECU testing system, a multi-CAN communication network based on the CAN-bus is designed in this project. It introduces the basic structure, sub-nodes hardware interface and software design of the communication network in detail, and provided a real time, reliable and flexible communication system based on CAN bus.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER-1

INTRODUCTION

1.1 Introduction

To make the communication faster and reliable, we have to use Controller Area Network. An **Electronic Control Unit** (**ECU**) is any embedded system in automotive electronics that controls one or more of the electrical systems or subsystems in a vehicle.

Types of ECU include Engine Control Module (ECM), Powertrain Control Module (PCM), Transmission Control Module (TCM), Brake Control Module (BCM or EBCM), Central Control Module (CCM), Central Timing Module (CTM), General Electronic Module (GEM), Body Control Module (BCM), Suspension Control Module (SCM), control unit, or control module. Taken together, these systems are sometimes referred to as the car's computer (Technically there is no single computer but multiple ones.) Sometimes one assembly incorporates several of the individual control modules (PCM is often both engine and transmission).

Some modern motor vehicles have up to 80 ECUs. Embedded software in ECUs continues to increase in line count, complexity, and sophistication. Managing the increasing complexity and number of ECUs in a vehicle has become a key challenge for original equipment manufacturers (OEMs).

In this project we have used 3 ECUs. The 2 ECUs are for transmission and the other ECU is for reception and displaying the values. In this project an ECU is an Arduino Uno board. One transmitter ECU is connected to the IR sensor and the other transmitter ECU is connected to the Humidity sensor and LM35 temperature sensor for monitoring the humidity and temperature in an automobile. One Arduino Uno-CAN pair is connected to the other Arduino Uno-CAN pair by High-Low pins of MCP2515 CAN Module. So, in this way different ECUs are connected and the communication is established between them.

1.2 Aim of the Project

CAN (Controller Area Network) bus networks are found everywhere. They are found in vehicles, farm equipment, and in industrial environments. These networks allow for control and data acquisition. Depending on the application, they can be formed around a stringent set of standards (such as J1939) or in a 'get it done' approach suitable for an Arduino DIY project.

The aim of the project is:

- To know how different ECUs communicate in an automobile.
- To measure real time Temperature.
- To measure real time Humidity.
- To transmit and receive the data over CAN Bus.
- To analyze the data over CAN Bus.
- To sense the obstacle.

1.3 Scope of the Project

- To interface Arduino Uno boards with MCP2515 CAN modules:
    1. Interface 3 Arduino Uno boards with 3 MCP2515 CAN modules i.e. interface each Arduino Uno board with each MCP2515 CAN module for sending the data from Arduino Uno board to the CAN module.
- To interface CAN modules to each other:
    1. Interface one CAN module with the other using high-low pins on the CAN module. This works as CAN Bus.
- To interface ECUs to the sensors:
    1. Interface one transmitter ECU to the IR sensor for obstacle detection.

2. Interface the other transmitter ECU to the Humidity sensor and LM35 Temperature sensor for Humidity monitoring and Temperature monitoring respectively.
3. Interface the receiver ECU to the serial monitor for displaying the readings or values.

- To use Arduino IDE for code execution (for transmitters and receiver).

CHAPTER-2

PROJECT OVERVIEW

Block Diagram

The figure 2.1 shown below is the Block Diagram of ECU communication.

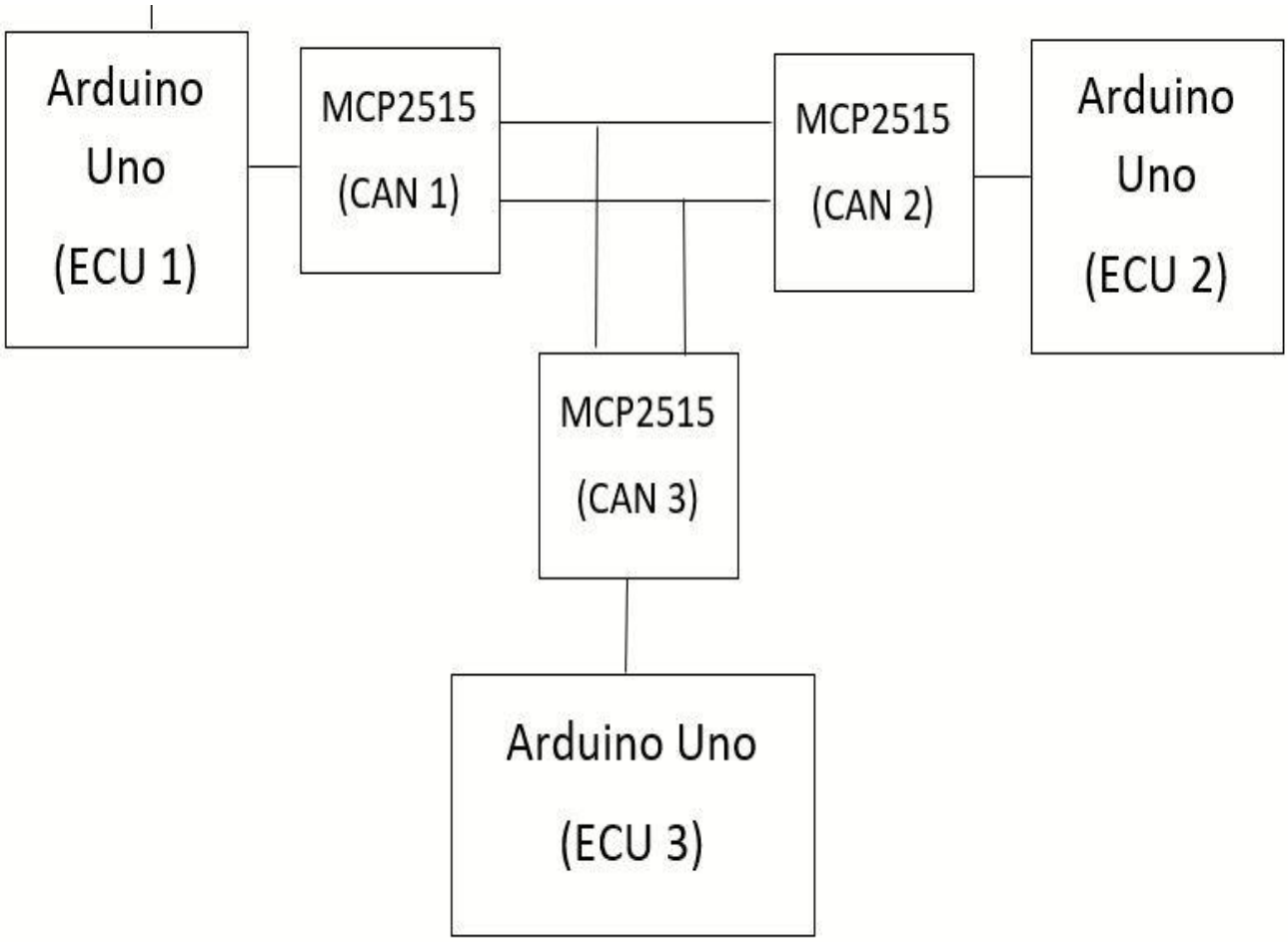

Figure 2.1 Block Diagram of ECU communication

CHAPTER-3

HARDWARE REQUIREMENTS

Following are the components which are used in this project

3.1 Arduino UNO

Figure 3.1 Arduino UNO Board

The figure 3.1 shows Arduino UNO which is an open-source microcontroller board based on the Microchip **ATmega328P microcontroller** and developed by Arduino.cc. The board is equipped with sets of **digital and analog input/output (I/O) pins** that may be interfaced to various expansion boards (shields) and other circuits.

The board has **14 Digital pins , 6 Analog pins**, and programmable with the **Arduino IDE** (Integrated Development Environment) via a type B USB cable. It can be powered by a USB cable or by an external 9 volt battery, though it accepts voltages between **7 and 20 volts**.

Specifications :

Microcontroller: Microchip ATmega328P

Operating Voltage: 5 Volts

Input Voltage: 7 to 20 Volts

Digital I/O Pins: 14 (of which 6 provide PWM output)

Analog Input Pins: 6

DC Current per I/O Pin: 20 mA

DC Current for 3.3V Pin: 50 mA

Flash Memory: 32 KB of which 0.5 KB used by bootloader

SRAM: 2 KB

EEPROM: 1 KB

Clock Speed: 16 MHz

Pin configuration of ATMega328P

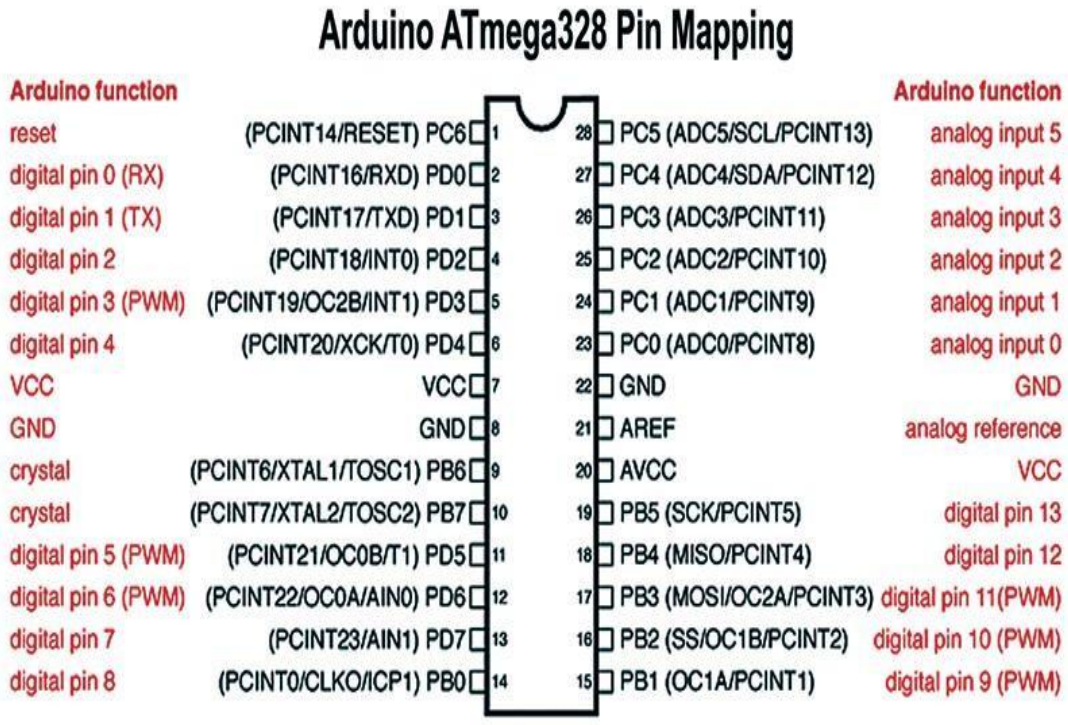

Figure 3.2 Arduino UNO Pin Mapping

3.2 MCP2515 CAN Module

The figure 3.3 shows MCP2515 CAN Bus Controller. It is a simple Module that supports CAN Protocol version 2.0B and can be used for communication at 1Mbps. In order to setup a complete communication system, you will need two CAN Bus Module.

The module used in the project is shown in the image below.

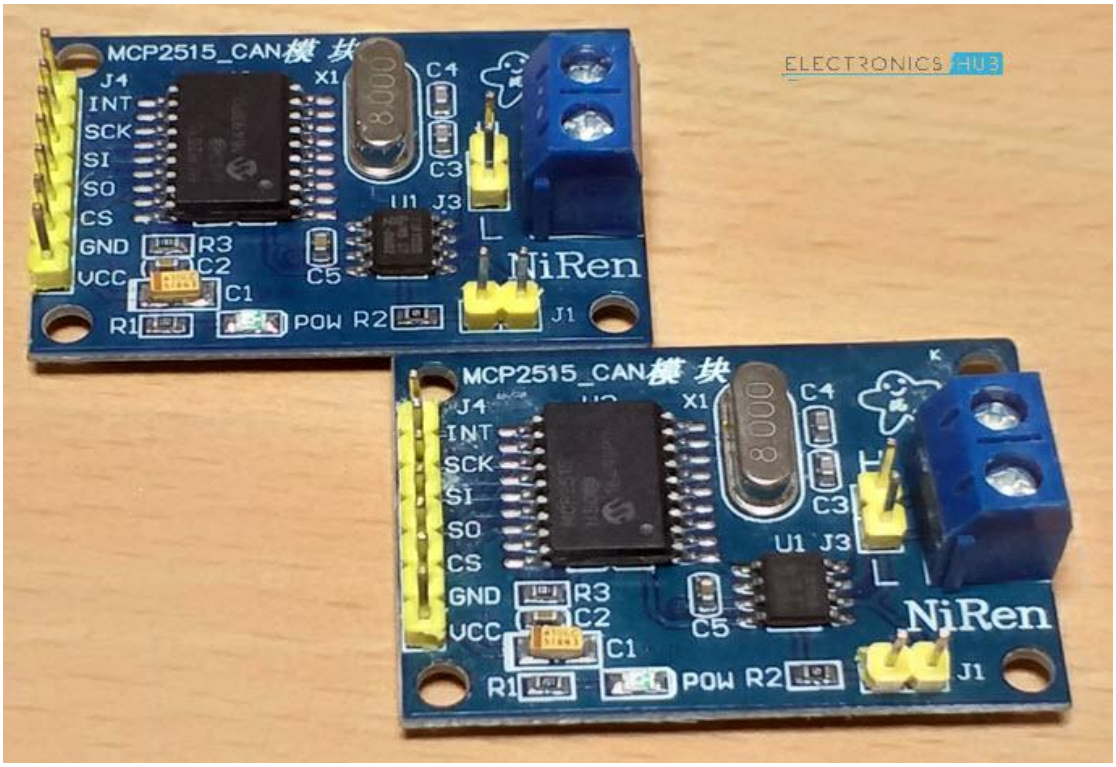

Figure 3.3 MCP2515 CAN Module

This particular module is based on MCP2515 CAN Controller IC and TJA1050 CAN Transceiver IC. The MCP2515 IC is a standalone CAN Controller and has integrated SPI Interface for communication with microcontrollers.

Coming to the TJA1050 IC, it acts as an interface between the MCP2515 CAN Controller IC and the Physical CAN Bus.

Communication to this module with your Arduino is achievable via SPI. The device includes a jumper configurable 120 Ohm Resistor and a CAN V2.B

support. More, there are existing libraries that make building projects relatively easy.

The figures 3.4a and 3.4b shows the components and pins on a typical MCP2515 Module.

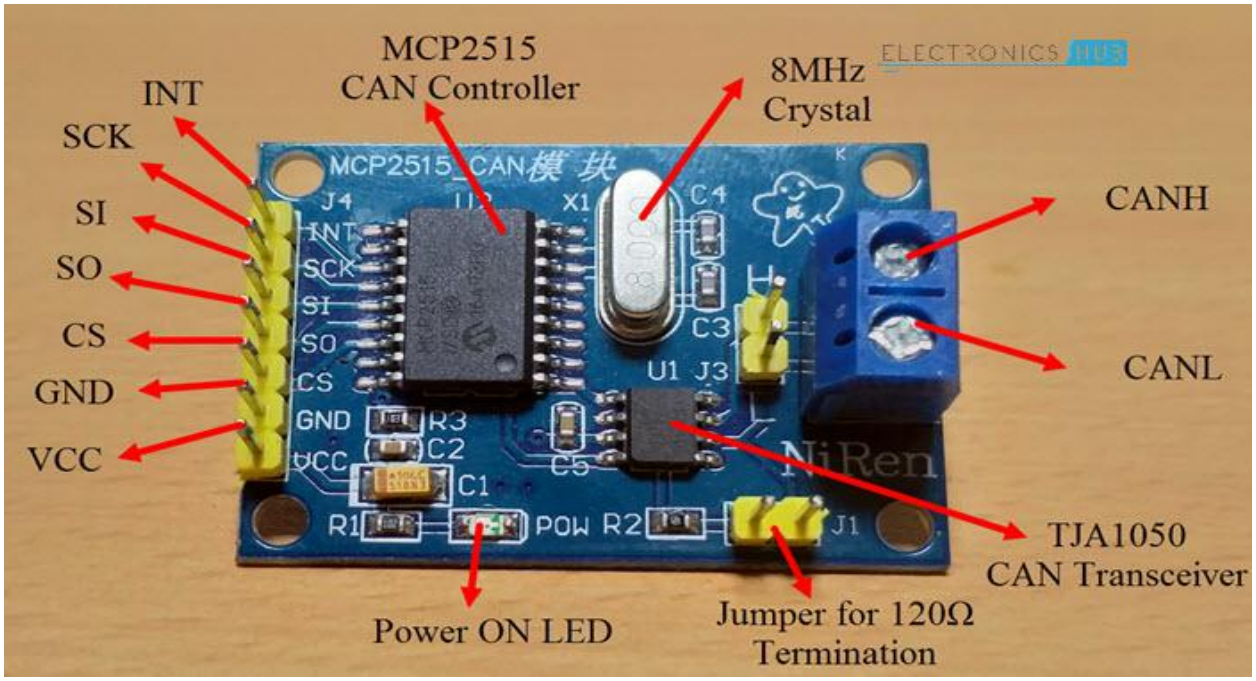

Figure 3.4a Components labelling of MCP2515 CAN Module

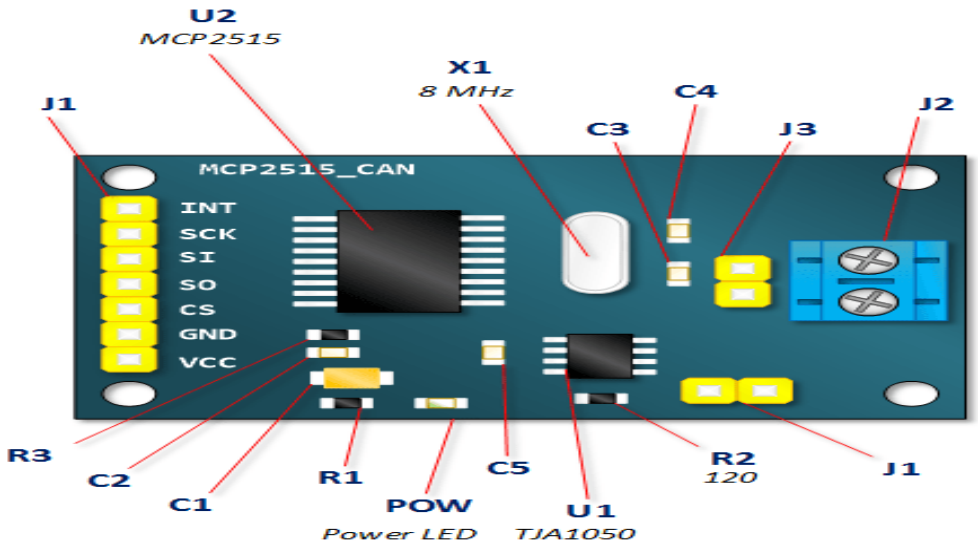

Figure 3.4b Components labelling of MCP2515 CAN Module

3.3 IR Sensor Module

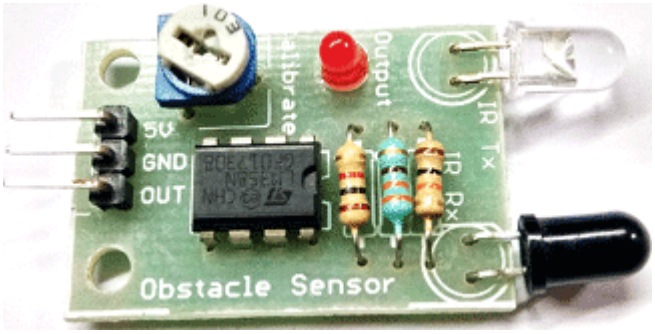

Figure 3.5 IR Sensor Module

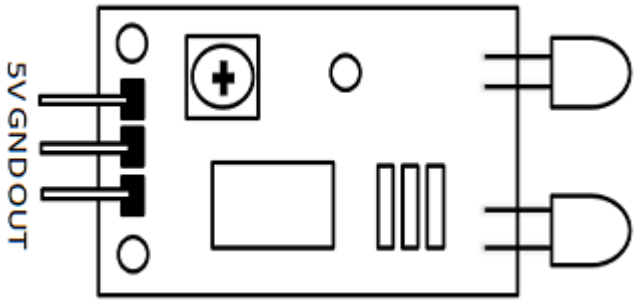

Figure 3.6 IR Sensor Module Pinout

Pin Configuration

Pin Name	Description
VCC	Power Supply Input
GND	Power Supply Ground
OUT	Active High Output

Table 3.1 IR Sensor Pin Configuration

IR Sensor Module Features

- 5VDC Operating voltage
- I/O pins are 5V and 3.3V compliant

- Range: Up to 20cm
- Adjustable Sensing range
- Built-in Ambient Light Sensor
- 20mA supply current
- Mounting hole

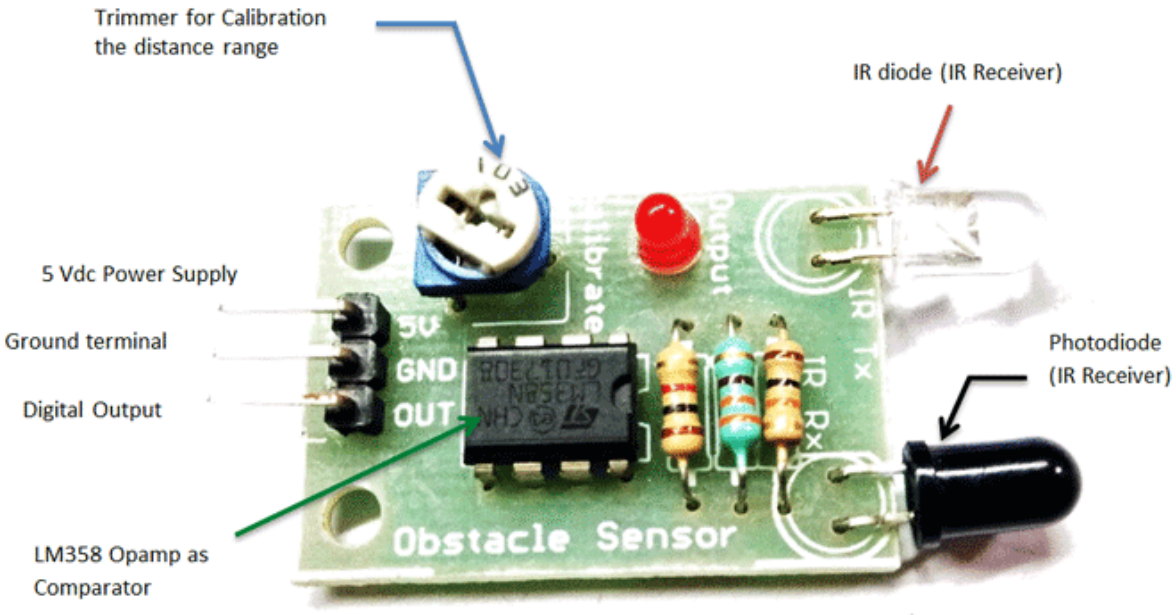

Figure 3.7 IR Sensor Components Labelling

The IR sensor module consists mainly of the IR Transmitter and Receiver, Opamp, Variable Resistor (Trimmer pot), output LED in brief.

IR LED Transmitter

IR LED emits light, in the range of Infrared frequency. IR light is invisible to us as its wavelength (700nm – 1mm) is much higher than the visible light range. IR LEDs have light emitting angle of approx. 20-60 degree and range of approx. few centimeters to several feets, it depends upon the type of IR transmitter and the manufacturer. Some transmitters have the range in kilometers. IR LED white or transparent in colour, so it can give out amount of maximum light.

Photodiode Receiver

Photodiode acts as the IR receiver as its conducts when light falls on it. Photodiode is a semiconductor which has a P-N junction, operated in Reverse

Bias, means it start conducting the current in reverse direction when Light falls on it, and the amount of current flow is proportional to the amount of Light. This property makes it useful for IR detection. Photodiode looks like a LED, with a black colour coating on its outer side, Black colour absorbs the highest amount of light.

LM358 Opamp

LM358 is an Operational Amplifier (Op-Amp) is used as voltage comparator in the IR sensor. the comparator will compare the threshold voltage set using the preset (pin2) and the photodiode's series resistor voltage (pin3).

Photodiode's series resistor voltage drop > Threshold voltage = Opamp output is High

Photodiode's series resistor voltage drop < Threshold voltage = Opamp output is Low

When Opamp's output is **high** the LED at the Opamp output terminal **turns ON** (Indicating the detection of Object).

Variable Resistor

The variable resistor used here is a preset. It is used to calibrate the distance range at which object should be detected.

3.4 DHT11–Temperature and Humidity Sensor

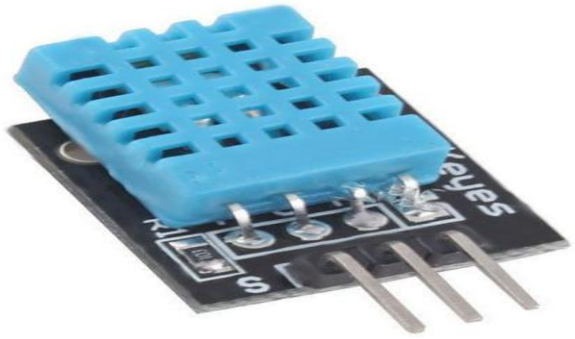

Figure 3.8 DHT11–Temperature and Humidity Sensor

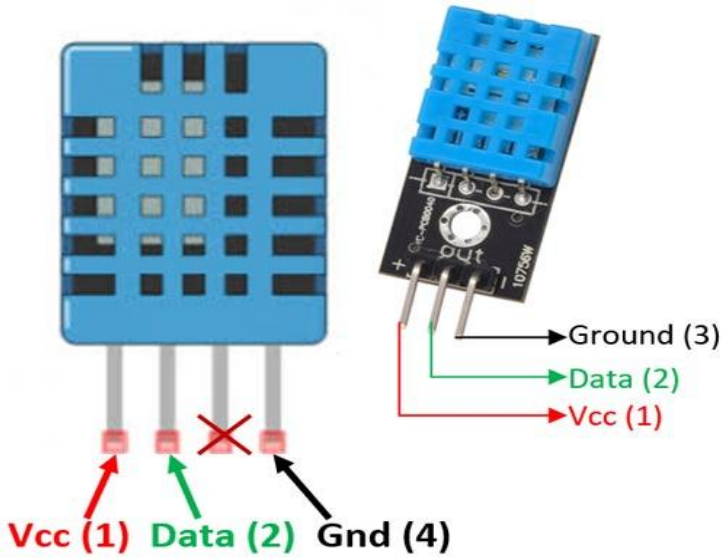

Figure 3.9 DHT11 Sensor Pinout

Pin Identification and Configuration

For Sensor

No:	Pin Name	Description
1	Vcc	Power supply 3.5V to 5.5V
2	Data	Outputs both Temperature and Humidity through serial Data
3	NC	No Connection and hence not used
4	Ground	Connected to the ground of the circuit

Table 3.2 Humidity Sensor Pin Configuration for sensor

For module

1	Vcc	Power supply 3.5V to 5.5V
2	Data	Outputs both Temperature and Humidity through serial Data
3	Ground	Connected to the ground of the circuit

Table 3.3 Humidity Sensor Pin Configuration for module

DHT11 Specifications

- Operating Voltage: 3.5V to 5.5V
- Operating current: 0.3mA (measuring) 60uA (standby)
- Output: Serial data
- Temperature Range: 0°C to 50°C
- Humidity Range: 20% to 90%
- Resolution: Temperature and Humidity both are 16-bit
- Accuracy: ±1°C and ±1%

3.5 LM35 Temperature Sensor

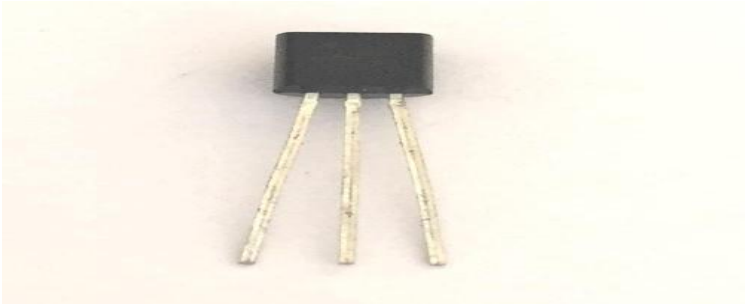

Figure 3.10 LM35 Temperature Sensor

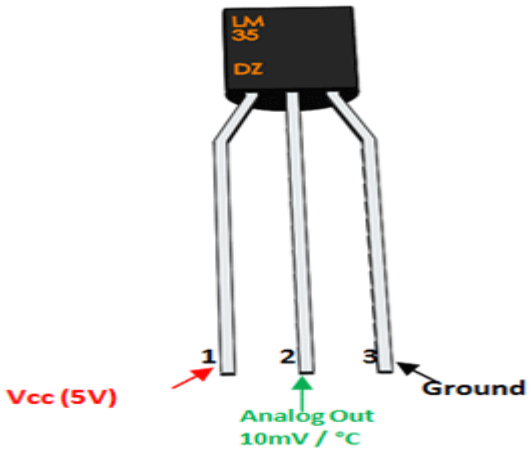

Figure 3.11 LM35 Temperature Sensor Pinout

Pin Configuration

Pin Number	Pin Name	Description
1	Vcc	Input voltage is +5V for typical applications
2	Analog Out	There will be increase in 10mV for raise of every 1°C. Can range from -1V(-55°C) to 6V(150°C)
3	Ground	Connected to ground of circuit

Table 3.4 LM35 Pin Description

LM35 Regulator Features:

- Minimum and Maximum Input Voltage is 35V and -2V respectively. Typically 5V.
- Can measure temperature ranging from -55°C to 150°C.
- Output voltage is directly proportional (Linear) to temperature (i.e.) there will be a rise of 10mV (0.01V) for every 1°C rise in temperature.

- ±0.5°C Accuracy.
- Drain current is less than 60uA .
- Low cost temperature sensor.
- Small and hence suitable for remote applications.
- Available in TO-92, TO-220, TO-CAN and SOIC package.

CHAPTER-4

SOFTWARE REQUIREMENTS

4.1 Introduction to Arduino IDE

- Arduino IDE is an open source software that is largely used for writing and compiling the code into the Arduino devices.
- It is simply accessible for Operating systems such as MAC, Windows, Linux and also executes on the Java Platform that comes with built-in functions and instructions that play an important role for debugging, editing and compiling the code in the environment.
- It is a formal Arduino software, making code compilation too simple that even a normal person with no previous technical knowledge can get on with the learning process.
- There are a range of arduino modules obtainable in the market, including Arduino Uno, Arduino Mega, Arduino Nano, Arduino Micro and many more.
- The Figure 4.1 shows the desktop logo for the Arduino IDE

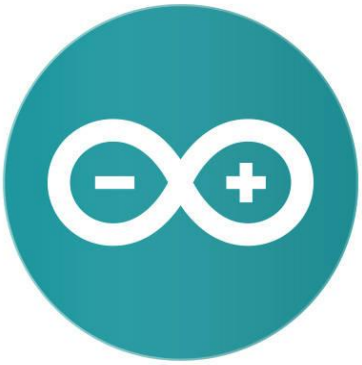

Figure 4.1 Arduino IDE Logo

- Each of them consists of a microcontroller on the board module that is previously programmed and it accepts the particulars in the form of exe-code.
- The Arduino IDE assists the languages such as C and C++ using special commands of code formatting. The Arduino IDE provides maximum software libraries, which provide many frequent input and output procedures.
- The Integrated Development Environment mainly consists of two parts.
- Editor and Compiler where former is utilised for writing the executable code and later is utilised for compiling and uploading the executable code into the Arduino Board.
- The primary code, also known as a sketch, done on the IDE platform will finally generate a Hex File which is then transmitted and uploaded in the microcontroller of the given Arduino board module.

4.2 How to Download Arduino IDE

One can download the Arduino IDE from Arduino's official website. As I mentioned earlier, the software is accessible for normal operating systems like Windows, Linux, and MAC, so make perfectly sure you are downloading the correct version of the Arduino IDE that is easily feasible with the given operating system.

- If you target to download Windows application version, make perfectly sure you have Windows 8.1 or Windows 10 versions, as app version is not feasible with Windows 7 or older versions of this operating system.

4.3 Different Sections in Arduino IDE

The IDE environment is primarily divided into three parts

- 1. Menu Bar
- 2. Text Editor
- 3. Output Pane

The Figure 4.2 below shows the description of Arduino IDE

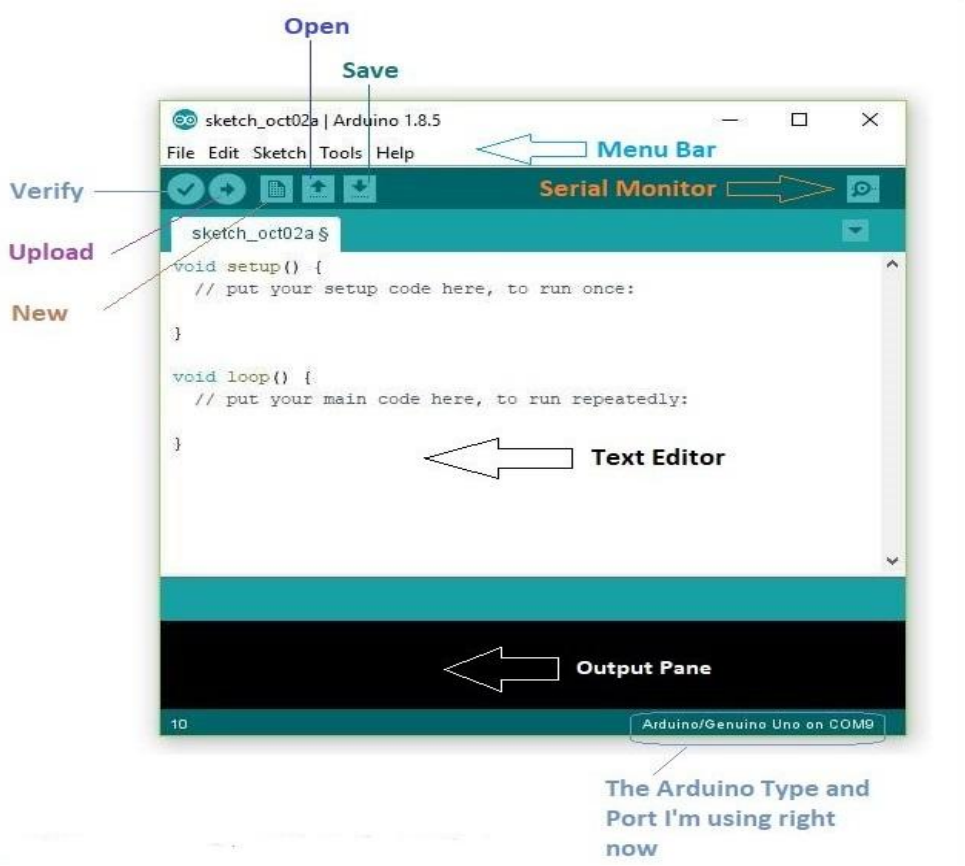

Figure 4.2 Description of Arduino IDE

- The button present at the top right corner is a Serial Monitor – A distinct pop-up window that behaves as an independent output terminal and performs a key role for transmitting and getting the Serial Data bits. You can also go to the Tools panel and click on the Serial Monitor in the drop down box, or pressing Ctrl+Shift+M all at once will unlatch it instantly. Your Arduino Board should be interfaced to your computer/laptop with the help of a USB cable in order to begin the Serial Monitor. The Serial

Monitor will actually aid us to debug the written codes where you can get to know how your program is operating.

- You need to pick the baud rate of the Arduino Module you are utilizing right now in the serial monitor at the bottom right corner.

- In order to upload the executable code, you must pick the pertinent board you are utilizing and the ports for that particular operating system. As you click on the Tools in the Menu bar, it will unlatch such as the figure 4.3 below

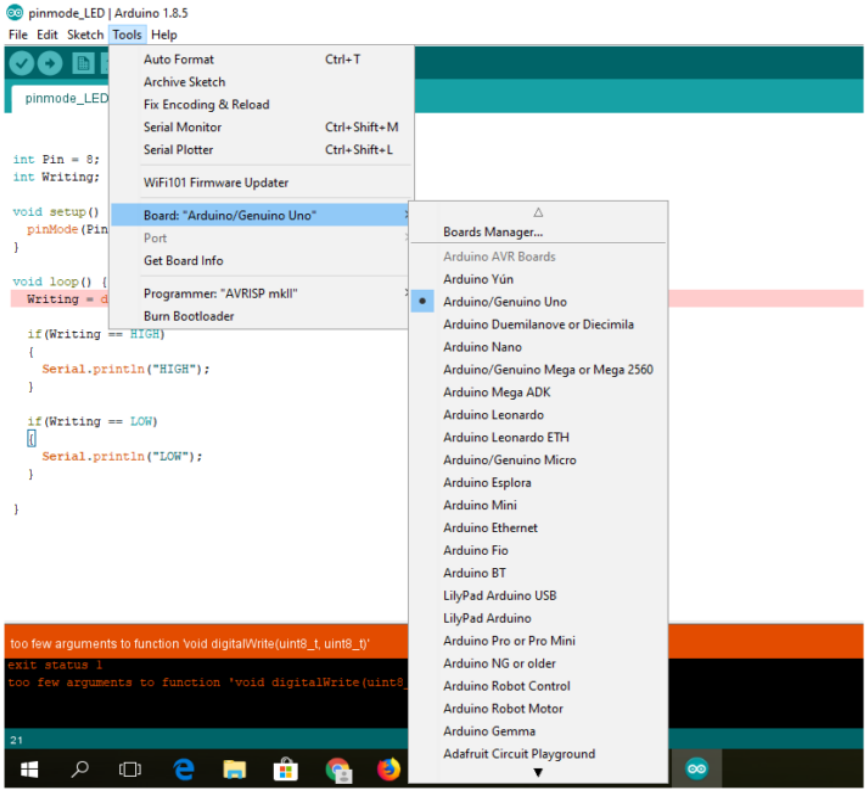

Figure 4.3 Board selection in Arduino IDE

- Just proceed to the “Board” section and pick the board module you target to work on with. Likewise, COM1, COM2, COM4, COM5, COM7 or higher are reticent for the serial and USB board. You can peer for the USB serial

device in the port part of the Device Manager of the windows operating system.

- The Figure 4.4 below shows the COM4 that I have utilized for my project, specifying the Arduino Nano with COM4 port at the bottom right corner of the screen which is highlighted in the red coloured box.

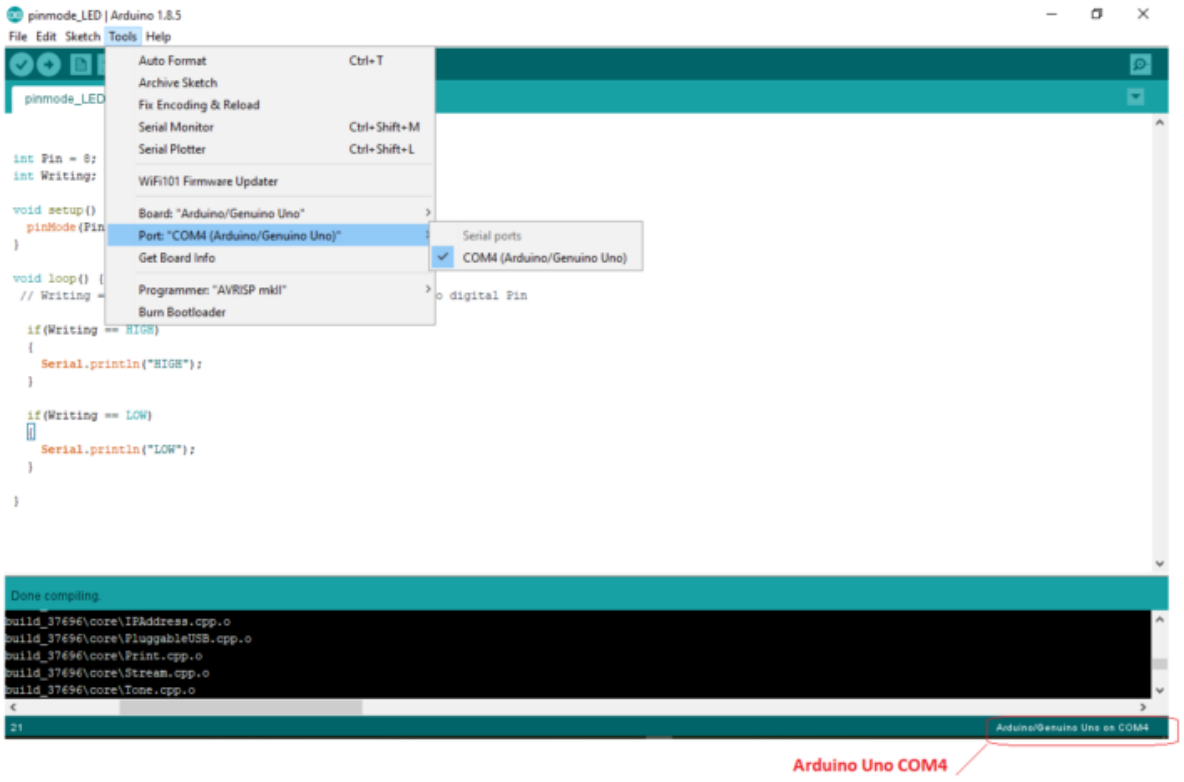

Figure 4.4 Port selection in Arduino IDE

CHAPTER-5

LIBRARIES

Libraries Used

5.1 MCP_CAN

CAN-BUS is a common industrial bus because of its long travel distance, medium communication speed and high reliability. It is commonly found on modern machine tools and as an automotive diagnostic bus. This CAN-BUS Shield adopts MCP2515 CAN Bus controller with SPI interface and MCP2551 CAN transceiver to give your Arduino/Seeeduino CAN-BUS capability. With an OBD-II converter cable added on and the OBD-II library imported, you are ready to build an onboard diagnostic device or data logger.

- Implements CAN V2.0B at up to 1 Mb/s
- SPI Interface up to 10 MHz
- Standard (11 bit) and extended (29 bit) data and remote frames
- Two receive buffers with prioritized message storage
- Industrial standard 9 pin sub-D connector
- Two LED indicators

Installation

git clone https://github.com/Seeed-Studio/CAN_BUS_Shield.git

or download zip.

Usage

Simply copy the CAN_BUS_Shield folder to your Arduino library collection. For example, arduino-1.6.12/libraries. Next time you run the Arduino IDE, you'll have a new option in Sketch -> Include Library -> CAN_BUS_Shield. Review the included examples in CAN_BUS_Shield/examples.

1. Set the Baud Rate

This function is used to initialize the baudrate of the CAN Bus system.

The available baudrates are listed as follows:

```
#define CAN_5KBPS    1
#define CAN_10KBPS   2
#define CAN_20KBPS   3
#define CAN_25KBPS   4
#define CAN_31K25BPS 5
#define CAN_33KBPS   6
#define CAN_40KBPS   7
#define CAN_50KBPS   8
#define CAN_80KBPS   9
#define CAN_83K3BPS  10
#define CAN_95KBPS   11
#define CAN_100KBPS  12
#define CAN_125KBPS  13
#define CAN_200KBPS  14
#define CAN_250KBPS  15
```

```
#define CAN_500KBPS  16

#define CAN_666kbps  17

#define CAN_1000KBPS 18
```

2. Set Receive Mask and Filter

There are 2 receive mask registers and 5 filter registers on the controller chip that guarantee you get data from the target device. They are useful especially in a large network consisting of numerous nodes.

We provide two functions for you to utilize these mask and filter registers. They are:

```
init_Mask(unsigned char num, unsigned char ext, unsigned char ulData);
init_Filt(unsigned char num, unsigned char ext, unsigned char ulData);
```

num represents which register to use. You can fill 0 or 1 for mask and 0 to 5 for filter.

ext represents the status of the frame. 0 means it's a mask or filter for a standard frame. 1 means it's for a extended frame.

ulData represents the content of the mask of filter.

3. Check Receive

The MCP2515 can operate in either a polled mode, where the software checks for a received frame, or using additional pins to signal that a frame has been received or transmit completed. Use the following function to poll for received frames.

```
INT8U MCP_CAN::checkReceive(void);
```

The function will return 1 if a frame arrives, and 0 if nothing arrives.

4. Get CAN ID

When some data arrives, you can use the following function to get the CAN ID of the "send" node.

```
INT32U MCP_CAN::getCanId(void);
```

5. **Send Data**

```
CAN.sendMsgBuf(INT32U id, INT8U ext, INT8U len, INT8U *buf);
```

This is a function to send data onto the bus. In which:

id represents where the data come from.

ext represents the status of the frame. '0' means standard frame. '1' means extended frame.

len represents the length of this frame.

buf is the content of this message.

For example, In the 'send' example, we have:

```
unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};
```

CAN.sendMsgBuf(0x00, 0, 8, stmp); //send out the message 'stmp' to the bus and tell other devices this is a standard frame from 0x00.

6. **Receive Data**

The following function is used to receive data on the 'receive' node:

```
CAN.readMsgBuf(INT8U *len, INT8U *buf);
```

Under the condition that masks and filters have been set, this function will only get frames that meet the requirements of those masks and filters.

len represents the data length.

buf is where you store the data.

7. **Check Additional Flags**

When frame is received you may check whether it was remote request and whether it was an extended (29bit) frame.

```
CAN.isRemoteRequest();
CAN.isExtendedFrame();
```

return value is '0' for a negative response and '1' for a positive

8. Sleep Mode

By setting the MCU, the CAN controller (MCP2515) and the transceiver (MCP2551) into sleep mode, you can reduce the power consumption of the whole setup from around 50mA down to 240uA (Arduino directly connected to 5V, regulator and power LED removed). The node will wake up when a new message arrives, process the message and go back to sleep afterwards.

Look at the examples "receive_sleep" and "send_sleep" for more info.

In order to set the MCP2551 CAN transceiver on the shield into sleep/standby mode, a small hardware modification is necessary. The Rs pin of the transceiver must either be connected to pin RX0BF of the MCP2515 or to a free output of the Arduino, both via the resistor R1 (17k). Cut the connection to ground after R1 and solder in a wire to one of the pins. Note however that from now on, you have to pull this pin low in software before using the transceiver. Pulling the pin high will set the transceiver into standby mode.

Without this modification, the transceiver will stay awake and the power consumption in sleep mode will be around 8mA - still a significant improvement!

5.2 DHT

Examples include both a "standalone" DHT example, and one that works along with the Adafruit Unified Sensor Library. Unified sensor library is required even if using the standalone version.

Recent Arduino IDE releases include the Library Manager for easy installation. Otherwise, to download, click the DOWNLOADS button in the top right corner, rename the uncompressed folder DHT. Check that the DHT folder contains DHT.cpp and DHT.h. Place the DHT library folder your /libraries/ folder. You may need to create the libraries subfolder if its your first library. Restart the IDE.

5.3 SPI

SPI is Serial Peripheral Interface. We also include this for serial interfacing.

CHAPTER-6

IMPLEMENTATION

6.1 Interfacing Arduino UNO with MCP2515 CAN Module

The figure 6.1 shows the circuit diagram of interfacing MCP2515 CAN Module with Arduino and possible communication between two Arduino over CAN Protocol.

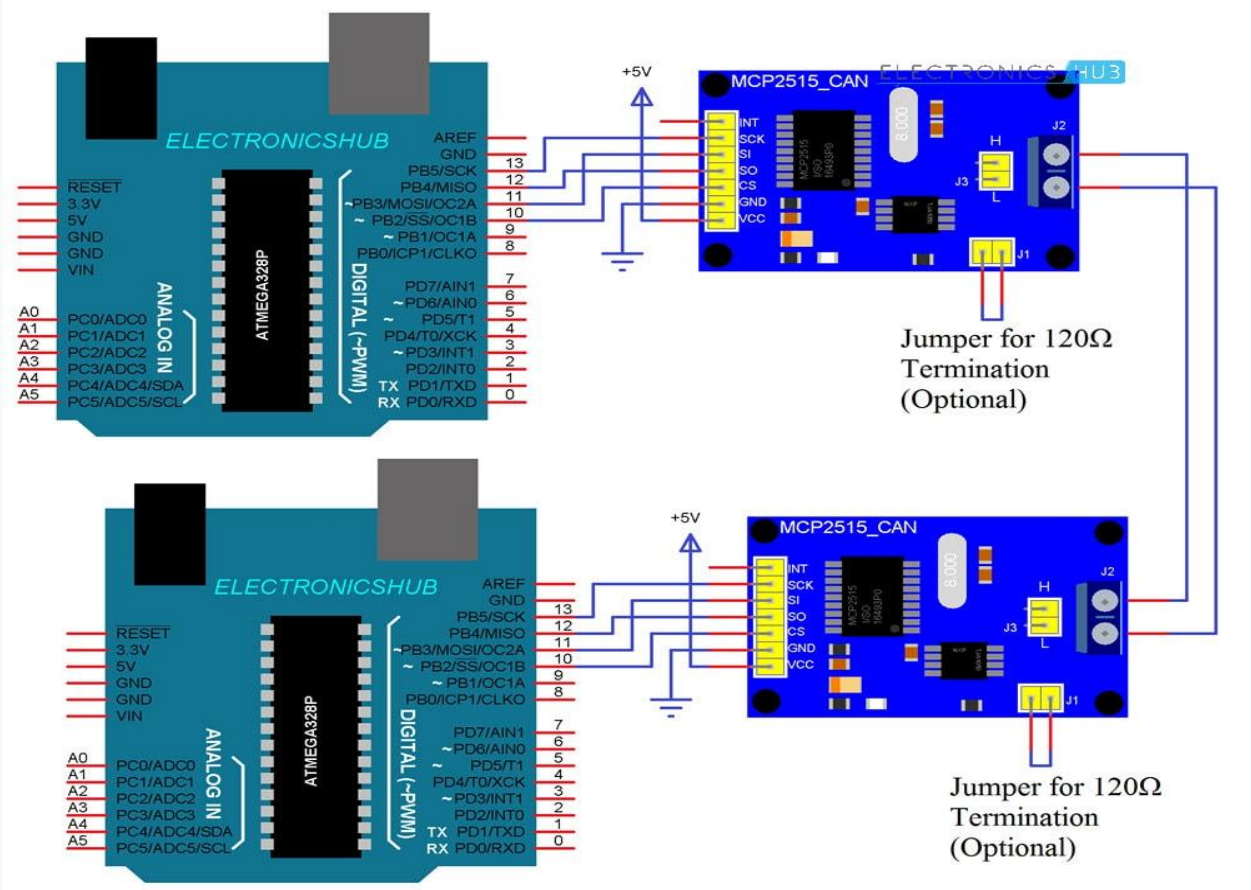

Figure 6.1 Interfacing Arduino UNO with MCP2515

6.2 Interfacing IR Sensor with Arduino UNO

The figure 6.2 shows the interfacing of IR Sensor with Arduino UNO

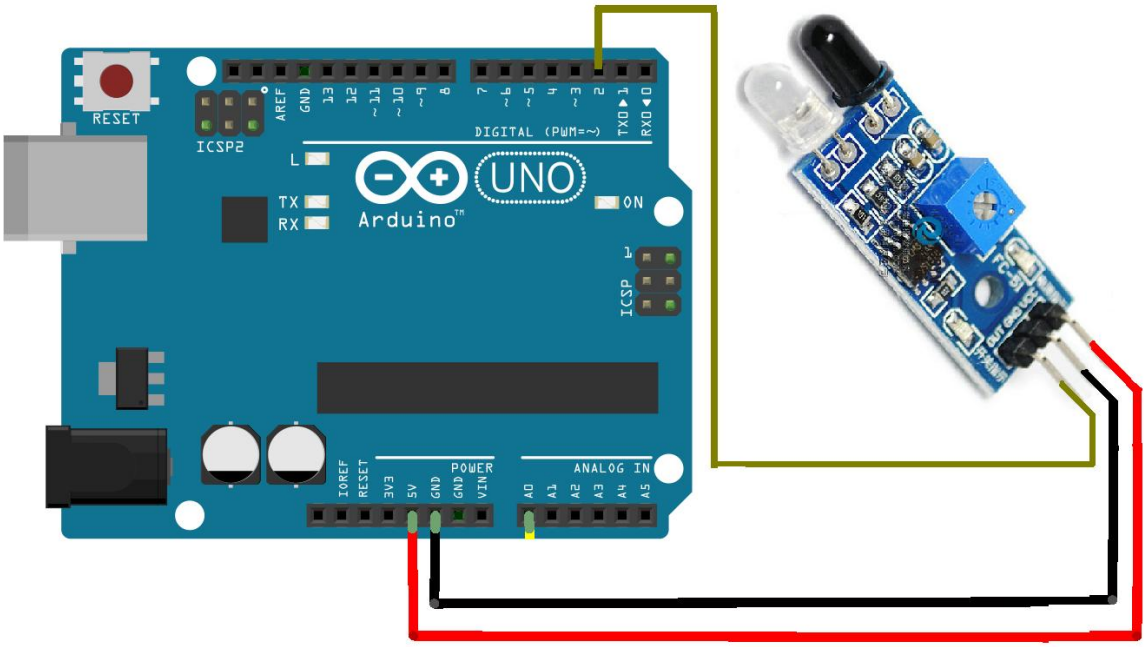

Figure 6.2 Interfacing IR Sensor with Arduino UNO

- IR Sensor has three pins(GND,V,OUT).
- It works under 5v. Connect V pin of sensor with Arduino 5v.
- GND pin of IR connected to GND of Arduino.
- Interface OUT pin of sensor with 2 nd pin of Arduino.

6.3 Interfacing Arduino UNO with DHT11 Humdidity Sensor

DHT11 is an electronic device which is used to measure the temperature and humidity in real environment. It consists of two part capacitive humidity sensor and thermistor. It is a low cost device but provides excellent and precise results. It has an internal small chip used for analog to digital conversions and to provide digital output. We can read this digital output easily through any of the micro-controller. In this tutorial I am using Arduino UNO as a micro-controller. It can be

used in home appliances, weather stations, medical humidity control, data loggers, HVAC and at several different places.

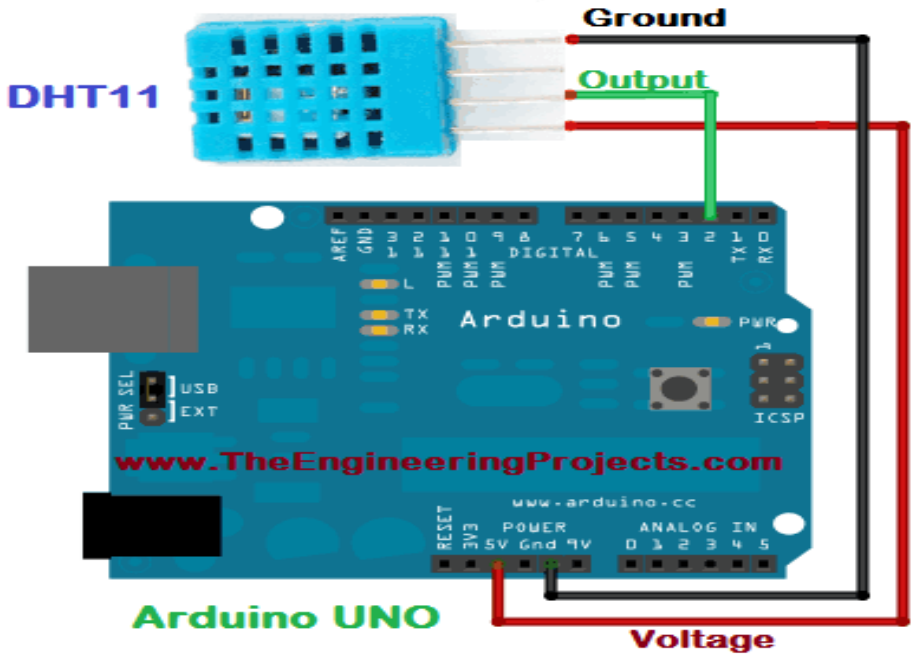

Figure 6.3 Interfacing Arduino UNO with DHT11 Humidity Sensor

6.4 Interfacing Arduino with LM35 Temperature Sensor

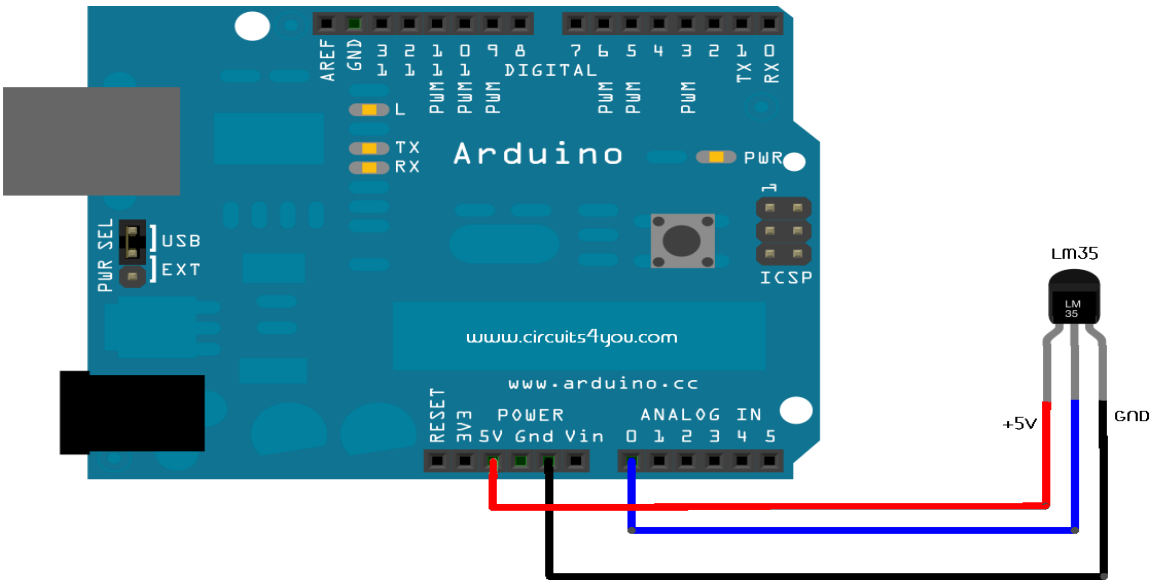

Figure 6.4 Interfacing Arduino UNO with LM35 Temperature Sensor

- It has three pins(Vcc,output,ground).
- It works under 5V.
- Connect V pin of the sensor to Arduino 5V.
- GND pin of the sensor with GND of Arduino.
- Interface OUT pin of sensor with A0 pin of Arduino.

CHAPTER-7

EXECUTABLE CODE

7.1 Transmitter 1

Ecu 1 has a transmission Id of 0x42. Here the temperatures values are obtained by Ecu1, transferred through the can bus and are received by Ecu 3. Thus, the output is displayed on the serial monitor.

```
#include  <mcp_can.h>
#include  <SPI.h>
const int SPI_CS_PIN = 10;
// Build an ID or PGN
long unsigned int txID = 0x42;
unsigned char stmp[8] = {0x1E, 0x00, 0xFF, 0x22, 0xE9, 0xFA, 0xDD, 0x51};
//Construct a MCP_CAN object and set Chip Select to 10.
MCP_CAN  CAN(SPI_CS_PIN);

void setup()
{
   Serial.begin(115200);
   while (CAN_OK !=CAN.begin(CAN_250KBPS))
   // init can bus : baudrate = 250K
   {
      Serial.println("CAN BUS Module Failed to Initialize") ;
      Serial.println("Retrying….");
      delay(200);
   }
```

```
    Serial.println("CAN BUS Shield init ok!");
}

void loop()
{
    Serial.println("In loop");
    int adc;
    adc=analogRead(A5);
    float v;
    v=adc*(5/(1023.00));
    stmp[0]=v*100;
    CAN.sendMsgBuf(txID,0, 8, stmp);
    delay(500);
}
```

7.2 Transmitter 2

Ecu 2 has a transmission Id of 0x43. Here the humidity values, Ir sensor values are obtained by Ecu2, transferred through the can bus and are received by Ecu 3. Thus, the output is displayed on the serial monitor.

```
#include  <mcp_can.h>
#include  <SPI.h>
#include  <dht.h>
dht DHT;
#define DHT11_PIN 7
const int SPI_CS_PIN = 10;
```

```
const int ir_sensor=2;
long unsigned int txID = 0x43;
unsigned char stmp[8] = {0x1E, 0x00, 0xFF, 0x22, 0xE9, 0xFA, 0xDD, 0x51};
//Construct a MCP_CAN object and set Chip Select to 10.
MCP_CAN  CAN(SPI_CS_PIN);

void setup()
{
   Serial.begin(115200);
   pinMode(ir_sensor, INPUT);
   while (CAN_OK !=CAN.begin(CAN_250KBPS))
   // init can bus : baudrate = 250K
   {
        Serial.println("CAN BUS Module Failed to Initialize") ;
        Serial.println("Retrying….");
        delay(200);
   }
   Serial.println("CAN BUS Shield init ok!");
}

void loop()
{
   Serial.println("In loop");
   stmp[0]=digitalRead(ir_sensor);
   int chk = DHT.read11(DHT11_PIN);
   stmp[1]=DHT.humidity;
```

```
   CAN.sendMsgBuf(txID,0, 8, stmp);
   Delay(500);
}
```

7.3 Receiver

The values from different sensors present in both the ECU's are obtained via the Can network and are received by ECU 3. The output is displayed on the Serial monitor.

```
#include  <SPI.h>
#include  "mcp_can.h"
long unsigned int rxId;
unsigned long rcvTime;
unsigned char len=0;
unsigned char buf[8];
const int SPI_CS_PIN = 10;
MCP_CAN  CAN(SPI_CS_PIN);       // Set CS pin


void setup()
{
   Serial.begin(115200);
   while (CAN_OK !=CAN.begin(CAN_250KBPS))
   // init can bus : baudrate = 250K
   {
        Serial.println("CAN BUS Module Failed to Initialize") ;
        Serial.println("Retrying….");
        delay(200);
```

```
    }
        Serial.println("CAN BUS Module Initialized!");
        Serial.println("Time\t\tPGN\t\tByte0\tByte1\tByte2\tByte3\tByte4\tByte5\tB
        yte6\tByte7");
}


void loop()
{
    if(CAN_MSGAVAIL == CAN.checkReceive())     // check if data coming
    {
        rcvTime = millis();
        CAN.readMsgBuf(&len, buf);   //read data, len: data length, buf: data buf
        rxId= CAN.getCanId();
        Serial.print("0x");
        Serial.print(rxId, HEX);
        Serial.print("\t");
        for(int i=0; i<len; i++)   //print the data
        {
            if(buf[i] > 15)
            {
                Serial.print("0x");
                Serial.print(buf[i], HEX);
            }
            else
            {
                Serial.print("0x0");
```

```
            Serial.print(buf[i], HEX);
        }
        Serial.print("\t");
    }
    Serial.println();
    if(rxId==0x42)
    {
        Serial.print("Raw Temperature value is");
        Serial.print(buf[0]);
     }
     Serial.println();
  }
}
```

CHAPTER-8

SIMULATION AND RESULT

8.1 Code Simulation in Arduino

- The Arduino IDE has a built-in feature which is used to compile the code we write.
- The next part is to compile the executable code into the machine language code that the Arduino runs.
- The figures 8.1 and 8.2 shows that the code is compiled successfully.

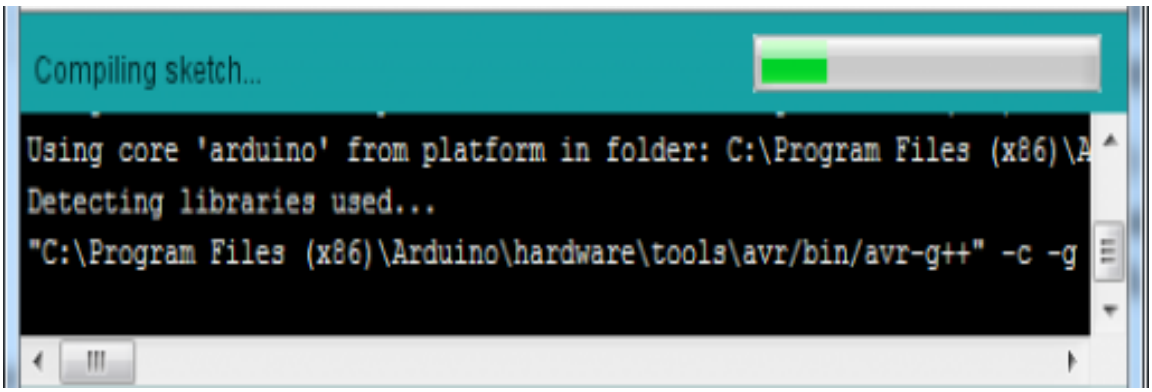

Figure 8.1 Code Compiling Sketch

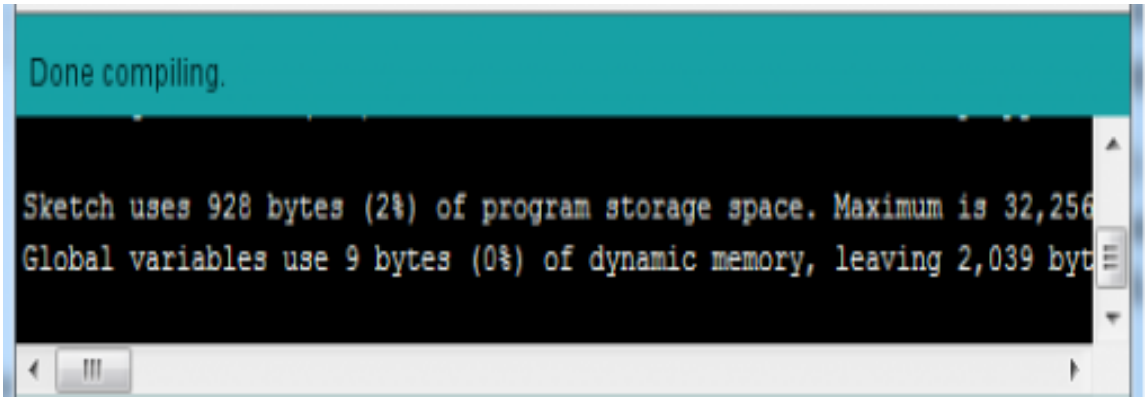

Figure 8.2 Code Compiled Sketch

- The figure 8.3 shows that the code is uploaded successfully.

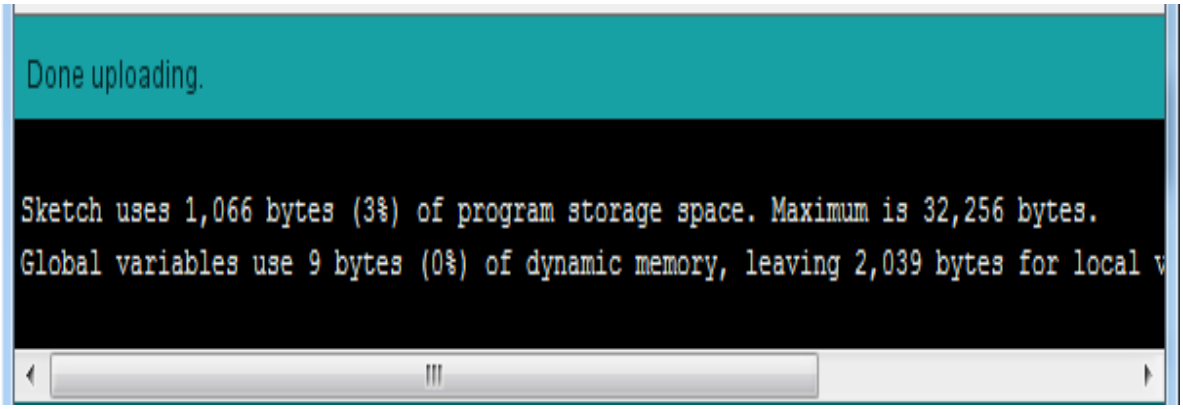

Figure 8.3 Code Uploaded

8.2 Arduino IDE Code

- Transmitter 1


```
#include <mcp_can.h>
#include <SPI.h>


const int SPI_CS_PIN = 10;

// Build an ID or PGN

long unsigned int txID = 0x42;
unsigned char stmp[8] = {0x1E, 0x00, 0xFF, 0x22, 0xE9, 0xFA, 0xDD, 0x51};

//Construct a MCP_CAN Object and set Chip Select to 10.

MCP_CAN CAN(SPI_CS_PIN);


void setup()
{
    Serial.begin(115200);

    while (CAN_OK != CAN.begin(CAN_250KBPS))              // init can bus : baudrate = 250K
    {
         Serial.println("CAN BUS Module Failed to Initialized");
        Serial.println("Retrying....");
        delay(200);

    }
    Serial.println("CAN BUS Shield init ok!");
}
```

```
void loop()
{
  Serial.println("In loop");
  int adc;
  adc=analogRead(A5);
  float v;
  v=adc*(5/(1023.00));
  stmp[0]=v*100;

    CAN.sendMsgBuf(txID,0, 8, stmp);
    delay(500);
}
```

Figure 8.4 Arduino IDE Transmitter 1 code

- Transmitter 2

File Edit Sketch Tools Help

irt §

```
#include <mcp_can.h>
#include <SPI.h>
#include <dht.h>
dht DHT;
#define DHT11_PIN 7
const int SPI_CS_PIN = 10;
const int ir_sensor=2;
long unsigned int txID = 0x43;
unsigned char stmp[8] = {0x1E, 0x00, 0xFF, 0x22, 0xE9, 0xFA, 0xDD, 0x51};

//Construct a MCP_CAN Object and set Chip Select to 10.

MCP_CAN CAN(SPI_CS_PIN);


void setup()
{
    Serial.begin(115200);
    pinMode(ir_sensor,INPUT);
    while (CAN_OK != CAN.begin(CAN_250KBPS))              // init can bus : baudrate = 250K
    {
         Serial.println("CAN BUS Module Failed to Initialized");
        Serial.println("Retrying....");
        delay(200);

    }
    Serial.println("CAN BUS Shield init ok!");
}
```

```
void loop()
{
  Serial.println("In loop");
  stmp[0]=digitalRead(ir_sensor);
  //Serial.print(stmp[0]);
  int chk = DHT.read11(DHT11_PIN);
  stmp[1]=DHT.humidity;
  //Serial.print(stmp[1]);

    CAN.sendMsgBuf(txID,0, 8, stmp);
    delay(500);
}
```

Figure 8.5 Arduino IDE Transmiter 2 code

- Receiver

```
    Serial.println("Time\t\tPGN\t\tByte0\tByte1\tByte2\tByte3\tByte4\tByte5\tByte6\tByte7");
}


void loop()
{

    if(CAN_MSGAVAIL == CAN.checkReceive())            // check if data coming
    {

      //Serial.println(" In loop");
        rcvTime = millis();
        CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf: data buf

        rxId= CAN.getCanId();

       // Serial.print(rcvTime);
        //Serial.print("\t\t");
        Serial.print("0x");
        Serial.print(rxId, HEX);
        Serial.print("\t");

        for(int i = 0; i<len; i++)    // print the data
        {
            if(buf[i] > 15){
              Serial.print("0x");
              Serial.print(buf[i], HEX);
            }

          else{
              Serial.print("0x0");
              Serial.print(buf[i], HEX);
          }

            //Serial.print("0x");
            //Serial.print(buf[i], HEX);

            Serial.print("\t");



        }
        Serial.println();
        if(rxId==0x42)
                      {
                         Serial.print("Raw Temperature value is");
                         Serial.print(buf[0]);
                      }
                       Serial.println();
    }
}
```

Figure 8.6 Arduino IDE Receiver code

8.3 Results on Breadboard

- The following pictures shows the connections for our project on a breadboard. They include both Transmitter and Receiver.

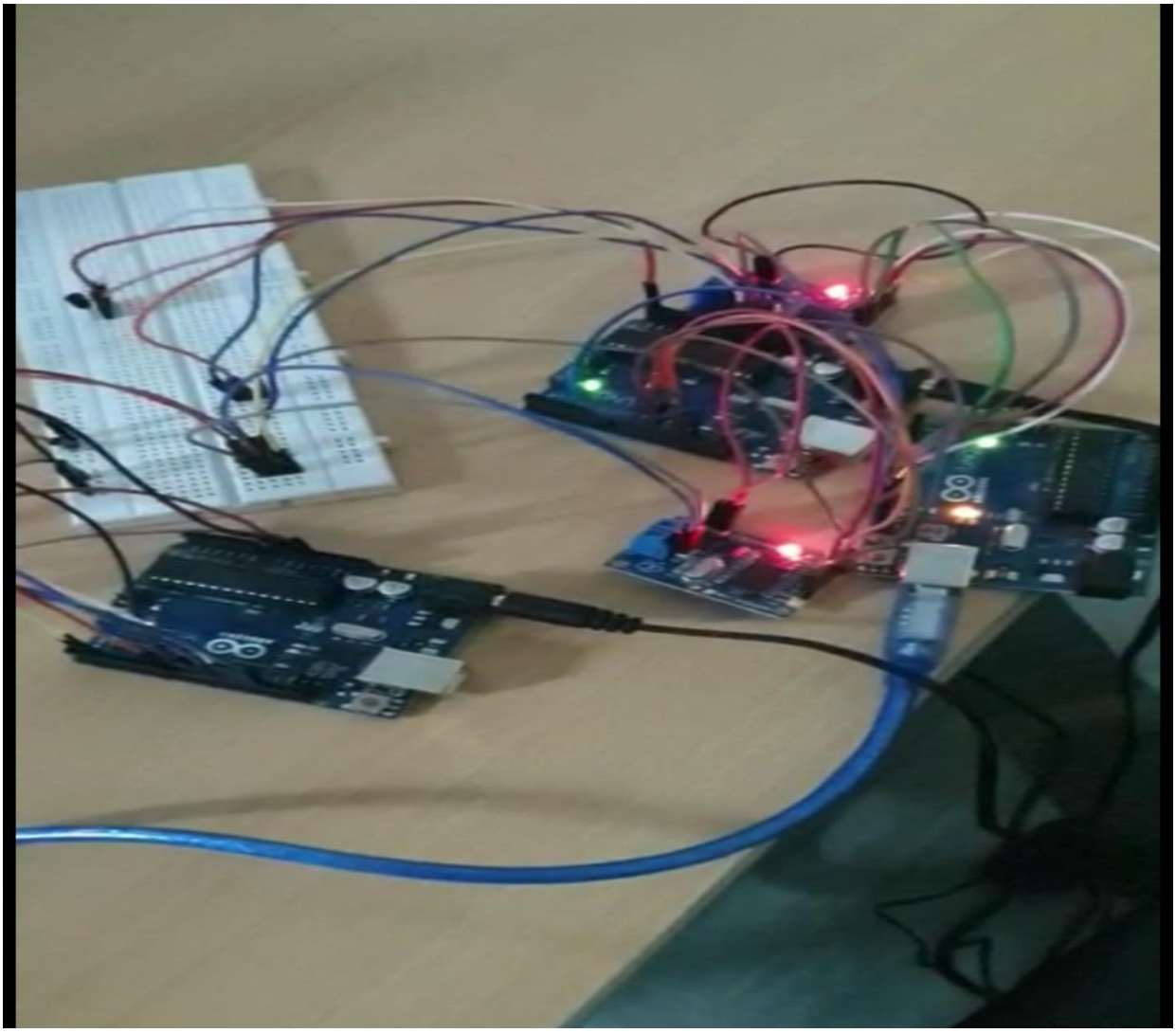

Figure 8.7a Breadboard implementation of circuit

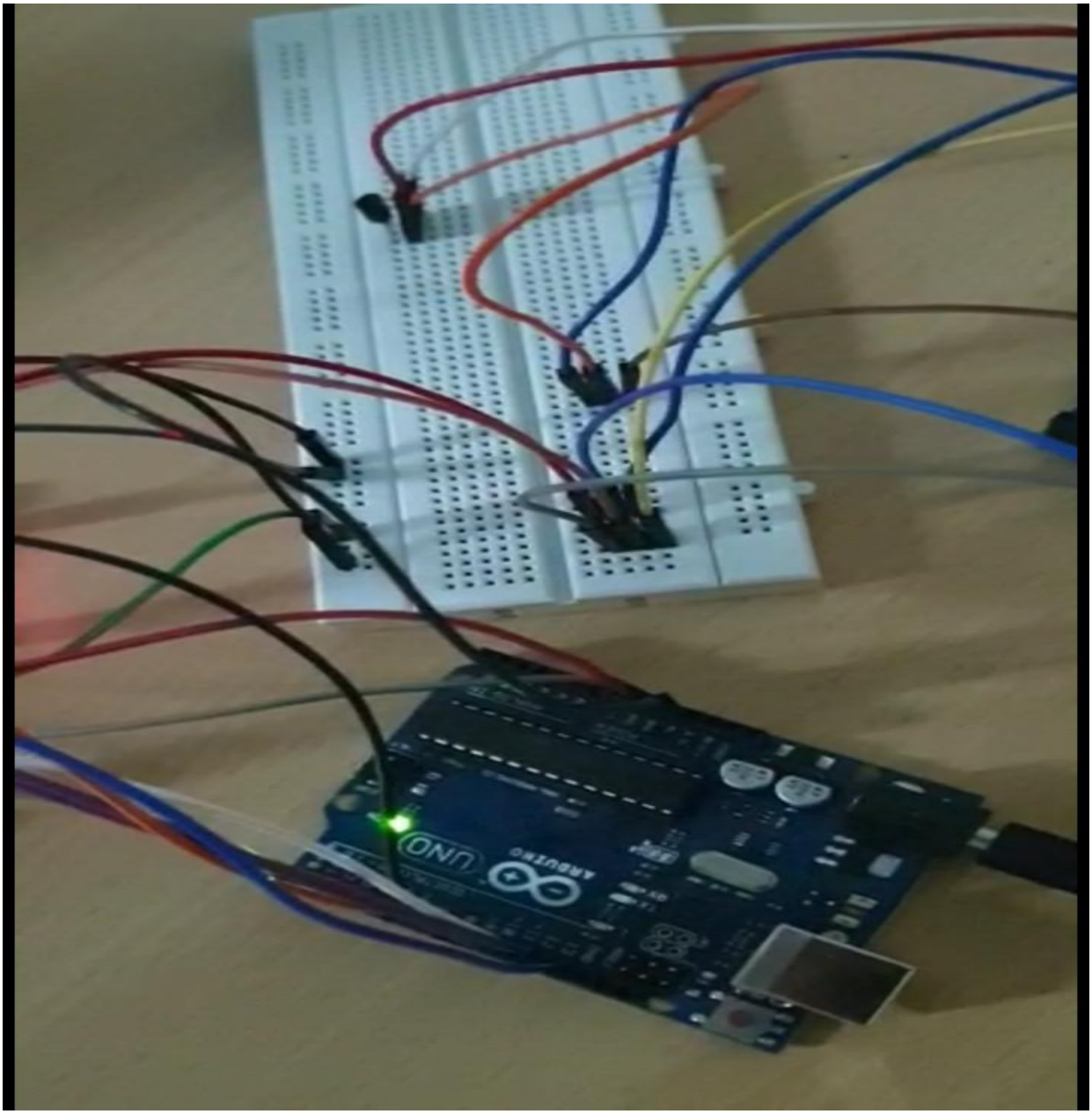

Figure 8.7b Breadboard implementation of circuit

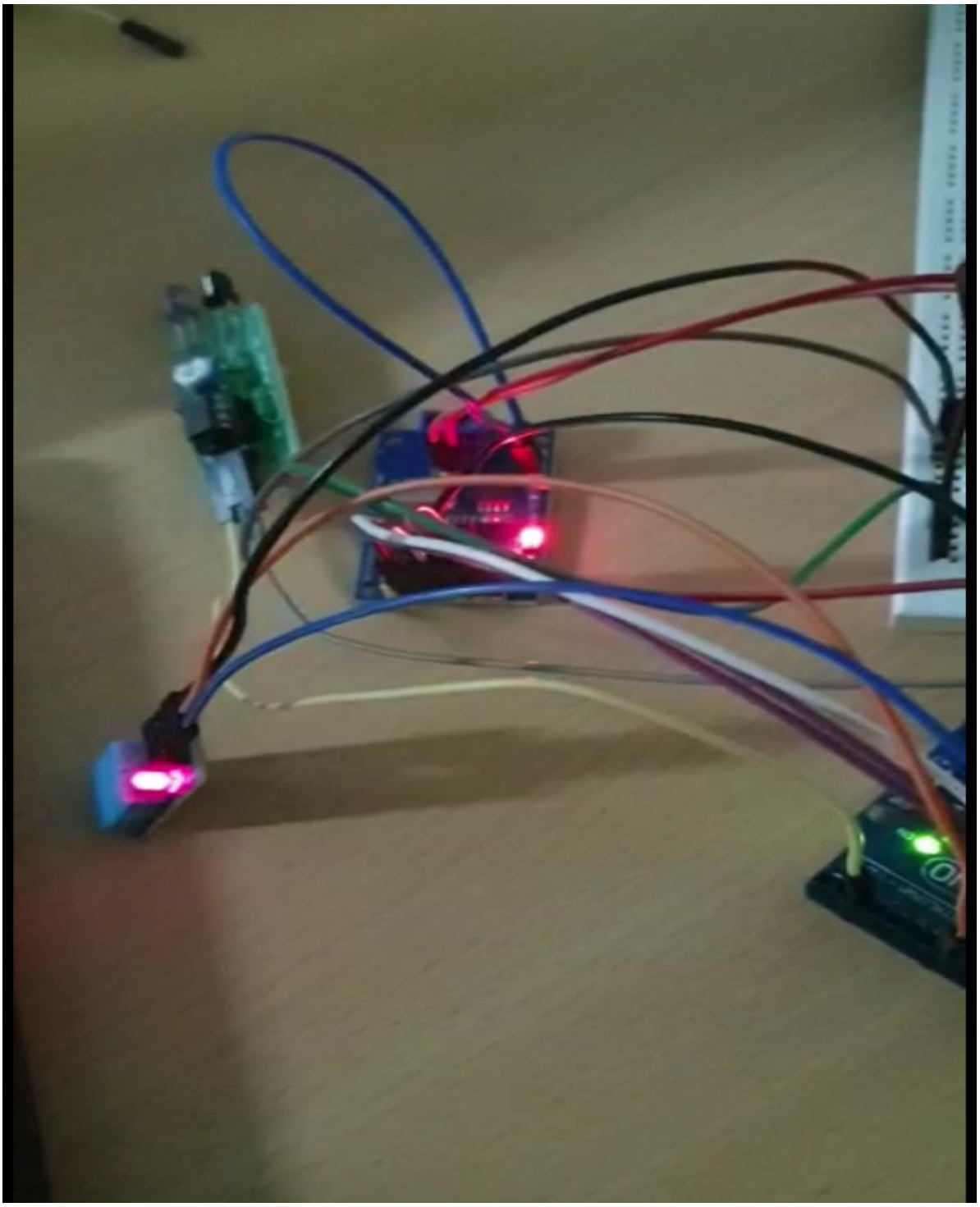

Figure 8.7c Breadboard implementation of circuit

8.4 Output

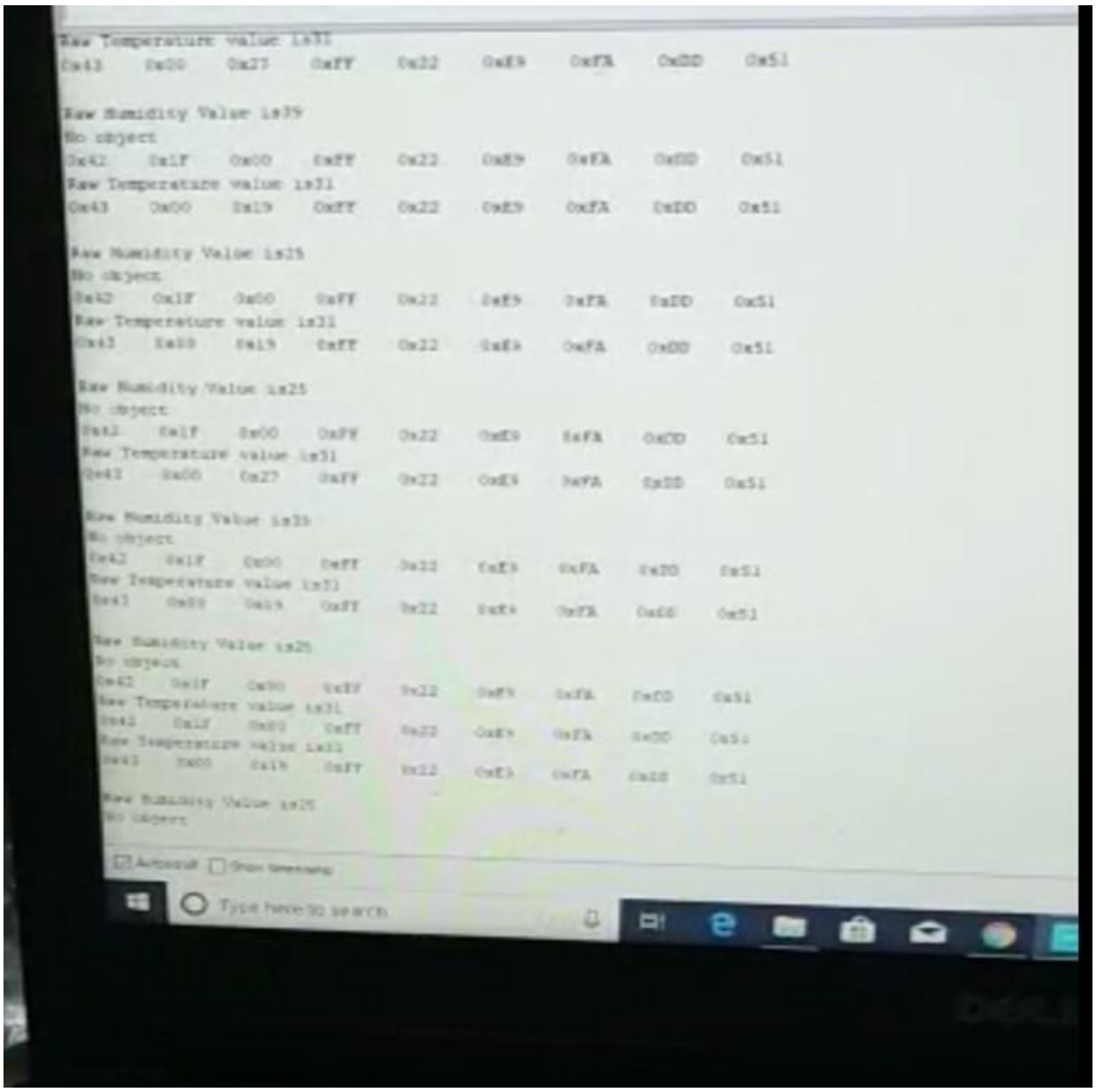

Figure 8.8 Readings on Serial Monitor

CHAPTER-9

CONCLUSION

The project deals with the data transmission between two ECUs using CAN protocol.The main basic concept of this technique is to reduce the testing cost of ECUs. In this project we present the system with CAN protocol to monitor and control and analysis the problems in the Automotive or Automobile application. The parameters are measured through the CAN interface module. The CAN protocol is used for serial communication which provides high data transmission rate and reliability.

CHAPTER-10

REFERENCES

- https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/
- http://henrysbench.capnfatz.com/henrys-bench/arduino-projects-tips-and-more/arduino-can-bus-module-1st-network-tutorial/
- http://troindia.in/journal/ijcesr/vol2iss7/114-118.pdf
- https://www.theengineeringprojects.com/2018/10/introduction-to-arduino-ide.html
- https://elementztechblog.wordpress.com/2017/12/07/interfacing-ir-sensor-with-arduino-2/
- https://www.instructables.com/id/How-to-interface-Humidity-and-Temperature-DTH11-Se/
- https://elementztechblog.wordpress.com/2017/12/07/arduino-and-temperature-sensorlm35-interfacing-how/