FORMULA STUDENT ELECTRONICS & DASHBOARD DESIGN

By

Muhammad Ahmed Amaan	**NUST 2012 00583 BSMME 11112F**
Usama Masud	**NUST 2012 01154 BSMME 11112F**
Shahzeb Khan	**NUST 2012 01259 BSMME 11112F**

Supervised By

Dr. Samiur Rahman Shah

School of Mechanical and Manufacturing Engineering,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan

June, 2016

FORMULA STUDENT ELECTRONICS & DASHBOARD DESIGN

By

Muhammad Ahmed Amaan	**NUST 2012 00583 BSMME 11112F**
Usama Masud	**NUST 2012 01154 BSMME 11112F**
Shahzeb Khan	**NUST 2012 01259 BSMME 11112F**

Supervised By

Dr. Samiur Rahman Shah

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelors of Engineering in Mechanical Engineering

School of Mechanical and Manufacturing Engineering,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan

June, 2016

National University of Sciences & Technology

FINAL YEAR PROJECT REPORT

We hereby recommend that the dissertation prepared under our supervision by: Muhammad Ahmed Amaan (NUST201200583BSMME11112F), Usama Masud (NUST201201154BSMME11112F) and Shahzeb Khan (NUST201201259BSMME11112F) Titled: **Formula Student Electronics & Dashboard Design** be accepted in partial fulfillment of the requirements for the award of Bachelors of Engineering in Mechanical Engineering degree with (A grade).

English and format checked by Ms Aamna Hassan, Signature: _____

Guidance Committee Members

1. Name:_____ Signature:_____

2. Name:_____ Signature:_____

3. Name:_____ Signature:_____

Supervisor's Name:_____ Signature:_____

_____ _____

Head of Department Date

COUNTERSIGNED

_____ _____

Date Dean/Principal

DECLARATION

We certify that this research work titled "*FORMULA STUDENT ELECTRONICS & DASHBOARD DESIGN"* is our own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student
Muhammad Ahmed Amaan
NUST201200583BSMME11112F

Signature of Student
Usama Masud
NUST201201154BSMME11112F

Signature of Student
Shahzeb Khan
NUST201201259BSMME11112F

COPYRIGHTS STATEMENT

Dedicated to my parents

ABSTRACT

Team Bolts aim to participate in this year's Formula Student and for that reason, the project undertaken by team Bolts Destiny was to design dashboard and electronics of Formula Student Car, that would be efficient and serves its purpose to the fullest.

The work started with a detailed literature review of many previous teams and how they achieved their simple goals. Many research papers and articles were thoroughly studied. Once that was done, CAD models were prepared for the dashboard and electronic circuit layouts along with the backend coding done in Arduino.

Once all the layouts and electronic components were complete, the team entered the fabrication phase. Each part was fabricated according to the required electronic and mechanical specifications and the whole system was then assembled together in the chassis.

The findings of the project can be divided into groups: mechanical and electrical. In the electronics part, the team studied coding and data sheets used for every single electronics component. The final system that was fabricated was the simplest system that could have been prepared. A few problems the team encountered included component availability, complex codes and telemetry, used to interface Accelerometer with an application on mobile phone using Bluetooth connectivity.

PREFACE

Formula Student is an international event that takes place around the globe numerous times every year. The competition comprises of teams from all over the globe who build their own Formula style racing car from scratch and during the event, compete in a series of events. The events can be divided into types: Static and Dynamic. Static events include exhibition of all the features, design, ergonomics and functions of each vehicle and all of these factors are judged by a group of expert judges from Formula Student.

Team Bolts is part of NUST Bolts Racing and its aim is to participate in Formula Student UK in July 2016. The team comprises of 10 members, each responsible for a specific system of the race car. Team Bolts Destiny was made responsible for the Dashboard Design, Data Logging and the electronics of the race car. That became the reason Team Bolts Destiny chose this as their Final Year Project, so that they could achieve two targets with one project.

This document contains all the literature review work, methodology, findings and drawings of the work done by Team Bolts Destiny in order to completely manufacture their own FS race car. The team started their work on this project in September 2015. The team consists of Muhammad Ahmed Amaan, Usama Masud and Shahzeb Khan.

The aim to take part in FS UK is dependent on the success of this project and Team Bolts Destiny hopes that their work takes them on the road to success.

ACKNOWLEDGEMENTS

Team Bolts Destiny took efforts in this project. However, it would not have been possible without the kind support and help of various individuals and organizations. Team Bolts Destiny would like to extend their sincere thanks to all of them.

The team is highly indebted to Dr. Samiur Rahman Shah for his guidance and constant supervision as well as providing necessary information regarding the project and also for his support and trust in completing the project. The team is also thankful to the parents of the individual members for their utmost support and motivation.

A special gratitude goes out to the staff at SMME and MRC for their assistance in completing the whole project.

Table of Contents

ELECTRONICS

1.1 BACKGROUND

In the dashboard head, there were four 16x2 LCDs used to display different outputs which are as follows:

1- Speed
2- Rpm
3- Fuel Gauge
4- Temperature Gauge

1.2 TEMPERATURE SENSOR BARGRAPH CODE

I. Pin Layout:

Following figure shows the pin layout for 16x2 LCD:

```
/*
lcd pins
GND    GND
Vcc    5V
V0     potentiometer mid pin
RS     12
R/W    GND
E      11
D0
D1
D2
D3
D4     5
D5     4
D6     3
D7     2
A      5V
K      GND
LM35:
1   5V
2   A0
3   GND
*/
```

Whereas the LCD is as shown below:

II. LCD Declaration in the code:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);

//Creating Progress Bar Characters

byte p20[8]={
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
};

byte p40[8]={
  B11000,
  B11000,
  B11000,
  B11000,
  B11000,
```

Steps involved are:

a) Initialization of the lcd library :

```
#include <LiquidCrystal.h>
```

b) Then declaring the pins through which data transfer will occour between the microcontroller and the LCD

```
LiquidCrystal lcd(12,11,5,4,3,2);
```

c) Creating Progress Bar Characters:

```
//Creating Progress Bar Characters

byte p20[8]={
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
};
```

This array makes every dot in the small dot matrix of a digit in 16x2 LCD to turn on.

III. Void Setup ( )

This is the part of the program gets initialized when the microcontroller starts operation or when its operation is reset.

```
void setup()
{
  lcd.begin(16,2);
  Serial.begin(9600);
  lcd.createChar(0,p20);
  lcd.createChar(1,p40);
  lcd.createChar(2,p60);
  lcd.createChar(3,p80);
  lcd.createChar(4,p100);
}
```

a) Declaring the type of LCD:

lcd.begin(16,2);

This is the definition that the LCD being used is 16x2

b) Serial library declaration :

Serial.begin(9600);

The serial library shows the data on the serial port

IV. Void Loop ( )

This part of the loop gets repeated again and again during the operation of the microcontroller and whatever is written in it is the main operational code of the microcontroller.

```
void loop ()
{


 int  val = analogRead(A0);
float mv = ( val/1024.0)*5000;
float cel = mv/10;
int a=cel/60*16;
  // print out the value you read:
  Serial.println(a);
  delay(1);
    for(int i=0; i<a; i++)
      {
        for(int j=0; j<5; j++)
          {
            lcd.setCursor(i,0);
           lcd.write(j);
          lcd.setCursor(i,1);
          lcd.write(j);
          delay(10);
          }

      }
lcd.clear();
delay(20);
  }
```

V. Bargraph Code Explained

a) Initialization of variables :

The variables are being initialized at first is an integer type variable and the variable name is **val** and its value is coming from input pin A0.

```
int  val = analogRead(A0);
```

Next variable's name is mv and it is a float type variable and its value is a factor multiplied with the variable **val.**

```
float mv = ( val/1024.0)*5000;
```

Next is a float type variable named **cel** and it is the temperature in Celsius and it comes by dividing the mv by 10.

```
float cel = mv/10;
```

Next variable is the scaling variable that will determine how many of the 16 bargraph levels should be turned on. It is an integer type variable named **a** and it is just a scaling of **cel** variable on 100 Celsius i.e. at 100 Celsius the value of a will be 16 so full graph will be turned on

```
int a=cel/60*16;
```

b) Printing the value of a on serial monitor :

Next command will display the value of a on serial monitor

```
Serial.println(a);
```

c) Delay Function:

Next command is a delay function that puts a delay in the operation of microcontroller and the function value is the number of mills it will take delay

```
delay(1);
```

d) Bargraph Displaying Loop :

This loop will print the bar graph on the LCD according to the value of variable **a**, the value of variable **a** ranges from 0 to 16 and whatever the value of variable **a** would be this loop will turn on respective amount of blocks in the bar graph:

```
for(int i=0; i<a; i++)
  {
    for(int j=0; j<5; j++)
      {
        lcd.setCursor(i,0);
       lcd.write(j);
      lcd.setCursor(i,1);
      lcd.write(j);
      delay(10);
      }

  }
```

The outer loop runs as many times as the value of variable **a** and every time it runs it turns one block in the bar graph.

This outer for loop runs as many times as the value of variable a

```
for(int i=0; i<a; i++)
```

The inner loop will display a block of bargraph whenever it is run.

and it will run 5 times because a block of the bargraph comprises of small 5 rows of dot matrix and this loop will turn all of them on to make the block fully turned on.

```
for(int  j=0;  j<5;  j++)
```

The loop will run 5 times to turn one block on because 16x2 has 2 rows so first it will display half of the block then the other half.

```
lcd.setCursor(i,0);
lcd.write(j);
```

The first line will take the cursor to first row of the LCD and the write command will print the block.

```
lcd.setCursor(i,1);
lcd.write(j);
```

This fist line will take the code to the 2nd row in the LCD and the write command will print the block.

delay(10);
Then there will be 10 milliseconds delay.

VI. End of Code

Before the void loop runs again the screen must be cleared to display the bargraph again.

```
lcd.clear();
delay(20);
  }
```

lcd.clear();
delay(20);

1.3 FUEL GAUGE BARGRAPH CODE

I. Pin Layout

Following is the illustration of the LCD Pin Layout:

```
/*
lcd pins
GND     GND
Vcc     5V
V0      potentiometer mid pin
RS      12
R/W     GND
E       11
D0
D1
D2
D3
D4      5
D5      4
D6      3
D7      2
A       5V
K       GND
LM35:
1    5V
2    A0
3    GND
*/
```

Whereas the LCD is as shown below:

II. LCD Declaration in the code

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);

//Creating Progress Bar Characters

byte p20[8]={
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
};

byte p40[8]={
  B11000,
  B11000,
  B11000,
  B11000,
  B11000,
```

a) Initialization of the LCD library:

#include <LiquidCrystal.h>

b) Then Declaring the pins through which data transfer will occour between the microcontroller and the LCD

LiquidCrystal lcd(12,11,5,4,3,2);

c) Creating Progress Bar Characters:

```
//Creating Progress Bar Characters

byte p20[8]={
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
};
```

This array makes every dot in the small dot matrix of a digit in 16x2 LCD to turn it on.

III. Void Setup ( )

This is the part of the program that gets initialized when the microcontroller starts operation or when its operation is reset.

```
void setup()
{
  lcd.begin(16,2);
  Serial.begin(9600);
  lcd.createChar(0,p20);
  lcd.createChar(1,p40);
  lcd.createChar(2,p60);
  lcd.createChar(3,p80);
  lcd.createChar(4,p100);
}
```

a) Declaring the type of LCD:

lcd.begin(16,2);

This is the definition that the LCD being used is 16x2.

b) Serial library declaration:

Serial.begin(9600);
The serial library shows the data on the serial port.

IV. Void Loop ( )

This part of the loop gets repeated again and again during the operation of the microcontroller and whatever is written in it is the main operational code of the microcontroller.

```
void loop ()
{


int sensorValue = analogRead(A0);
  // print out the value you read:
  int a =sensorValue*300/1000;

    for(int i=0; i<a; i++)
      {
        for(int j=0; j<5; j++)
          {
            lcd.setCursor(i,0);
           lcd.write(j);
          lcd.setCursor(i,1);
          lcd.write(j);
          delay(10);
          }
         int b=a;
      }
```

V. Void Loop ( )

a) Initialization of variables :

The variables are being initialized at first is an integer type variable and the variable name is **sensorValue** and its value is coming from input pin A0.

```
int  sensorValue = analogRead(A0);
```

Then this sensor value is multiplied with a factor to get the value of variable **a**.

```
int a =sensorValue*300/1000;
```

b) Bargraph Displaying Loop:

This loop will print the bar graph on the LCD according to the value of variable **a**, the value of variable a ranges from 0 to 16 and whatever the value of variable a would

be this loop will turn on that many blocks in the bar graph.

```
for(int i=0; i<a; i++)
  {
    for(int j=0; j<5; j++)
      {
        lcd.setCursor(i,0);
       lcd.write(j);
      lcd.setCursor(i,1);
      lcd.write(j);
      delay(10);
      }

  }
```

The outer loop runs as many times as the value of variable **a** and every time it runs it turns one block in the bar graph.

This outer for loop runs as many times as the value of variable **a**.

for(int i=0; i<a; i++)

The inner loop will display a block of bargraph whenever it is run.

And it will run 5 times because a block of the bargraph comprises of small 5 rows of dot matrix and this loop will turn all of them on to make the block fully turned on

for(int j=0; j<5; j++)

The loop will run 5 times to turn one block on

Because 16x2 has 2 rows so first it will display half of the block then the other half

lcd.setCursor(i,0);
lcd.write(j);

The first line will take the cursor to first row of the lcd and the write command will print the block

lcd.setCursor(i,1);
lcd.write(j);

This fist line will take the code to the 2nd row in the lcd and the write command will print the block

delay(10);

There will be a delay of 10 milliseconds.

VI. End of the Code

Before the void loop runs again the screen must be cleared to display the bargraph again.

```
lcd.clear();
delay(20);
  }
```

The **lcd.clear()** command clears the screen

lcd.clear();

The delay function puts 20 mills delay

delay(20);

And this is the end of the code.

1.4 REV COUNTER / TACHOMETER

I. Pin Layout

Following is the illustration of the LCD Pin Layout:

```
/*
  The circuit:
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * LCD VSS pin to ground
 * LCD VCC pin to 5V
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD V0 pin (pin 3)

 Library originally added 18 Apr 2008
 by David A. Mellis
 library modified 5 Jul 2009
 by Limor Fried (http://www.ladyada.net)
 example added 9 Jul 2009
 by Tom Igoe
 modified 22 Nov 2010
 by Tom Igoe

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/LiquidCrystal
 */
```

II. LCD Declaration in the Code

```
 http://www.arduino.cc/en/Tutorial/LiquidCrystal
 */

// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 6, 5, 4,3);
```

(a) Initialization of the lcd library.

#include <LiquidCrystal.h>

(b) Then Declaring the pins through which data transfer will occur between the microcontroller and the LCD

LiquidCrystal lcd(12,11,6,5,4,3);

III. Assigning Variables

```
byte freqpin = 2;

unsigned long timeone;

unsigned int frequency;
unsigned int timediff;

unsigned int tcount=0;

// sample time of 1 second or 1000 ms.
unsigned int sampletime=1000;
unsigned long sampletimetwo=1000000;
unsigned long j;

volatile unsigned int count=0;
```

byte freqpin = 2;

A byte stores an 8-bit unsigned number, from 0 to 255. Here the variable **freqpin** is given the value 2 and is a byte type variable.

unsigned long timeone;

Unsigned long variables are extended size variables for number storage, and store 32 bits (4 bytes). And here a variable **timeone** is assigned as unsigned long.

unsigned int frequency;

On the Uno and other ATMEGA based boards, unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). Here the variable **frequency** is declared as an unsigned int variable.

unsigned int tcount=0;

On the Uno and other ATMEGA based boards, unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). Here the variable **tcount** is declared as an unsigned int variable.

unsigned int timediff;

On the Uno and other ATMEGA based boards, unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). Here the variable **timediff** is declared as an unsigned int variable.

unsigned int sampletime=1000;

On the Uno and other ATMEGA based boards, unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). Here the variable **sampletime** is declared as an unsigned int variable.

IV. Void Setup ( )

```
void setup() {
  pinMode(freqpin, INPUT);
// set interrupt pin 2 and use rising ( from gnd to +5 )
attachInterrupt(0,freqinput,RISING);
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
   Serial.begin(9600);
  // Print a message to the LCD.

}
```

pinMode(freqpin, INPUT);

This sets the interrupt on the pin 2 so that the **Hall Effect sensor** will be attached to it.

attachInterrupt(0,freqinput,RISING);

Set up the LCD's number of columns and rows for the display.

Declaring the type of LCD:

lcd.begin(16,2);

this is the definition that the lcd being used is 16x2

Serial library declaration:

Serial.begin(9600);

The serial library shows the data on the serial port.

V. Void Setup ( )

This part of the loop gets repeated again and again during the operation of the microcontroller and whatever is written in it is the main operational code of the Microcontroller.

```
void loop()
{
/* Use while() loop and use millis().
    Calculate the time difference and compare the
    VALUE with the sample time.
*/
tcount=0;
count=0;
// activate the interrupt
timediff=0;
interrupts();
timeone=millis();
while(timediff<sampletime)
{
  tcount=count;
  timediff=millis()-timeone;
}
// de-activated the interrupt
noInterrupts();
// Calculated frequency
frequency=tcount;
```

Use while() loop and millis(). Calculate the time difference and compare the value with the sample time. So while loop and mills function are used to calculate the time in milliseconds of a frequency pulse.

a) Activation and Deactivation of interrupts:

```
// activate the interrupt
timediff=0;
interrupts();
timeone=millis();
while(timediff<sampletime)
{
  tcount=count;
  timediff=millis()-timeone;
}
// de-activated the interrupt
noInterrupts();
// Calculated frequency
frequency=tcount;
```

The interrupt function is activated and then deactivated to count the milliseconds for which the pulse lasted.

b) Displaying the frequency:

```
// Display frequency
Serial.print("The time dealy is : ");
Serial.println(timediff, DEC);
Serial.print("Test while - ");
Serial.print("Frequency : ");
Serial.println(frequency,DEC);
int rpm;
rpm=frequency*60;
Serial.println(rpm);
delay(30); // <-- Not being executed
```

The text and the frequency values were displayed on the serial monitor

```
int rpm;
rpm=frequency*60;
Serial.println(rpm);
delay(30);
```

The rpm is 60 times the frequency and a delay of 30 mills is put for smoother operation.

c) Displaying the rpm on the LCD:

```
lcd.setCursor(7,0);
 lcd.print("rpm");
Serial.println(" ");
rpm =frequency*60;

lcd.setCursor(8,1);
 lcd.print(rpm);

Serial.println(rpm);

delay(30);  // <-- Not being executed
```

lcd.setCursor(7,0);
The cursor is set on position 7,0 which means first row and 7th column on 16x2 LCD.

lcd.print("rpm");
The word rpm is printed at that location.

Serial.println(" ");
rpm =frequency*60;

lcd.setCursor(8,1);
The cursor is set on position 8,1 on the LCD

lcd.print(rpm);
The value of rpm is printed

Serial.println(rpm);
This value is also printed on the serial monitor.

delay(30);
Delay of 30 mills is taken.

This is the end of the void loop.

d) Interrupt routine:

```
// The interrupt routine
void freqinput ()
{
  count++;
}
```

The interrupt routine is a functions that works as a counter.

1.5 SPEEDOMETER

I. Pin Layout

The pin layout is as follows:

```
/*
 The circuit:
* LCD RS pin to digital pin 12
* LCD Enable pin to digital pin 11
* LCD D4 pin to digital pin 5
* LCD D5 pin to digital pin 4
* LCD D6 pin to digital pin 3
* LCD D7 pin to digital pin 2
* LCD R/W pin to ground
* LCD VSS pin to ground
* LCD VCC pin to 5V
* 10K resistor:
* ends to +5V and ground
* wiper to LCD V0 pin (pin 3)

Library originally added 18 Apr 2008
by David A. Mellis
library modified 5 Jul 2009
by Limor Fried (http://www.ladyada.net)
example added 9 Jul 2009
by Tom Igoe
modified 22 Nov 2010
by Tom Igoe

This example code is in the public domain.

http://www.arduino.cc/en/Tutorial/LiquidCrystal
*/
```

II. LCD Declaration in the Code

```
 http://www.arduino.cc/en/Tutorial/LiquidCrystal
 */

// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 6, 5, 4,3);
```

a) Initialization of the lcd library :

#include <LiquidCrystal.h>

b) Then Declaring the pins through which data transfer will occour between the microcontroller and the LCD

LiquidCrystal lcd(12,11,6,5,4,3);

II. Assigning Variables

```
byte freqpin = 2;

unsigned long timeone;

unsigned int frequency;
unsigned int timediff;

unsigned int tcount=0;

// sample time of 1 second or 1000 ms.
unsigned int sampletime=1000;
unsigned long sampletimetwo=1000000;
unsigned long j;

volatile unsigned int count=0;
```

byte freqpin = 2;
A byte stores an 8-bit unsigned number, from 0 to 255. Here the variable **freqpin** is given the value 2 and is a byte type variable

unsigned long timeone;

Unsigned long variables are extended size variables for number storage, and store 32 bits (4 bytes). And here a variable **timeone** is assigned as unsigned long.

unsigned int frequency;

On the Uno and other ATMEGA based boards, unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). Here the variable **frequency** is declared as an unsigned int variable.

unsigned int tcount=0;

On the Uno and other ATMEGA based boards, unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). Here the variable **tcount** is declared as an unsigned int variable.

unsigned int timediff;

On the Uno and other ATMEGA based boards, unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). Here the variable **timediff** is declared as an unsigned int variable.

unsigned int sampletime=1000;

On the Uno and other ATMEGA based boards, unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). Here the variable **sampletime** is declared as an unsigned int variable.

III. Void Setup ( )

```
void setup() {
  pinMode(freqpin, INPUT);
// set interrupt pin 2 and use rising ( from gnd to +5 )
attachInterrupt(0,freqinput,RISING);
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
   Serial.begin(9600);
  // Print a message to the LCD.

}
```

pinMode(freqpin, INPUT);

This sets the interrupt on the pin 2 so that the **Hall Effect sensor** will be attached to it.

attachInterrupt(0,freqinput,RISING);

Set up the LCD's number of columns and rows for the display.

Declaring the type of lcd:

lcd.begin(16,2);

This is the definition that the LCD being used is 16x2

Serial library declaration :

Serial.begin(9600);

The serial library shows the data on the serial port.

IV. Void Loop ( )

This part of the loop gets repeated again and again during the operation of the microcontroller and whatever is written in it is the main operational code of the Microcontroller.

```
void loop()
{
/* Use while() loop and use millis().
    Calculate the time difference and compare the
    VALUE with the sample time.
*/
tcount=0;
count=0;
// activate the interrupt
timediff=0;
interrupts();
timeone=millis();
while(timediff<sampletime)
{
  tcount=count;
  timediff=millis()-timeone;
}
// de-activated the interrupt
noInterrupts();
// Calculated frequency
frequency=tcount;
```

\

Use while() loop and millis(). Calculate the time difference and compare the value with the sample time. So while loop and mills function are used to calculate the time in milliseconds of a frequency pulse.

a) Activation and Deactivation of interrupts:

```
// activate the interrupt
timediff=0;
interrupts();
timeone=millis();
while(timediff<sampletime)
{
  tcount=count;
  timediff=millis()-timeone;
}
// de-activated the interrupt
noInterrupts();
// Calculated frequency
frequency=tcount;
```

The interrupt function is activated and then deactivated to count the milliseconds for which the pulse lasted.

b) Lcd Display:

```
lcd.setCursor(6,0);
 lcd.print("speed");
Serial.println(" ");
rpm =frequency*60*2*3.14/60*4.14*0.3;

lcd.setCursor(8,1);
 lcd.print(rpm);

Serial.println(rpm);

delay(30);  // <-- Not being executed

}
```

lcd.setCursor(6,0);

Sets the cursor on column 6 and row 0 of 16x2 LCD.

lcd.print("speed");
Prints the word speed.

Serial.println(" ");
Gives a space.

rpm =frequency*60*2*3.14/60*4.14*0.3;

Multiplies the frequency with 60 to get rpm then multiplies with final drive ratio 4.14, 2 and pi followed by dividing it by 60 and multiplying with the radius of the tire to get speed of vehicle.

(c) Interrupt routine:

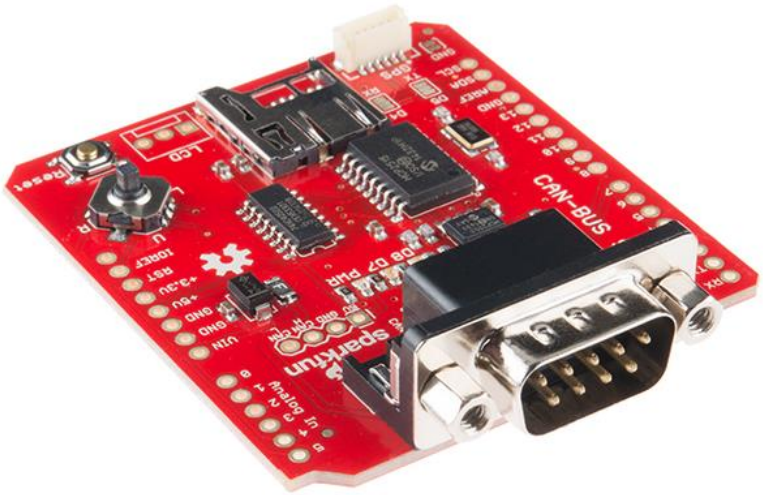
```
// The interrupt routine
void freqinput ()
{
  count++;
}
```

The interrupt routine is a functions that works as a counter.

1.6 CAN-BUS SHIELD

I. Introduction

The CAN-Bus Shield provides Arduino with CAN-Bus capabilities and allows to hack the vehicle. This shield allows to poll the ECU for information including coolant temperature, throttle position, vehicle speed, and engine rpms which can later be stored or displayed on a screen to make in-dash project.



II. Explanation

CAN-Bus uses the Microchip MCP2515 CAN controller with the MCP2551 CAN transceiver. CAN connection is via a standard 9-way sub-D for use with OBD-II cable. This particular shield is ideal for automotive CAN application. The shield also has a card holder, serial LCD connector and connector for an EM506 GPS module. These features make this shield ideal for data logging application.

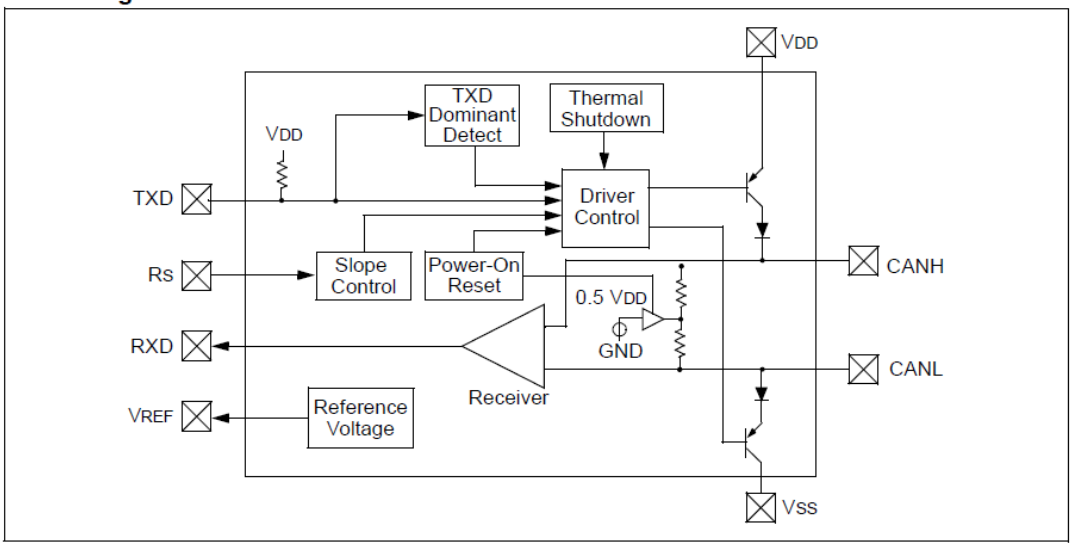
III. Features

- CAN v2.0B up to 1 Mb/s
- High speed SPI Interface (10 MHz)

- Standard and extended data and remote frames
- CAN connection via standard 9-way sub-D connector
- Power can supply to Arduino by sub-D via resettable fuse and reverse polarity protection.
- Socket for EM506 GPS module
- Micro SD card holder
- Connector for serial LCD
- Reset button
- Joystick control menu navigation control
- Two LED indicator

IV. Block Diagram

Block Diagram



V. Transmitter Function

The CAN bus has two states: Dominant and Recessive. A Dominant state occurs when the differential voltage between CANH and CANL is greater than a defined voltage (e.g.,1.2V). A Recessive state occurs when the differential voltage is less than a defined voltage (typically 0V). The Dominant and Recessive states correspond to the Low and High state of the TXD input pin, respectively. However, a Dominant state initiated by another CAN node will override a Recessive state on the CAN bus.

VI. Operating Modes

The RS pin allows three modes of operation to be selected:

1. High-Speed
2. Slope-Control
3. Standby

(a) HIGH-SPEED

High-Speed mode is selected by connecting the RS pin to VSS. In this mode, the transmitter output drivers have fast output rise and fall times to support high-speed CAN bus rates.

(b) SLOPE-CONTROL

Slope-Control mode further reduces EMI by limiting the rise and fall times of CANH and CANL. The slope, or slew rate (SR), is controlled by connecting an external resistor (REXT) between RS and VOL (usually ground). The slope is proportional to the current output at the RS pin. Since the current is primarily determined by the slope-control resistance value REXT, a certain slew rate is achieved by applying a specific resistance.

(a) STANDBY MODE

The device may be placed in Standby or SLEEP mode by applying a high-level to the RS pin. In SLEEP mode, the transmitter is switched off and the receiver operates at a lower current. The receive pin on the controller side (RXD) is still functional, but will operate at a slower rate. The attached microcontroller can monitor RXD for CAN bus activity and place the transceiver into normal operation via the RS pin (at higher bus rates, the first CAN message may be lost)

CONCLUSION AND FUTURE WORK

2.1 CONCLUSION

In this project, different sensors were used in order to acquire respective outputs. We tried to connect the accelerometer with an application installed on cell phone named *Bluetooth Terminal,* this interface allowed us to calibrate the accelerometer according to any initial position by entering 0 0 0 coordinates on that app. This also helped us using telemetry in our project in order to inspect and monitor the data acquired by accelerometer in real time on our cell phone.

2.2 FUTURE WORK

1. Displaying all the outputs on a single Dot Matrix Display instead of using 4 different LCDs.
2. Installing the Dot Matrix Display in a detachable steering.
3. Using Wifi and GPS instead of Bluetooth in telemetry.

REFERENCES

[1] https://www.sparkfun.com/

[2] https://www.arduino.cc/

[3] http://www.aemelectronics.com/