A picture containing tableware, clipart, plate, dishware

Description generated with very high confidence

**A Project Report**

**on**

**Car Over-Speeding Detection System**

**For the award of**

**ONTARIO GRADUATE CERTIFICATE**

Submitted by

Group 4

Govind Raj (C0753448)

Sunny Devgan (C0752045)

Varundeep Singh Sethi (C0754247)

Under the guidance of

**Dr. Mike Aleshams**

**ESE-4009 Embedded System Design Project**

(September 2020 – December 2020)

**Abstract**

This project aimed to develop a car over-speeding detection system with an LCD screen to display the vehicles' speed that crosses the sensors and alert the authorities when a car drives at a rate more than the given speed limit. Since the communication is wireless, the setup can be placed at different locations, and still, the data can be accessed from the cloud storage.

Background reading in a similar technology area leads us to the originality of the product. Research into further development also identified the minimum usability of our product. Our product runs.

After implementation, the detailed evaluation was conducted using testing techniques to evaluate product usability.

**Acknowledgment**

We want to express our most profound appreciation to everyone at Embedded Systems Design Engineering Class at Lambton College in Toronto, who provided us with the possibility to complete this project. Special gratitude to the project guide, Dr. Mike Aleshams, whose engagement and contribution help us coordinate our project and filing the final project report.

We would also like to extend our appreciation to Dr. Takis Zourntos, Course Coordinator, Embedded Systems Design Engineering, for guiding us throughout our education here.

In Toronto, Lambton College is a great place to study and work, mainly because of its supporting faculty members. Thanks to all our colleagues in college for supporting and encouraging our idea.

* Govind Raj
* Sunny Devgan
* Varundeep Singh Sethi

Table of Contents

List of Figures5

List of Tables6

Chapter I - Introduction7

Overview7

Problem Statement7

Goal and Objective7

Deliverables7

Milestones8

Limitations8

Outcomes9

Procedure 9

Chapter II – Literature Review10

Chapter III – Requirement and Analysis11

Hardware11

Software11

Power Requirement12

Block Diagram13

Chapter IV – Design14

Schematic layout14

Beaglebone Black15

16\*2 LCD with I2C Adapter16

HM-10 Bluetooth Module17

ESP826619

Relay19

Buzzer20

IR LED21

Chapter V – Implementation and Test22

Beaglebone Black, Arduino and peripherals22

Beaglebone Black, ESP8266 and ThingSpeak23

Keywords24

Program…………………………………………………………………………………………..26

Chapter VI – Evaluation37

Introduction37

Minimum Requirements37

Troubleshooting37

Chapter VII – Conclusion38

Future Work38

Chapter – User’s Guide39

Introduction39

Features39

Installation Process40

Thingspeak Cloud Service40

Chapter IX – References…………………………………………………………41

References41

**List of Figures**

|  |  |
| --- | --- |
| Figure No. | Description |
| 3.1 | Block Diagram |
| 4.1 | The schematic diagram for the Beaglebone side |
| 4.2 | The schematic diagram for the Arduino side |
| 4.3 | Beaglebone Black |
| 4.4 | Pinout of Beaglebone Black |
| 4.5 | 16\*2 I2C LCD |
| 4.6 | HM-10 Bluetooth module |
| 4.7 | ESP8266 |
| 4.8 | 5V relay |
| 4.9 | Piezo Buzzer |
| 4.10 | IR sensor |
| 5.1 | Initiating the system |
| 5.2 | Working of the system |
| 5.3 | Thingspeak channel showing off the over-speeding cars |

**List of tables**

|  |  |
| --- | --- |
| Figure No. | Description |
| 4.1 | Connection of LCD with Beaglebone Black |
| 4.2 | Connection details of HM-10 with Arduino Uno |
| 4.3 | Connection details of HM-10 with Beaglebone Black |
| 4.4 | Connection details of ESP8266 with Beaglebone Black |
| 4.5 | Connection details of relay with Beaglebone Black |
| 4.6 | Connection details of IR sensors with Arduino Uno |
| 8.1 | Included Components |
| 8.2 | Technical Specifications |

**Chapter I**

**Introduction**

**Overview –**

This project is about developing a portable 'car over-speeding detection system' with a touchscreen display, Bluetooth, wifi capabilities and cloud storage. The system will consist of an LCD screen, which will display the vehicle's speed that crosses the IR sensors. The display is with the police authorities and a buzzer and bulb that will notify the cops whenever an over-speeding vehicle passes by. The cost of product development was between $400-$500.

This type of system has restricted use. It is not a general-purpose system and can only be used by the police authorities. However, this setup can be made more effective by adding a few more components.

**Problem Statement –**

Street mishap events have expanded as of late, so there should be a framework to distinguish over-speeding vehicles. Current speed identification frameworks are handheld firearms held by police faculty that permit them to check vehicle speed and physically illuminate specialists about the car. This proposed framework needn't bother with any human capture and records vehicle speed and remotely advises specialists regarding over-speeding location.

**Goal –**

Develop a cheap and portable car over-speeding detection system by the end of the Fall 2020 academic term.

**Objective –**

* I am connecting a host processor, Beaglebone Black (BBB), to an Arduino Uno using a Bluetooth module.
* I am adding a 16 x 2 LCD screen to BBB.
* I am configuring wifi using an ESP8266 to connect to ThingSpeak.

**Deliverables –**

* A wireless car over-speeding detection system.
* Wifi, Bluetooth and cloud storage capabilities.
* Li-ion batteries to make the system portable.

**Milestones –**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task name** | **Start Date** | **End Date** | **Person in-charge** |
| Project Proposal | September 14, 2020 | October 9, 2020 |  |
| Finalizing hardware requirements | October 10, 2020 | October 13, 2020 | Varun |
| Testing each hardware part | October 14, 2020 | October 18, 2020 | Govind |
| Designing circuit in software | October 19, 2020 | November 1, 2020 | Varun |
| Interfacing IR sensors with Arduino Uno | November 2, 2020 | November 7, 2020 | Sunny |
| Interfacing LCD with Beaglebone Black | November 8, 2020 | November 13, 2020 | Govind |
| Interfacing ESP8266 with Beaglebone Black | November 14, 2020 | November 17, 2020 | Sunny |
| Interfacing buzzer and bulb with the Beaglebone Black | November 18, 2020 | November 23, 2020 | Sunny |
| Interfacing ThingSpeak with Beaglebone Black | November 24, 2020 | December 1, 2020 | Govind |
| The connection between the Beaglebone Black and Arduino Uno | December 2, 2020 | December 9, 2020 | Varun |
| Final Presentation | December 10, 2020 | December 15, 2020 |  |
| Final Report | December 18, 2020 |  |  |

**Limitations –**

* + The authorities can only find out the speed of the vehicles, not their license plates.
  + The range of the Bluetooth module is only 100 metres. So, the system won't work if the setups are kept far apart.
  + A delay of 15 sec is there to upload to ThingSpeak.

**Outcome –**

* The cops on the road can easily view the speeds.
* Buzzer and bulb to notify the over-speeding vehicle.
* The speeds are stored on the cloud storage so that everyone can view them.
* Lightweight and portable to carry.

**Procedure –**

1. Design the circuit on EasyEDA.
2. Install a suitable Debian image from Beagleboard.org on Beaglebone Black
3. Install Arduino IDE in the host machine.
4. Code ESP8266 by using Arduino IDE.
5. Connect IR sensors with Arduino Uno.
6. Connect the LCD with Beaglebone Black.
7. Connect ESP8266 to the Beaglebone Black.
8. Attach the Buzzer and bulb to the Beaglebone Black
9. Interface ThingSpeak with the Beaglebone Black.
10. Test the setup.
11. Solder the components on Zero PCB.

**Chapter II**

**Literature Review**

*"Vehicle Overspeeding Detection Project"* by Nevon Projects (n.d),[4] is talking about to compute the speed of a vehicle utilizing an 8051 Microcontroller. The framework figures the ideal opportunity for a car to cover the separation from point A to point B. This is the way speed (distance/time) is determined for the given vehicle. The gathering comprises an 8051 microcontroller and an IT transmitter-beneficiary pair, which are answerable for computing the speed. Contingent upon the time taken by the car to go between the focuses, we can check if the vehicle is over-speeding or not. On the off chance that the car is over-speeding, a signal alert will get actuated and tell us.

*"Computerized Over Speeding Detection and Reporting System"* by S. Malik, M. Iqbal, Z. Hassan, T. Tauqeer, R. Hafiz (2014),[8] planned a mechanized speed recognition framework which could identify the speed of vehicles and if over-speeding happened, extricated the permit number of car and sent it using email to Toll Plaza to charge fine. Doppler impact wonder was utilized for the speed estimation. If over-speeding was distinguished, a camera consequently caught the vehicle's image, and Digital Image Processing (DIP) strategies were applied to separate the permit number. MATLAB was utilized for picture preparation. After that, the extricated tag number was sent using email to Toll Plaza. The outcomes indicated that the proposed framework effectively-recognized overspeeding, freed the tag, had elite and could be sent on streets to check for overspeeding.

*"Arduino Car Speed Detector"* by Electronics Hub (2018), [1] meant to plan a vehicle speed identifier utilizing an Arduino. IR Sensors were the principal part of the venture that distinguished the speed of a vehicle. At the point when a vehicle voyaging arrived at the primary sensor, the IR Sensor got enacted. A clock was started from that second ahead, and it kept on keeping time until the vehicle arrived at the second IR Sensor. By mimicking the distance between the two sensors to be 5 meters, the speed could be determined at which the car went from IR Sensor 1 to IR Sensor 2 as the hour of movement was at that point known. Arduino finished all the computations and information gathering, and the end-product was shown on a 16X2 LCD Module.

**Chapter III**

**Requirements/Analysis**

**Hardware –**

Our portable speed detection system consists of an LCD screen, a master MCU, a slave MCU with wifi, Bluetooth and cloud storage capabilities. Also, to make the design more compact and tidier, we have neatly soldered the Zero PCB components.

Following is a list of the required hardware for this project –

* Beaglebone Black
* Arduino Uno
* IR Sensors
* HM-10 Bluetooth module
* Li-Ion Batteries
* ESP8266
* LCD Screen
* Relay
* Bulb
* Piezo Buzzer
* DC Voltage Regulator

**Software –**

Considering the software requirement, developing a prototype team will need a Linux machine or a Windows PC with Putty installed. We decided to forward with a Debian image for Beaglebone Black from Beagleboard.org.

Following is a list of required software for this project:

* Debian version 10 OS in Beaglebone Black
* Embedded C Programming Language
* Putty (For using Windows PC for programming and testing)
* EasyEDA for schematic designing
* ThingSpeak as cloud storage
* Nano Editor for writing the program
* Arduino IDE for programming Arduino Uno
* Microsoft Office for preparing all the documents
* Linux bash scripting to run our program directly after bootup without the help of a laptop

**Power Requirement –**

Each component has a power requirement. The list is as follows:

1. Beaglebone Black – 5V
2. Arduino Uno – 5V
3. LCD – 5V
4. ESP8266 – 3.3V
5. Bulb – 12V
6. Buzzer – 5V
7. IR sensors – 3.3V
8. HM-10 Bluetooth module – 3.3V

**Block Diagram –**

Diagram

Description automatically generated

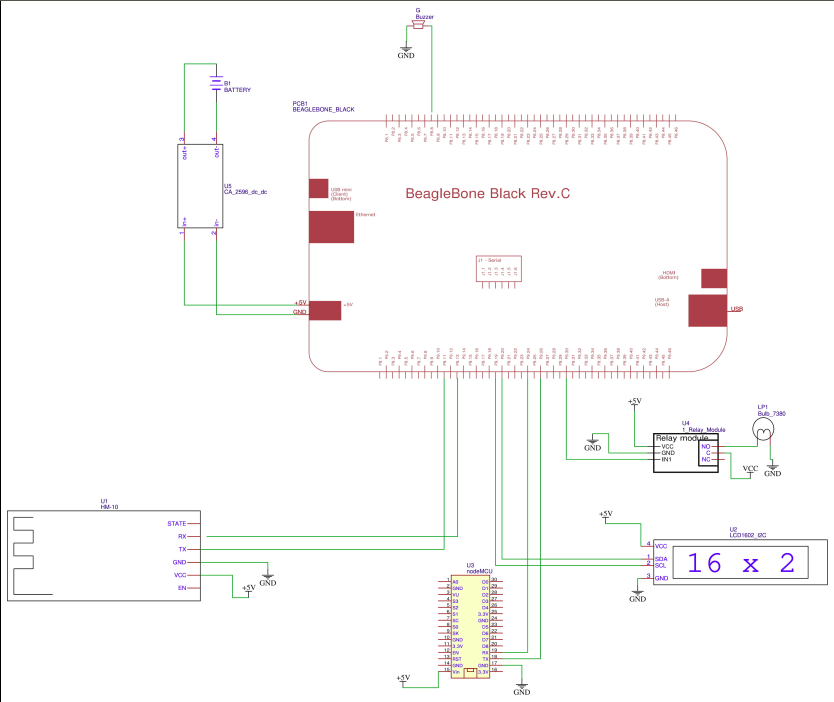
Figure 3.1 *Block Diagram*

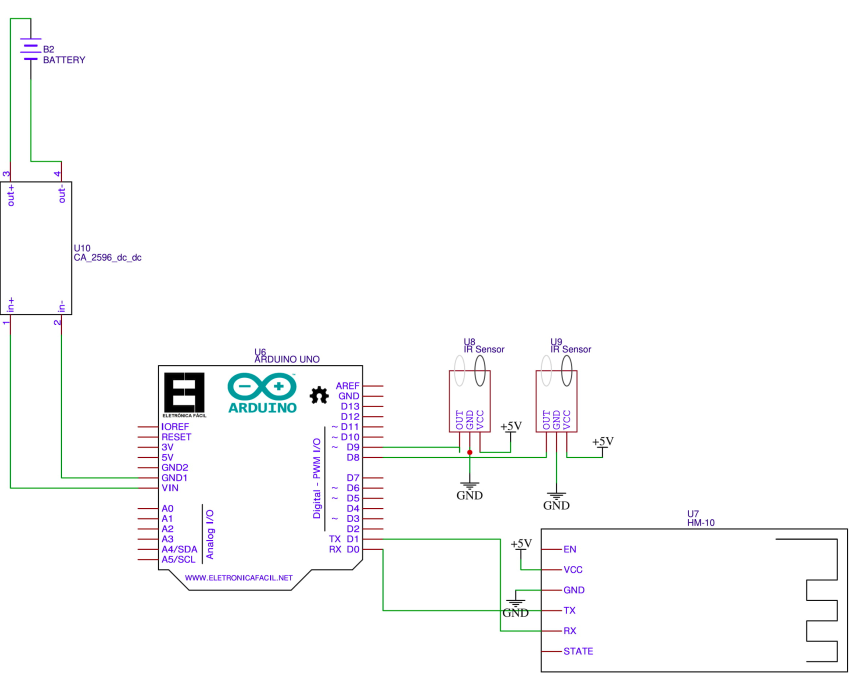
**Chapter IV**

**Design**

**Schematic layout -**

EasyEDA is an open-source platform that is used for the schematic design of our entire project.[11] Interfacing of Beaglebone Black with all other components such as 16\*2 LCD, ESP8266, Relay, Bulb, Buzzer, Bluetooth module and the slave device Arduino with IR sensors and Bluetooth module can be seen in below Figure 4.1 and Figure 4.2.

Figure 4.1 *Schematic diagram for Beaglebone side*

 Figure 4.2 *Schematic diagram for Arduino side*

**Beaglebone Black -**

Beaglebone Black is a single board computer manufactured by Texas Instruments[13]. We have used a Beaglebone Black Rev C, a 32-bit microcontroller with an AM335X processor with a 1 GHz ARM Cortex core, and has 4 GB of the flash memory 512 MB of RAM. Below, Figure 4.3 shows Beaglebone Black and Figure 4.4 shows the pinout of Beaglebone Black.



Figure 4.3 *Beaglebone Black*

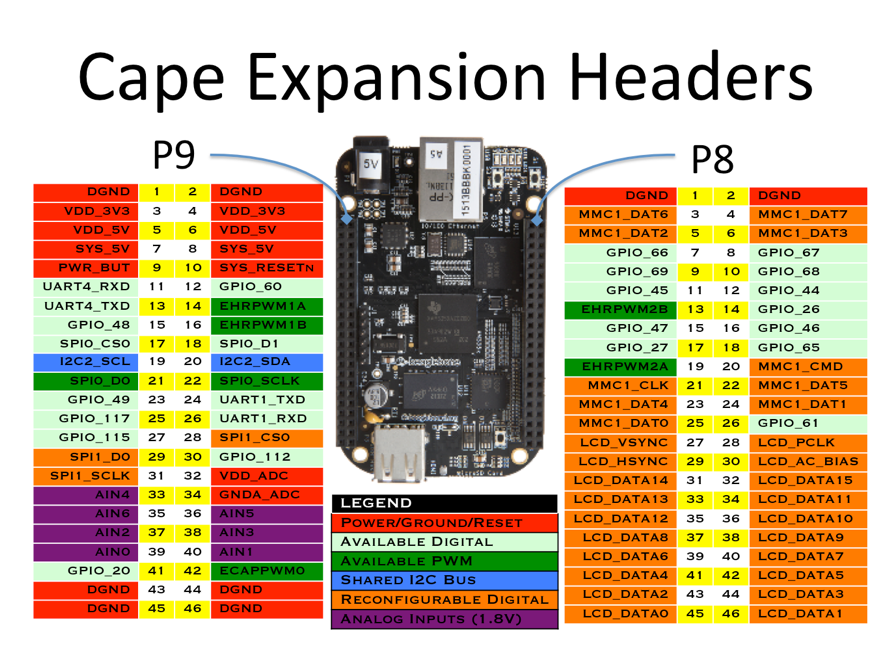


Figure 4.4 *Pinout of Beaglebone Black*

**16\*2 LCD with I2C adapter -**

16\*2 LCD used in our project [12] as shown in below Figure 4.5. It supports 16 characters per line, and there are two lines on the display. This is used in our project to display the vehicle's speed that passes and indicates whether it is average speed or Overspeed. A 5V supply from Beaglebone Black powers it. A PCF8574 module provides LCD to be interfaced with the Beaglebone Black using the I2C protocol.

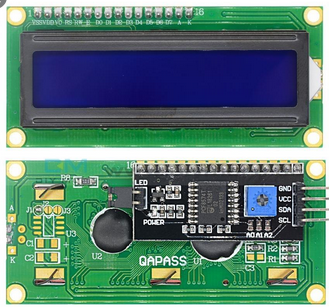


Figure 4.5 *16\*2 I2C LCD*

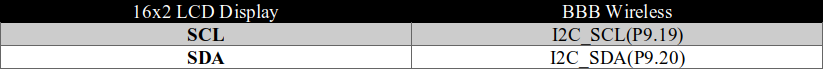


Table 4.1 *Connection of LCD with the Beaglebone Black*

**HM-10 Bluetooth Module -**

The Bluetooth module that we are using in our project is HM-10 [14]. HM-10 is a Bluetooth 4.0 module. It has a BLE chip (Bluetooth low energy consumption). UART Serial Interface. 12 GPIO. It can work in long-range (approximately 100 meters). Operating voltage is 2 to 6 V. It is used to communicate between Arduino Uno and Beaglebone Black. Figure 4.6 shows an HM-10 Bluetooth module. Table 4.2 and 4.3 show the connections of HM-10 with Arduino Uno and Beaglebone Black, respectively.

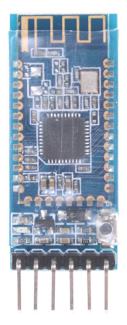


Figure 4.6 *HM-10 Bluetooth module*

|  |  |
| --- | --- |
| Arduino | HM-10 |
| D10 | TX |
| D11 | RX |
| 5v | VCC |
| GND | GND |

Table 4.2 *Connection details of HM-10 with Arduino Uno*

|  |  |
| --- | --- |
| Beaglebone Black | HM-10 |
| P9\_11(UART\_4 RXD) | TX |
| P9\_13(UART\_4 TXD) | RX |
| 5v | VCC |
| GND | GND |

Table 4.3 *Connection details of HM-10 with Beaglebone Black*

**ESP8266 -**

Beaglebone Black, our main MCU, does not have an inbuilt wifi module to connect to the internet. So, interfacing the ESP8266 module[15] to BBB gives the ability to connect to the internet and send data to the cloud. Below, Table 4.4 shows the connection details of ESP8266 with BBB.

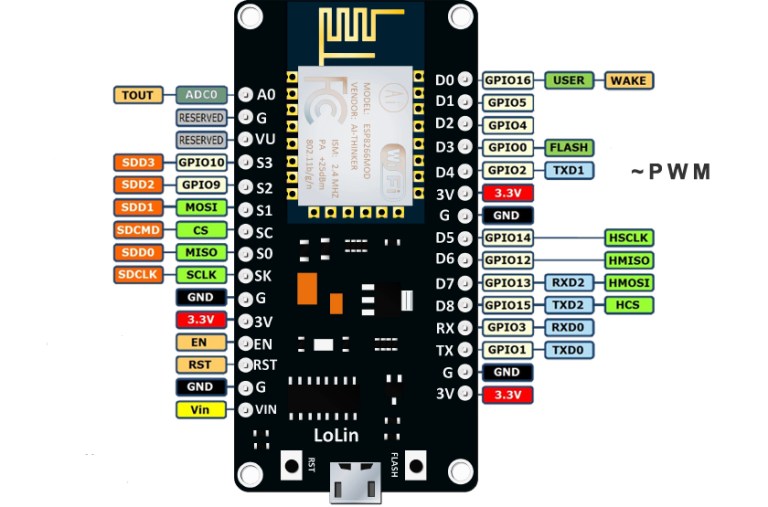


Figure 4.7 *ESP8266 -12E wifi module*

|  |  |
| --- | --- |
| Beaglebone Black | ESP8266 |
| P9\_24(UART\_1 TXD) | Pin 19 |
| P9\_26(UART\_1 RXD) | Pin 18 |
| 5v | VCC |
| GND | GND |

Table 4.4 *Connection details of ESP8266 with Beaglebone Black*

**Relay -**

Here, we are using a 5V relay [16] to operate a bulb. Relay acts as a switch that will turn on/off the bulb. We are using a single channel relay. Figure 4.7 shows the relay used. Table 4.5 shows the connection details of the relay with BBB.

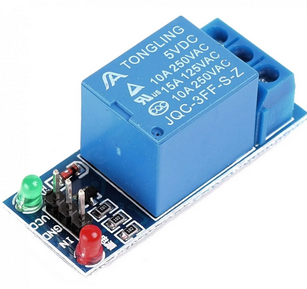


Figure 4.8 *5V Relay*

|  |  |
| --- | --- |
| Beaglebone Black | Relay |
| P9\_30 | IN |
| 5v | VCC |
| GND | GND |

Table 4.5 *Connection details of Relay with Beaglebone Black*

**Buzzer -**

Buzzer [17] is used as an indication to show the over-speeding of the car. Figure 4.8 shows the piezo Buzzer used in our project.



Figure 4.9 *Piezo buzzer*

|  |  |
| --- | --- |
| Beaglebone Black | Buzzer |
| P8\_8 | Positive pin |
| GND | GND |

Table 4.6 *Connection details of Buzzer with Beaglebone Black*

**IR LED -**

Two IR LEDs are used with Arduino Uno to detect the vehicle and calculate the speed. Figure 4.9 shows the IR sensor [18] used in the project. Table 4.7 shows the connection details of the IR sensor with the Arduino Uno.



Figure 4.10 *IR sensor*

|  |  |  |
| --- | --- | --- |
| Arduino | IR Sensor 1 | IR Sensor 2 |
| D8, D9 | OUT | OUT |
| 5v | VCC | VCC |
| GND | GND | GND |

Table 4.7 *Connection details of IR sensor with Arduino Uno*

**Chapter V**

**Implementation and Test**

Shape**Beaglebone Black, Arduino and its Peripherals -**

As we have discussed earlier, our project has two parts. The first part consists of the main MCU - Beaglebone Black, LCD, wifi module, Bluetooth module, Bulb, Buzzer, Relay and DC voltage regulator. This whole part is powered up with an 11.1 V Li-Ion battery. Since we have used a 12V bulb in our project, we had to use an 11.1 V battery. However, Beaglebone Black has a maximum supply voltage of only 5V, so we had to use a DC voltage regulator to reduce the voltage supply for the Beaglebone Black. The peripheral devices such as LCD, wifi module, Bluetooth, Relay are powered from a 5V input from Beaglebone Black.

Secondly, the Arduino Uno part consists of two IR sensors, a Bluetooth module and a Regulator. The power to operate this circuit comes from an 11.1 V battery via a voltage regulator. Both IR sensors and the Bluetooth module have a 5V input voltage from Arduino Uno.

When the system is powered up, Beaglebone Black takes some seconds to boot up.[19] Since we have kept our code in an auto script, we can see our project name displayed on LCD as soon as the Beaglebone Black boots up. After the car passes the sensor, we can see the speed of the vehicle on the LCD. If the car's rate exceeds the speed limit, we can see an ‘over speed’ message displayed on LCD, and the bulb and Buzzer turn on. Also, the Overspeed data is available on the ThingSpeak channel. Figure 5.1 shows the LCD screen after the Beaglebone Black has booted up.



Figure 5.1 *Initiating the system.*

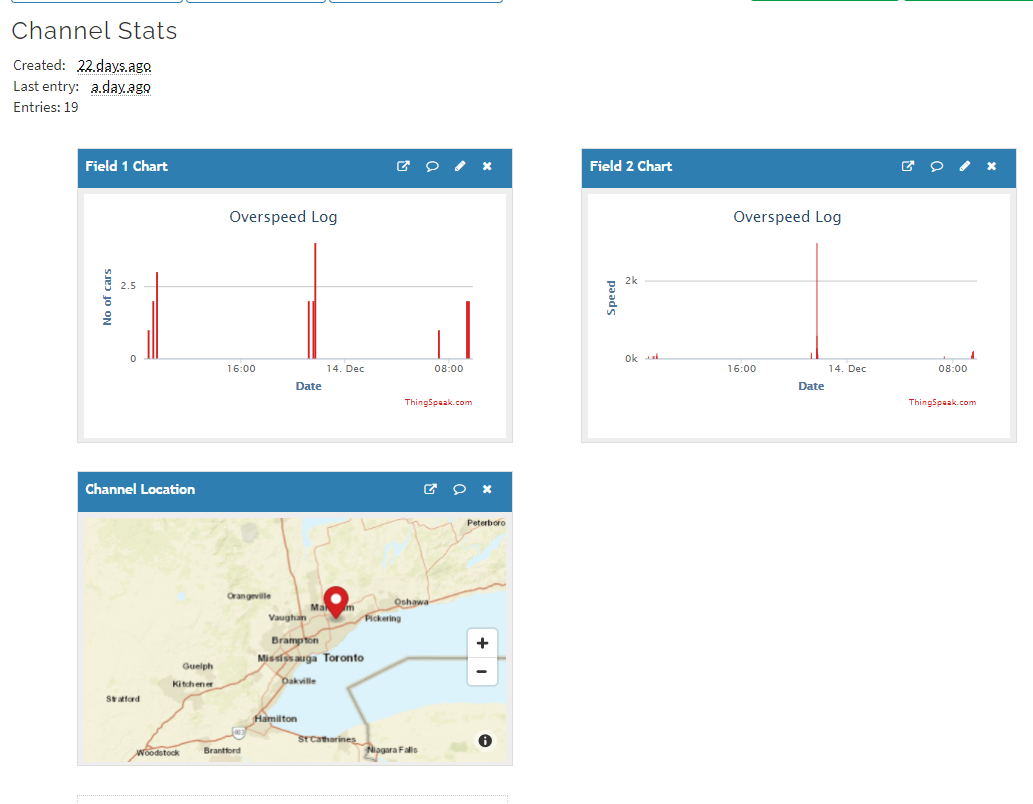
Below, Figure 5.2 shows the detected speed when a car passes through the sensors.



Figure 5.2 *Working of the system*

**Beaglebone Black, ESP8266 and ThingSpeak -**

Speed detected by the sensors is sent to a cloud server, which is ThingSpeak. We can see the number of over-speeding cars and also their speed in two fields of the channel. The status of the over-speeding car will update on ThingSpeak with a delay of 15 seconds. This delay is because we are using a trial version. If we buy the paid version, the wait can be reduced to 5 seconds. ThingSpeak channel is shown in Figure 5.3.

 Figure 5.3 *ThingSpeak channel showing status of the over-speeding cars*

**Keywords -**

In this chapter, we will discuss some of the components mentioned in our Project Proposal. Here is the list –

1. 8051 Microcontroller – The original project on the internet had 8051 as its main MCU. As this is an old technology product, we have replaced it with a Beaglebone Black.
2. Photodiodes – Photodiode comes with the IR sensor. IR sensor has two parts IR LED and photodiode. IR sensor works when the IR LED emits IR radiation and photodiode senses the IR radiation. So, photodiodes are placed in the sensor to capture the reflected IR radiations in a pre-defined distance range.
3. RF Transmitter/Receiver – In the original project involving an 8051 Microcontroller, RF signals were used as a communication medium. Hence, RF transmitter and receiver were used.
4. Keil Compiler – Keil Compiler is the compiler for 8051 microcontrollers. It is one of the most used 8051 compilers in the world. We are not using that in our project.
5. C Programming Language – Since our course focuses on C language, we have used it for writing codes.
6. Beaglebone Black – This is the primary MCU/master device of our project.
7. ESP8266 – This is a wifi module. Since Beaglebone Black does not have inbuilt wifi, we are using an external module. This device will help our system in connecting to the internet.
8. Relay – To alert the authorities about an over-speeding vehicle, we have used a bulb in our setup. The bulb has its operating voltage as 12V, and the Beaglebone Black cannot provide that. So, a relay has been used to convert the 5V from Beaglebone Black to 12V for the bulb.
9. LCD – An LCD screen is being used to display the vehicles' speed that passes the IR sensors.
10. HDMI - Beaglebone Black has an HDMI port. By using this port, we can display output on any screen that supports the HDMI feature.
11. Debian – Debian is the default OS for the Beaglebone Black. We are also using it to establish connections of different components with the Beaglebone Black.
12. Android – We can also install Android OS on the Beaglebone Black. This is one of the features of Beaglebone Black.
13. Ubuntu – Ubuntu is a Linux distribution based on Debian, and the good thing is that it's free and an open-source OS. We have earlier worked with the Beaglebone Black on Ubuntu, but we have not used it for this project.
14. Arduino – We are using an Arduino Uno as a slave device in our project. IR sensors are connected to the Arduino, and then information is transmitted to the Beaglebone Blak to display.
15. Bluetooth 4.0 – This is the Bluetooth version of the HM-10 Bluetooth module that we are using in our project.
16. Arduino IDE – This is the software to write codes for Arduino Uno.
17. Eclipse IDE – This software has been used to write our C language codes.
18. EasyEDA – This is an online PCB design and circuit simulator. We have used it to prepare our schematic diagram.
19. Piezo Buzzer – To alert the authorities about an over-speeding vehicle, we have also used a piezo buzzer. This Buzzer will create noise to notify the cops.
20. Proximity IR Sensors – These sensors are the input devices of our project. The sensors are used to detect the vehicle whose speed is then displayed on the LCD screen.
21. I2C - I2C protocol is used for interfacing the LCD screen as the screen supports I2C protocol.
22. LM2596 Regulator – This is the voltage regulator to regulate the voltage from the battery to 5V, needed by the Beaglebone Black.
23. Li-Ion Battery - This battery is the power source of our project. Since our project is portable, we are using these batteries.
24. GCC Compiler – GCC is a compiler system developed by the GNU project that supports various languages such as C, C++, Fortran etc. We have used GCC compiler to compile the code in Debian OS inside the Beaglebone Black.
25. SSH – SSH is short for secure shell. SSH client allows us to convert to a remote computer running an SSH server. A secure shell protocol can be used for the remote connection. We have made a connection with the Beaglebone Black using SSH.
26. UART – UART is the communication protocol used to interface multiple components like ESP8266, HM-10 Bluetooth module, etc.
27. Full Duplex Communication – This type of communication is when both devices can communicate with each other simultaneously. The transmission is bi-directional in this case. Full duplex communication is the advantage of using a UART protocol.
28. IEEE 802.11 - IEEE 802.11 is essential for the [IEEE 802](https://en.wikipedia.org/wiki/IEEE_802) arrangement of the neighbourhood (LAN) conventions. It indicates the area of [media access control](https://en.wikipedia.org/wiki/Medium_access_control)(MAC) and [even layer](https://en.wikipedia.org/wiki/Physical_layer) (PHY) patterns for actualizing remote community (WLAN) wifi PC correspondence in different frequencies, including, however not restricted to 2.4 GHz, 5 GHz, 6 GHz, and 60 GHz recurrence groups. Our wifi module deals with this LAN convention.
29. ThingSpeak – This is the cloud storage in our project. The speeds of the over-speeding vehicles are detected and stored on ThingSpeak.
30. MQTT – Message queuing telemetry transport is a lightweight publish-subscribe protocol used for transporting messages between devices. It can interface with cloud platforms such as ThingSpeak, but we have not used MQTT in our project.

**Program:**

**Main.c**

|  |
| --- |
| 1. #include <stdio.h> //Standard C input Output Library 2. #include <unistd.h> //defines miscellaneous symbolic constants and types, and declares miscellaneous functions 3. #include <string.h> //C Library for various String Operations 4. #include <termios.h> // Contains the definitions used by the terminal I/O interfaces 5. #include <fcntl.h> // File control, Open, close 6. #include <sys/signal.h>//calls readback function whenever uart data comes 7. #include <time.h> //for providing delay function 8. #include <stdlib.h> //general purpose standard library which includes memory allocation,conversions etc.. 9. #include <sys/ioctl.h> //input/output control //for reading and writing to uart 11. #include <linux/i2c-dev.h> //communicate with I2C devices 13. #include <iobb.h> //input/output 15. #define I2C\_BUS "/dev/i2c-2" // I2C bus device on a Beaglebone Black 16. #define I2C\_ADDR 0x27 // I2C slave address for the LCD module 18. #define BAUDRATE B9600 20. int i2cFile; 21. int cursor\_pos; 23. int file, file1, i; // Variable integers 24. char receive[20]; // declare a char array for receiving data 25. char buf[100]; // A buffer char array to store temporary data 27. int speed = 0; 28. int threshold = 60; 30. int bytes\_read = 0; 31. int nbytes; 32. int bytes\_written; 34. int relayPin = 30; 35. int buzzerPin = 8; 37. //Function Declaration 38. void readback(int status); 39. void delay(unsigned int mseconds); 40. void setUART(); 41. void setUART1(); 42. void send\_simple\_string\_to\_lcd(char \* str); 43. void go\_to(int line, char position); 44. void display\_on(); 45. void clear\_display(); 46. void i2c\_send\_byte(unsigned char data); 47. void i2c\_start(); 48. void lcd\_init(); 50. //https://github.com/xanthium-enterprises/Serial-Port-Programming-on-Linux/blob/master/USB2SERIAL\_Read/Reciever%20(PC%20Side)/SerialPort\_read.c 52. int main(void) // Main function 53. { 55. iolib\_init(); 56. iolib\_setdir(9, relayPin, DigitalOut); 57. iolib\_setdir(8, buzzerPin, DigitalOut); 58. pin\_high(9, relayPin); 59. pin\_low(8, buzzerPin); 61. i2c\_start(); //Start I2C Communication 63. lcd\_init(); //Initialize the LCD 65. clear\_display(); //Clear the display 67. // /dev/ttyO4 is linked with UART4 Port of Beaglebone 68. if ((file = open("/dev/ttyO4", O\_RDWR | O\_NOCTTY | O\_NONBLOCK)) < 0) // Try opening file in Read Write mode 69. { 70. printf("UART4: Failed to open the file.\n"); //A message Print 71. return 0; 72. } 74. // /dev/ttyO4 is linked with UART4 Port of Beaglebone 75. if ((file1 = open("/dev/ttyO1", O\_RDWR | O\_NOCTTY | O\_NONBLOCK)) < 0) // Try opening file in Read Write mode 76. { 77. printf("UART1: Failed to open the file.\n"); //A message Print 78. return 0; 79. } 81. setUART(); 83. setUART1(); 85. printf("\n Wait \n"); 87. go\_to(1, 0); 88. send\_simple\_string\_to\_lcd("SPEED: --"); //Print a message on LCD 90. while (1) { 92. delay(1000); 93. if (speed > 0) { 94. printf("Speed : %d\n", speed); 95. go\_to(1, 0); 96. send\_simple\_string\_to\_lcd("SPEED: "); //Print a message on LCD 97. char str[5]; 98. sprintf(str, "%d", speed); 99. send\_simple\_string\_to\_lcd(str); 100. send\_simple\_string\_to\_lcd("km/h "); 102. if (speed > threshold) { 103. go\_to(2, 0); 104. send\_simple\_string\_to\_lcd(" OVERSPEED "); 105. pin\_low(9, relayPin); //Relay On 106. pin\_high(8, buzzerPin); 107. delay(1000); 108. pin\_high(9, relayPin); //Relay OFF 109. pin\_low(8, buzzerPin); 111. printf("Sending data to ESP\n\n\n"); 112. memset(buf, 0, sizeof(buf)); 113. strcpy(buf, "1"); // Copy a string in buf char array 114. nbytes = strlen(buf); // Store size of buf array in nbytes 115. bytes\_written = write(file1, buf, nbytes); // Sending message to ESP Module 117. } else { 119. go\_to(2, 0); 120. send\_simple\_string\_to\_lcd(" NORMAL SPEED "); 121. pin\_high(9, relayPin); //Relay OFF 122. pin\_low(8, buzzerPin); 123. } 124. speed = 0; 125. } 127. } 129. printf("\n Done \n"); 131. close(file); //Close the file at last 132. } 134. void readback(int status) { 136. bytes\_read = read(file, & receive, sizeof(receive)); // Read the incoming Message from GSM Module 137. // Read the file and store the data in receive , read 100 bytes max 139. if (bytes\_read > 1) //If no. of bytes are read is more than 1 140. { 141. //printf("\n\nBytes Received - %d\n",bytes\_read); // Print how many bytes was received 142. printf("Got Speed: "); 143. for (int i = 0; i < bytes\_read; i++) //a for loop to print data byte by byte 144. { 145. if (receive[i] == 'z') { 146. receive[i] = '4'; 147. } 149. printf("%c", receive[i]); //print a byte of message from GSM Module 150. } 151. printf("\n###\n"); //General Print 153. bytes\_written = write(file, receive, bytes\_read); // Sending message to GSM Module 155. sscanf(receive, "%d", & speed); 157. } else { 158. tcflush(file, TCIFLUSH); 159. } 160. tcflush(file, TCIFLUSH); 162. } 164. void delay(unsigned int mseconds) { 165. clock\_t goal = mseconds \* 1000 + clock(); 166. while (goal > clock()); 167. } 169. void setUART() { 170. struct sigaction saio; /\* definition of signal action \*/ 171. /\* install the signal handler before making the device asynchronous \*/ 172. saio.sa\_handler = readback; 173. //saio.sa\_mask = 0; 174. sigemptyset( & saio.sa\_mask); 175. sigaddset( & saio.sa\_mask, SIGINT); 176. saio.sa\_flags = 0; 177. saio.sa\_restorer = NULL; 178. sigaction(SIGIO, & saio, NULL); 179. fcntl(file, F\_SETOWN, getpid()); 180. fcntl(file, F\_SETFL, FASYNC); 182. struct termios newtio; 183. /\* set new port settings for canonical input processing \*/ 184. newtio.c\_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD; 185. newtio.c\_iflag = IGNPAR | ICRNL; 186. newtio.c\_oflag = 0; 187. newtio.c\_lflag = ICANON; 188. newtio.c\_cc[VMIN] = 0; 189. newtio.c\_cc[VTIME] = 1; 190. tcflush(file, TCIFLUSH); 191. tcsetattr(file, TCSANOW, & newtio); 192. tcflush(file, TCIOFLUSH); 194. } 196. void setUART1() { 197. // struct sigaction saio1; /\* definition of signal action \*/ 198. // /\* install the signal handler before making the device asynchronous \*/ 199. // saio1.sa\_handler = readback; 200. // //saio.sa\_mask = 0; 201. // sigemptyset(&saio1.sa\_mask); 202. // sigaddset(&saio1.sa\_mask, SIGINT); 203. // saio1.sa\_flags = 0; 204. // saio1.sa\_restorer = NULL; 205. // sigaction(SIGIO,&saio1,NULL); 206. // fcntl(file1, F\_SETOWN, getpid()); 207. // fcntl(file1, F\_SETFL, FASYNC); 209. struct termios newtio1; 210. /\* set new port settings for canonical input processing \*/ 211. newtio1.c\_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD; 212. newtio1.c\_iflag = IGNPAR | ICRNL; 213. newtio1.c\_oflag = 0; 214. newtio1.c\_lflag = ICANON; 215. newtio1.c\_cc[VMIN] = 0; 216. newtio1.c\_cc[VTIME] = 1; 217. tcflush(file1, TCIFLUSH); 218. tcsetattr(file1, TCSANOW, & newtio1); 219. tcflush(file1, TCIOFLUSH); 221. } 223. void i2c\_start() //Function to start i2c communication 224. { 225. if ((i2cFile = open(I2C\_BUS, O\_RDWR)) < 0) { 226. printf("Error failed to open I2C bus [%s].\n", I2C\_BUS); 227. exit(-1); 228. } 229. // set the I2C slave address for all subsequent I2C device transfers 230. if (ioctl(i2cFile, I2C\_SLAVE, I2C\_ADDR) < 0) { 231. printf("Error failed to set I2C address [%s].\n", I2C\_ADDR); 232. exit(-1); 233. } 234. } 236. void i2c\_send\_byte(unsigned char data) //Sens a byte of data via i2c 237. { 238. unsigned char byte[1]; 239. byte[0] = data; 240. write(i2cFile, byte, sizeof(byte)); 241. /\* -------------------------------------------------------------------- \* 242. \* Below wait creates 1msec delay, needed by display to catch commands \* 243. \* -------------------------------------------------------------------- \*/ 244. usleep(1000); 245. } 247. void clear\_display() //Function to clear Display 248. { 249. /\* -------------------------------------------------------------------- \* 250. \* Display clear, cursor home \* 251. \* -------------------------------------------------------------------- \*/ 252. usleep(40); // wait 40usec 253. i2c\_send\_byte(0b00000100); // 254. i2c\_send\_byte(0b00000000); // D7-D4=0 255. i2c\_send\_byte(0b00010100); // 256. i2c\_send\_byte(0b00010000); // D0=display\_clear 258. } 260. void display\_on() { 261. /\* -------------------------------------------------------------------- \* 262. \* Turn on the display \* 263. \* -------------------------------------------------------------------- \*/ 264. usleep(40); // wait 40usec 265. i2c\_send\_byte(0b00001100); // 266. i2c\_send\_byte(0b00001000); // D7-D4=0 267. i2c\_send\_byte(0b11101100); // 268. i2c\_send\_byte(0b11101000); // D3=1 D2=display\_on, D1=cursor\_on, D0=cursor\_blink 269. cursor\_pos = 1; 270. } 272. void go\_to(int line, char position) //Function to move cursor of lcd 273. { 274. if (line == 1) position = 0x80 + position; 276. else if (line == 2) position = 0xC0 + position; 278. int i; 280. char n1, n2, ln1, ln2, mask1, mask2; 282. ln1 = 0b11111100; 283. ln2 = 0b11111000; 285. mask1 = 0b11110000; 286. mask2 = 0b00001111; 288. n1 = (position & mask1) | mask2; 289. n2 = ((position & mask2) << 4) | mask2; 291. i2c\_send\_byte(n1 & ln1); 292. i2c\_send\_byte(n1 & ln2); 293. i2c\_send\_byte(n2 & ln1); 294. i2c\_send\_byte(n2 & ln2); 296. } 298. void send\_simple\_string\_to\_lcd(char \* str) //A function to send Simple String to LCD 299. { 301. int i; 302. for (i = 0; 303. (str[i] != 0x00); i++) { 305. char chr = str[i]; 307. int i; 309. char n1, n2, ln1, ln2, mask1, mask2; 311. ln1 = 0b11111101; 312. ln2 = 0b11111001; 314. mask1 = 0b11110000; 315. mask2 = 0b00001111; 317. n1 = (chr & mask1) | mask2; 318. n2 = ((chr & mask2) << 4) | mask2; 320. i2c\_send\_byte(n1 & ln1); 321. i2c\_send\_byte(n1 & ln2); 322. i2c\_send\_byte(n2 & ln1); 323. i2c\_send\_byte(n2 & ln2); 325. } 327. } 329. void lcd\_init() { 331. /\* -------------------------------------------------------------------- \* 332. \* Initialize the display, using the 4-bit mode initialization sequence \* 333. \* -------------------------------------------------------------------- \*/ 335. sleep(0.4); // wait 40msec 336. i2c\_send\_byte(0b00110100); // D7=0, D6=0, D5=1, D4=1, RS,RW=0 EN=1 337. i2c\_send\_byte(0b00110000); // D7=0, D6=0, D5=1, D4=1, RS,RW=0 EN=0 339. sleep(0.1); // wait 10msec 340. i2c\_send\_byte(0b00110100); // 341. i2c\_send\_byte(0b00110000); // same 342. sleep(0.1); // wait 10msec 343. i2c\_send\_byte(0b00110100); // 344. i2c\_send\_byte(0b00110000); // 8-bit mode init complete 345. sleep(0.1); // wait 10msec 346. i2c\_send\_byte(0b00100100); // 347. i2c\_send\_byte(0b00100000); // switched now to 4-bit mode 349. /\* -------------------------------------------------------------------- \* 350. \* 4-bit mode initialization complete. Now configuring the function set \* 351. \* -------------------------------------------------------------------- \*/ 352. usleep(100); // wait 100usec 353. i2c\_send\_byte(0b00100100); // 354. i2c\_send\_byte(0b00100000); // keep 4-bit mode 355. i2c\_send\_byte(0b10000100); // D3=2lines 356. i2c\_send\_byte(0b10000000); // D2=char5x8 358. clear\_display(); 360. display\_on(); 362. sleep(1); //delay of 1 second 363. } |

**Arduino Code:**

1. #include <SoftwareSerial.h>// Library for making a UART port
2. SoftwareSerial transmitter (10, 11); //RX & TX pin for UART Port
3. //Keeping Sensor 1 and Sensor 2 at a difference of 10 cm
4. int ir1 = 8; // Pin Number for Sensor 1
5. int ir2 = 9; // Pin Number for Sensor 2
7. long begin, end; // Variables to keep track of time
8. int difference; // Variable to Store time taken by Car
9. float speed; //Variable to Store Calculated Speed of Car
10. int speedLim = 60; //Speed limit for Over speeding
12. void setup () {
13. Serial.begin(9600); //Initializing Serial Communications
14. transmitter.begin(9600);
15. pinMode(ir1, INPUT\_PULLUP); //Making IR-1 pin as Input
16. pinMode(ir2, INPUT\_PULLUP); //Making IR-2 pin as Input
17. Serial.println("STARTED");
18. }
19. void loop() {
20. if (digitalRead(ir1) == LOW) //If Car detected in front of Sensor -1
21. {
22. Serial.println("Car Detected at First Sensor"); //Print a message
23. begin = millis(); //Record the Time when Car reaches sensor 1
24. while (digitalRead(ir2) == HIGH) //Until Car is not detected at Sensor-2
25. {
26. delay(1); //Waiting until Car Reaches Sensor 2
27. }
28. //Car reached at Sensor 2
29. end = millis(); //Record the Time when Car reaches sensor 2
31. difference = end - begin; //Calculate total time taken
32. speed = difference / 1000.0; // Converting milliseconds to seconds
33. speed = 10 / speed;
34. speed = (speed \* 3600) / 1000.0; //Converting seconds to Hour and Km
35. Serial.print("Speed of Car Detected: ");
36. Serial.print(int(speed));
37. Serial.println(" km/h"); //Print the calculated speed
38. String data = "";
39. data += String(int(speed));
40. data += "";
41. data.replace('4', 'z');
42. if (speed > speedLim) //If Detected Speed is more that Speed Limit
43. {
44. Serial.println("OVERSPEED DETECTED"); //Show Warning message
45. }
46. for (int i = 0; i < 3; i++) {
47. transmitter.println(data);
48. transmitter.println(" ");
49. transmitter.println(" ");
50. transmitter.println(" ");
51. transmitter.println(" ");
52. transmitter.println(" ");
53. transmitter.println(" ");
54. delay(1000);
55. String received;
56. if (transmitter.available()) {
57. Serial.print("GOT: ");
58. while (transmitter.available()) {
59. received += (char) transmitter.read();
60. delay(1);
61. }
62. }
63. Serial.println(received);
64. if (int(speed) == received.toInt()) {
65. Serial.println("Data Transmitted Successfully");
66. break;
67. } else {
68. Serial.println("Data Transmission Failed. Retrying..");
69. }
70. }
72. }
73. else {
74. if (transmitter.available()) {
75. Serial.write(transmitter.read());
76. }
77. }
78. }

**ESP8266 Node MCU code:**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

1. #include "ThingSpeak.h"
2. #include <ESP8266WiFi.h>
3. char ssid[] = "572pharmacy"; // your network SSID (name)
4. char pass[] = "572pharmacY001"; // your network password
5. WiFiClient client;
6. unsigned long myChannelNumber = 1239710; //thingspeak channel id
7. const char \* myWriteAPIKey = "K9N399Q6EPCKDTTH"; //API key for channel
8. long oldmillis;
9. int speedx = -1;
10. void setup() {
11. Serial.begin(9600); //Initialize serial
12. WiFi.mode(WIFI\_STA);
13. ThingSpeak.begin(client); // Initialize ThingSpeak
14. oldmillis = millis();
15. }
17. void loop() {
19. // Connect or reconnect to WiFi
20. if (WiFi.status() != WL\_CONNECTED) {
21. Serial.print("Attempting to connect to SSID: ");
22. Serial.println(ssid);
23. while (WiFi.status() != WL\_CONNECTED) {
24. WiFi.begin(ssid, pass); // Connect to WPA/WPA2 network. Change this line if using open or WEP network
25. Serial.print(".");
26. delay(5000);
27. }
28. Serial.println("\nConnected.");
29. }
31. if (Serial.available()) {
32. speedx = Serial.parseInt();
33. Serial.println(speedx);
34. }
36. if (speedx != -1 and(millis() - oldmillis) > 16000) {
38. ThingSpeak.setField(1, 1);
39. ThingSpeak.setField(2, speedx);
41. oldmillis = millis();
43. Serial.print("Uploading Data: ");
44. Serial.println(speedx);
46. // Write to ThingSpeak. There are up to 8 fields in a channel, allowing you to store up to 8 different
47. // pieces of information in a channel. Here, we write to field 1.
48. int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
49. if (x == 200) {
50. ;
51. //Serial.println("Channel update successful.");
52. } else {
54. Serial.println("Problem updating channel. HTTP error code " + String(x));
55. }
56. }
57. delay(100);
58. speedx = -1;
59. }

**Chapter VI**

**Evaluation**

**Introduction -**

This section clarifies the evaluation of the final structure for ensuring proper functioning. This also details the procedures and approaches we have undergone for troubleshooting at each stage of the project.

**Minimum Requirements -**

The project's minimum requirement was to develop a sensing system using proximity IR sensors and send the information in a user-friendly format to an IOT website called ThingSpeak, which has the feature for data storage.

**Troubleshooting -**

The challenging task that we faced while troubleshooting was making the connection between the two Bluetooth modules. On connecting the Bluetooth modules to the two MCUs, both the modules were not able to connect.

For the Beaglebone Black, UART must be manually enabled. So, we made a script file for automatically helping UART during booting up.

I2C protocol was used for interfacing LCD to the Beaglebone Black, so data was sent as nibbles from the Beaglebone Black.

**Chapter VII**

**Conclusion**

To conclude, we can say that the aim of the project has been achieved successfully. We have managed to build a small-scale car over-speeding detection system. The project consists of 2 MCUs – Beaglebone Black and Arduino Uno, Bluetooth modules to establish a connection between the MCUs, wifi module to connect our system to the internet and alert systems. Gone are the days when the manual intervention was needed to detect over-speeding cars. Our system does that for you, and with a little, more advancements can be automatically subtle the culprit as well. Upon completing the project, the user can view the live status from the [CAR OVERSPEED](https://thingspeak.com/channels/1239710) public channel from ThingSpeak.

**Future Work –**

Our project has a lot of potentials. This project will automate the existing challan system and will assist the cops in making their work easy. Here are a few future scopes of the project –

1. A bigger LCD can be placed on the road to show the speed to the driver as well.
2. A camera can be installed with the setup, which automatically captures the picture of the license plate of the over-speeding vehicle (like a red-light camera).
3. The license plate can be looked up in the database, and a fine can be sent to the driver of that particular vehicle.
4. Since the data is being uploaded on the cloud storage, the driver can also view his fines and pay them directly.

**Chapter VIII**

**User’s Guide**

**Introduction -**

Car Overspeed detection system is an IOT based web platform that features live updation of over-speeding vehicles on the road. The steps for proper installation and working of the system are discussed in the upcoming sections.

**Features -**

* 2 IR sensors connected to Arduino Uno are kept on the road for detecting the vehicle speed.
* 16x2 LCD that shows the speed of the vehicle when any vehicle passes through the sensors.
* Bluetooth 4.0 is used to send data wirelessly.
* A cloud platform called [Thingspeak](https://thingspeak.com/channels/1239710) shows the live updates of several over-speeding cars and their speeds and the channel's location.

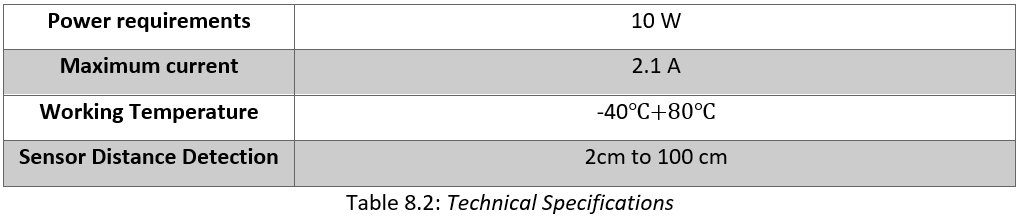
The sensor placed on the road detects the vehicle, and the speed calculated by Arduino Uno is sent to the Beaglebone Black wirelessly via Bluetooth. After that, we can see the rate on the LCD. The LCD also displays if the speed is an over-speed or an average pace. The number of vehicles and their over-speed can be viewed on the cloud platform ThingSpeak, which can be accessed from any laptop/computer. The SSID and Password of wifi are already set in hardware so that the system can automatically connect to the internet when powering up.

The information about the components used and the system's technical specifications are given in Table 8.1 and Table 8.2.

|  |  |
| --- | --- |
| COMPONENT | FUNCTION |
| LCD DISPLAY | To display the speed of the vehicle. |
| BUZZER AND BULB | To let know the user about over-speeding |
| IR SENSOR | To detect the vehicle on the road. |
| CONTROL UNIT | To process the data and send it to ThingSpeak. |
| ESP8266 | To connect the main MCU to the internet. |
| HM-10 | Bluetooth module for connecting two MCU |

Table 8.1 *Included Components*





**Installation process -**

* Both Arduino Uno and Beaglebone Black can be powered with an 11.1V battery.
* Bluetooth LED will stop blinking if both the devices are connected.
* The LCD screen should be kept in front of the officer to have a clear display of speed.
* Sensors should be placed along the road's side and should be 1 cm or 2 cm above the ground to avoid malfunction.
* Bulb and Buzzer also should be placed facing the officer to have an alert of an over-speeding vehicle.

**Thingspeak Cloud Service -**

* The link to the ThingSpeak channel is [https://thingspeak.com/channels/1239710.](https://thingspeak.com/channels/1239710)
* The link is public, and it can be shared with anyone who is engaged in monitoring vehicle speed detection.
* The main features of the website are:
* It can be accessed from anywhere in the world.
* Fields in the ThingSpeak channel can be viewed in different graph formats.
* The location of the system can also be viewed from the channel.

**Chapter IX**

**References**

1. Arduino Car Speed Detector. (2018, December 28). Retrieved from https://www.electronicshub.org/arduino-car-speed-detector/
2. Beaglebone Black (n.d). Abra Electronics Retrieved from [BEAGLEBONE\_BLACK(2).jpeg (750×750) (abra-electronics.com)](https://abra-electronics.com/images/detailed/131/BEAGLEBONE_BLACK(2).jpeg)
3. Beaglebone Black. (n.d). Retrieved from https://beagleboard.org/black
4. Botshop (n.d). 5V relay with status led. Retrieved from [1-Channel-Relay-Module.jpg (1000×750) (bot shop.co.za)](https://www.botshop.co.za/wp-content/uploads/2016/09/1-Channel-Relay-Module.jpg)
5. Campbell, S. (n.d). Basics of UART Communication. Retrieved from <https://www.circuitbasics.com/basics-uart-communication/>
6. Car Overspeeding Detection Project. (n.d). Retrieved from <https://nevonprojects.com/car-overspeeding-detection-system/>
7. Circuit Diagrams(n.d). ESP8266 wifi module. Retrieved from [ESP8266-12e-Pinout-13.jpg (768×507) (circuit-diagramz.com)](https://circuit-diagramz.com/wp-content/uploads/2018/11/ESP8266-12e-Pinout-13.jpg)
8. Components 101(2017 August)*16x2 LCD Module.* Retrieved from:

<https://components101.com/16x2-lcd-pinout-datasheet>

1. Components 101(n.d). HM-10 Bluetooth module. Retrieved from [HM-10-Bluetooth-Module.jpg (700×960) (components101.com)](https://components101.com/sites/default/files/components/HM-10-Bluetooth-Module.jpg)
2. Currey, M(2016, April 09). Arduino to Arduino by Bluetooth. Retrieved from <http://www.martyncurrey.com/arduino-to-arduino-by-Bluetooth/>
3. EasyEDA (n.d). *EasyEDA* Retrieved from <https://easyeda.com/page/about>
4. Elinux.org(2015). Beaglebone/I2CLCDDemo Retrieved from: [https://elinux.org/BeagleBone/I2CLCDDemo#I2C\_LCD\_wiring](https://elinux.org/BeagleBone/I2CLCDDemo)
5. ESP8266. (n.d). Retrieved from https://www.espressif.com/en/products/socs/esp8266
6. Gyawali, P. K. (2014, August). A final year project report on the vehicle over speed detection and recognition. Retrieved from https://www.researchgate.net/publication/325108341\_A\_FINAL\_YEAR\_PROJE CT\_REPORT\_ON\_VEHICLE\_OVER\_SPEED\_DETECTION\_AND\_RECOGNI TION
7. Malik, S., Asad, I., Zohaib, H., Tauqeer, T. (2014). Automated Over Speeding Detection and Reporting System. Retrieved from <https://www.researchgate.net/publication/269828594_Automated_Over_Speeding_Detection_and_Reporting_System>
8. Nuttyenginner (n.d). IR sensor module. Retrieved from [IR-sensor-600x600.jpg (600×600) (nuttyengineer.com)](https://www.nuttyengineer.com/wp-content/uploads/2016/10/IR-sensor-600x600.jpg)
9. Pandit, A(2019, June 10).Circuit Digest. How to Use the HM-10 BLE Module with Arduino to control an LED using the Android App**.** Retrieved from https://circuitdigest.com/microcontroller-projects/how-to-use-arduino-and-hm-10- ble-module-to-controlled-with-android-app
10. Pcbboardca (n.d). Mini Piezo Buzzer. Retrieved from [piezo-buzzer-01-800x800.jpg (800×800) (pcboard.ca)](https://www.pcboard.ca/image/cache/catalog/products/buzzers/piezo-buzzer/piezo-buzzer-01-800x800.jpg)
11. ThingSpeak (n.d). ThingSpeak for IoT projects. Retrieved from: <https://thingspeak.com/>
12. Wifi – IEEE Standards. (n.d). Retrieved from [https://www.tutorialspoint.com/wi- fi/wifi\_ieee\_standards.htm](https://www.tutorialspoint.com/wi-%20fi/wifi_ieee_standards.htm)
13. Xanthium-enterprises (2015, Sept 04). Serial port programming on linux. Retrieved from [Serial-Port-Programming-on-Linux/SerialPort\_read.c at master · xanthium-enterprises/Serial-Port-Programming-on-Linux (github.com)](https://github.com/xanthium-enterprises/Serial-Port-Programming-on-Linux/blob/master/USB2SERIAL_Read/Reciever%20(PC%20Side)/SerialPort_read.c)
14. How to program NodeMCU with Arduino IDE. (2019, February 7). Retrieved from <https://www.youtube.com/watch?v=flLMMHCNkQE>
15. NodeMCU GPIO Arduino IDE. (n.d.). Retrieved from <https://www.electronicwings.com/nodemcu/nodemcu-gpio-with-arduino-ide>
16. Serial ports/ UART. (n.d.). Retrieved from <http://beaglebone.cameon.net/home/serial-ports-uart>
17. How to get UART serial ports working on the beaglebone black. (2018, February 9). Retrieved from <https://electronics.trev.id.au/2018/02/09/get-uart-serial-ports-working-beaglebone-black/>
18. NodeMCU GPIO with Arduino IDE. (n.d.). Retrevied from <https://www.electronicwings.com/nodemcu/nodemcu-gpio-with-arduino-ide>
19. Establishing wifi connection.(n.d). Retrieved from: <https://tttapa.github.io/ESP8266/Chap07%20-%20Wi-Fi%20Connections.html>
20. Trevor's electronic blog.(n.d). How to get UART serial port working on beaglebone black. Retrieved from <https://electronics.trev.id.au/2018/02/09/get-uart-serial-ports-working-beaglebone-black/>
21. Dorn, C(April 23). Setup a Node MCU to communicate with thingspeak. Retrieved from <https://medium.com/@chardorn/set-up-a-nodemcu-to-communicate-with-thingspeak-using-mqtt-fbad7556a5bc>
22. Technical Hub(2020, April 18). MQTT vs HTTP. Retrieved from <https://www.youtube.com/watch?v=1ujAX-sunQc>
23. Robo india (n.d.).NodeMCU data upload on Thingspeak on Arduino IDE. Retrieved from <https://roboindia.com/tutorials/nodemcu-dht11-thingspeak-data-upload/>
24. Shabaz123. (2019, august 17). Retrieved from <https://github.com/shabaz123/iobb/blob/master/BBBio_lib/BBBiolib.c>
25. Manpreet(2013, Nov 16).16x2 Lcd interfacing in 4 bit mode. Retrieved from <https://learningmsp430.wordpress.com/2013/11/16/16x2-lcd-interfacing-in-4-bit-mode/>
26. Pelayo,R(n.d).Beaglebone black i2c tutorial. Retrieved from <https://www.teachmemicro.com/beaglebone-black-i2c-tutorial/>
27. I2C devices. Enable the I2C devices on beaglebone black. Retrieved from <https://sites.google.com/a/cameon.net/beaglebone/home/i2c-devices>
28. Yoder, M. A.(n.d). EBC I2C. Retrieved from <https://elinux.org/EBC_I2C>
29. EG PRojects.(2019, June 19).Engineers Garage. How to display 5×8 and 5×10 size font characters on 16×2 lcd with 8-bit microcontrollers. Retrieved from <https://www.engineersgarage.com/pic-microcontroller/5x10-size-font-characters-displayed-on-16x2-lcd/>
30. Christian, C(2020, April 15)Element 14.pocketbeagle i2c lcd,16x2, hd44780, PCF8574. Retrieved from <https://www.element14.com/community/thread/75486/l/pocketbeagle-i2c-lcd-16x2-hd44780pcf8574>