I. INTRODUCTION

The application consists of three main modules:

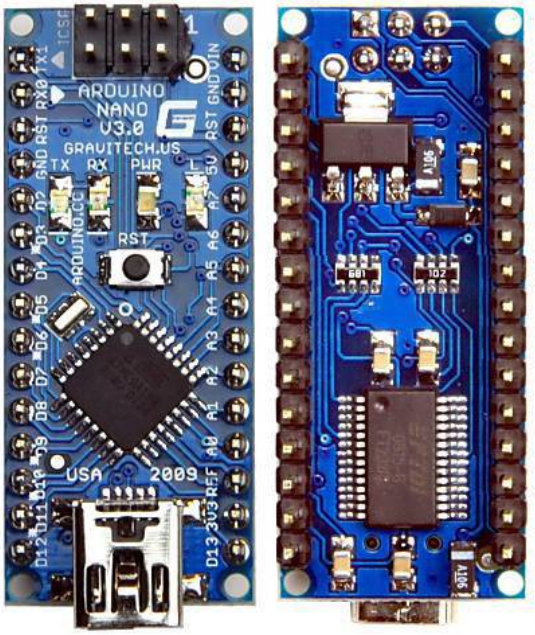


Fig 1:- ATMEGA328P Arduino Nano microcontroller

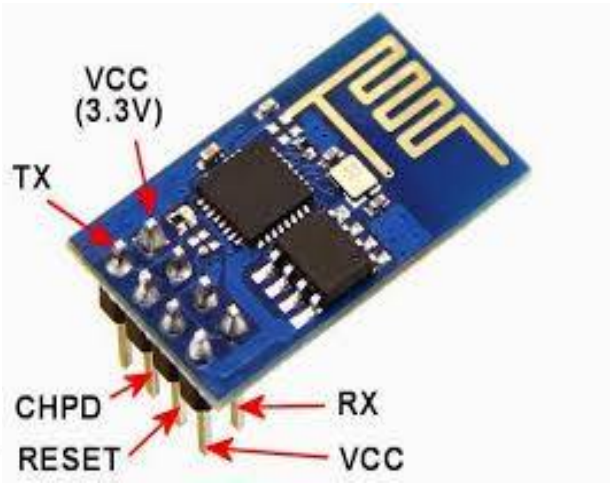


Fig 2:- Wi-Fi ESP 8266

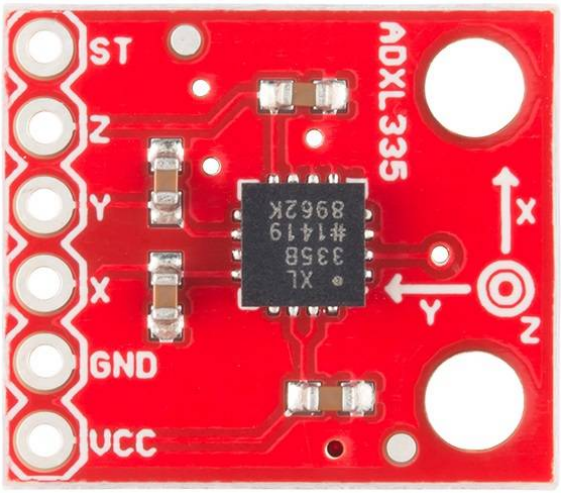


Fig 3:- 3-Axis Analog Accelerometer

Fig 4:- 3-Axis digital

Each of the modules consists of several processes which provide various functionalities.

Safety is a necessary part of man's life. Due to the accident cases reported daily on the major roads in all parts of the developed and developing countries, more attention is needed for research in the designing an efficient car driving aiding system. It is expected that if such a device is designed and incorporated into our cars as a road safety device, it will reduce the incidence of accidents on our roads and various premises, with subsequent reduction in loss of life and property.

However, a major area of concern of an engineer should be safety, as it concerns the use of his/her inventions and the accompanying dangers due to human limitations. When it comes to the use of a motor vehicle, accidents that have occurred over the years tell us that something needs to be done about them from an engineering point of view. According to the 2007 edition of the Small-M report on the road accident statistic in Malaysia, a total of 6,035 people were killed in 2000 and the fatality spring up to 6,287 in 2006 from accident cases reported in 250,429 and 341,252 cases of accident for 2000 and 2006 respectively. In India according to 2016 data analysis, at every 60 sec there were 26 people loss their life in accidents. Suffice to say that the implementation of certain highway safety means such as speed restrictions, among others, has done a lot in reducing the rates of these accidents. The issue here is that policies of safe driving alone would not eradicate this, the engineer has a role to play, after all the main issue is an engineering product (the motor vehicle). Many motorists have had to travel through areas with little light under much fatigue, yet compelled to undertake the journey out of necessity. It is not always irresponsible to do this. A lot of cases reported is as a result of drivers sleeping off while driving, and when he/she eventually woke up, a head-on collision might have taken place. Not many have had the fortune to quickly avert this.

Fig 5:- Car detecting front collision

Fig 6:- Car detecting collision from any side

II. OBJECTIVE OF PROJECT

Safety is a necessary part of every living body's life. Due to the accident cases reported daily on the major roads in all parts of the developed and developing countries, more attention is needed for research in the designing an efficient vehicle driving aiding system. It is expected that if such a device is designed and incorporated into our vehicle as a road safety device, it will reduce the incidence of accidents on our roads and various premises, with subsequent reduction in loss of life and property. A lot of cases reported is as a result of drivers sleeping off while driving, and when he/she eventually woke up, a head-on collision might have taken place. It is therefore important to consider the advantages of an early warning system where the driver is alerted of a possible collision with some considerable amount of time before it occurs.

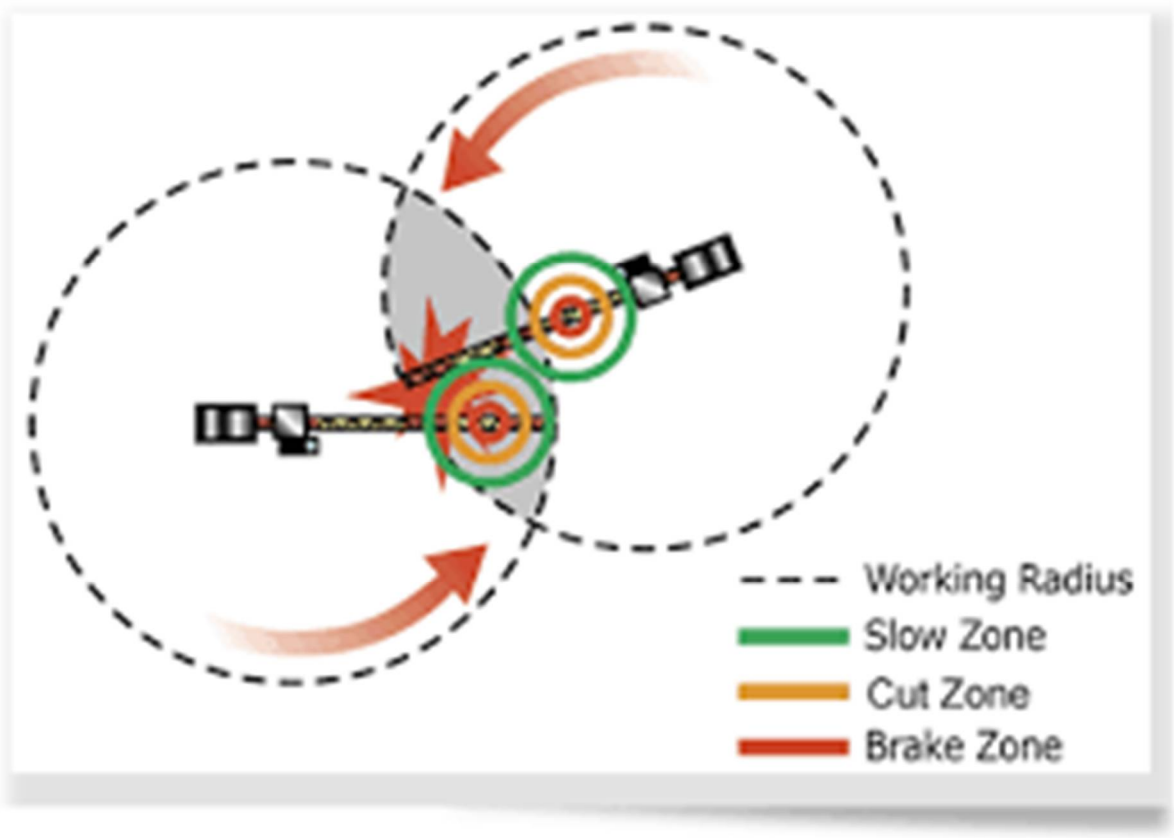


Fig 7:- Collision zone detected by virtual boundary

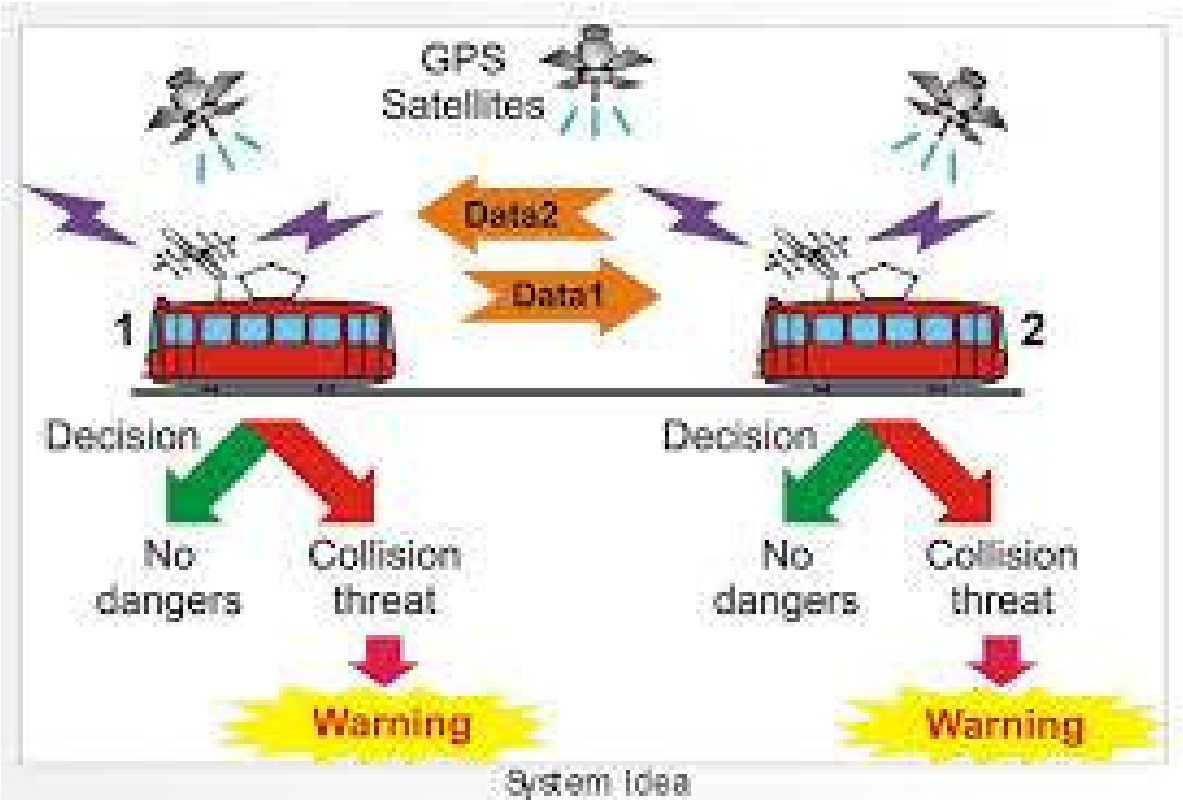


Fig 8:- Identification of collision and its probability

III. JUSTIFICATION OF PROJECT

In our project, we are going to make a system as described below: -

i. Using A-GPS we can get current location of vehicle, living body or devices.

ii. According to the dimension of object our system will create a protective virtual wall.

iii. In our system, every device will be programmed to give them artificial intelligence.

iv. In our project, every device will have intelligent feature to communicate with each other to get information of moving or static vehicle/devices.

v. Our project can prevent collision between vehicle, human body and vehicle etc.

vi. This project can be installed easily.

IV. METHODOLOGY

i. A-GPS will track the location of devices in every second, and will send data to microcontroller.

ii. Using that A-GPS microcontroller will be programmed to useful data like speed of moving vehicle, direction of motion, Distance etc.

iii. A microcontroller will be programmed to create protective virtual boundary.

iv. A microcontroller will try to communicate with another microcontroller using peer to peer multipoint connection to get data of another microcontroller/devices.

v. This connection will be established by using Wi-Fi local host communication.

vi. Since connection is peer to peer and multipoint so the range of system will increase n-times.

vii. Using this data every device/vehicle will try to prevent that virtual boundary from collision, and hence no one will collide.

V. ADOPTED, SYSTEM IMPLEMENTATION & DETAILS OF HARDWARE & SOFTWARE USED

5.1. HARDWARE USED:

5.1.1. ATMEGA328P microcontroller:-

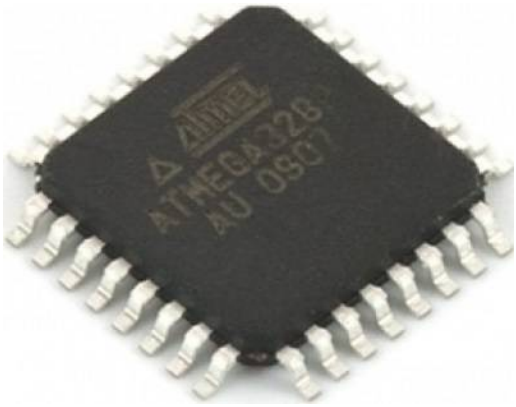


Fig 9:- ATMEGA328 QIP (Quad in Package) IC

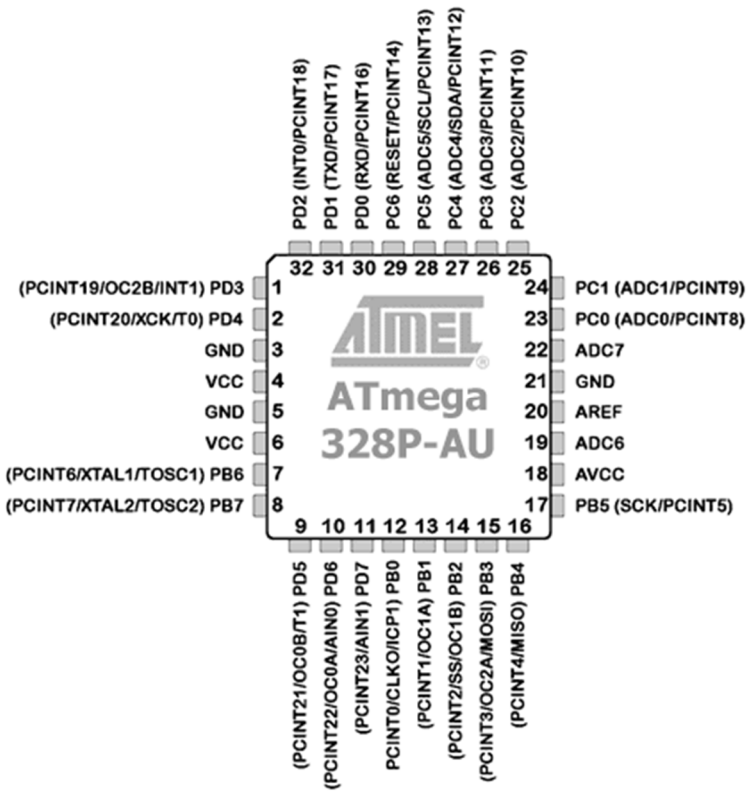


Fig 10:- ATMEGA328 QIP (Quad in Package) IC pin detail

5.1.2. Arduino Nano (ATMEGA328P module):-

The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328. The Arduino Nano is programmed using the Arduino IDE (Integrated Development Environment).

The ATmega328 has 32 KB, (also with 2 KB used for the boot loader). The ATmega328 has 2 KB of SRAM and 1 KB of EEPROM. The Arduino Nano can be powered via the Mini-B USB connection, 6-20V unregulated external power supply (pin 30), or 5V regulated external power supply (pin 27). The power source is automatically selected to the highest voltage source.

Each of the 14 digital pins on the Nano can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 KOhms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the FTDI USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Nano has 8 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the analogReference() function. Additionally, some pins have specialized functionality:

- I2C: A4 (SDA) and A5 (SCL). Support I2C (TWI) communication using the Wire library (documentation on the Wiring website).

There are a couple of other pins on the board:

- AREF. Reference voltage for the analog inputs. Used with analogReference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

The Arduino Nano has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provide UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An FTDI FT232RL on the board channels this serial communication over USB and the FTDI drivers (included with the Arduino software) provide a virtual com port to software on the computer. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the FTDI chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial.h library allows for serial communication on any of the Nano's digital pins.

The ATmega168 and ATmega328 also support I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. To use the SPI communication.

Power Source:

The Arduino Nano can be powered via the mini-B USB connection, 6-20V unregulated external power supply (pin 30), or 5V regulated external power supply (pin 27). The power source is automatically selected to the highest voltage source.

Table 1:- Features of ATMEGA328P IC

Operating Voltage (logic level)	5 V
Input Voltage (recommended)	7-12 V
Input Voltage (limits)	6-20 V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	8
DC Current per I/O Pin	40 mA
Flash Memory	32 KB (of which 2KB used by bootloader)
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz

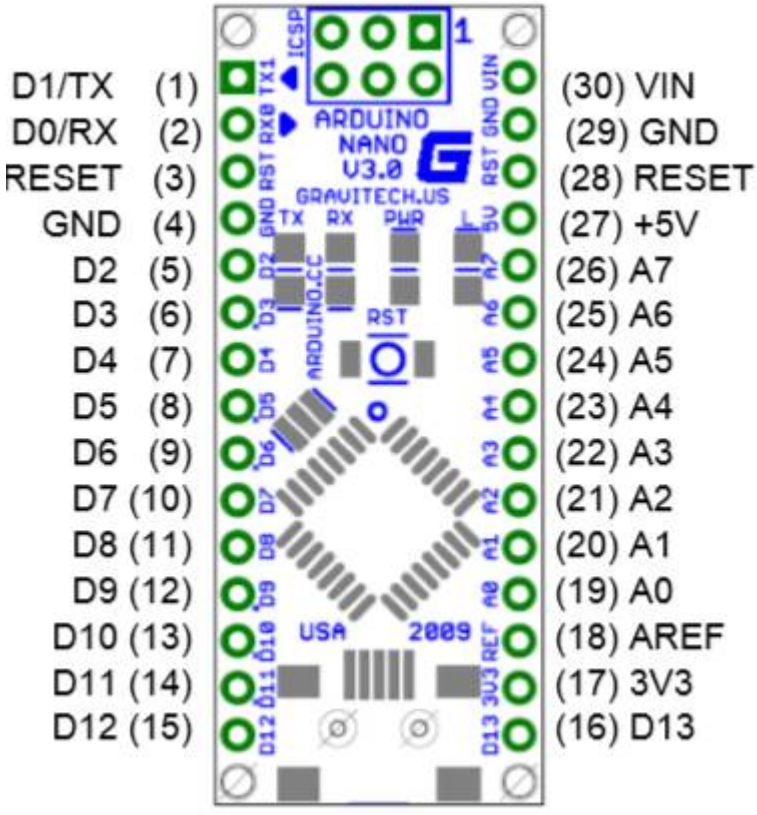


Fig 11:- Arduino Nano pin detail

Table 2:- Pin Description

Pin No.	Name	Type	Description
1-2, 5-16	D0-D13	I/O	Digital input/output port 0 to 13
3, 28	RESET	Input	Reset (active low)
4, 29	GND	PWR	Supply ground
17	3V3	Output	+3.3V output (from FTDI)
18	AREF	Input	ADC reference
19-26	A0-A7	Input	Analog input channel 0 to 7
27	+5V	Output or Input	+5V output (from on-board regulator) or +5V (input from external power supply)
30	VIN	PWR	Supply voltage

5.1.3. Accelerometer:

An accelerometer is a device that measures proper acceleration; proper acceleration is not the same as coordinate acceleration (rate of change of velocity). They measure in meters per second squared (m/s2) or in G-forces (g). A single G-force for us here on planet Earth is equivalent to 9.8 m/s2, but this does vary slightly with elevation (and will be a different value on different planets due to variations in gravitational pull). Accelerometers are useful for sensing vibrations in systems or for orientation applications.

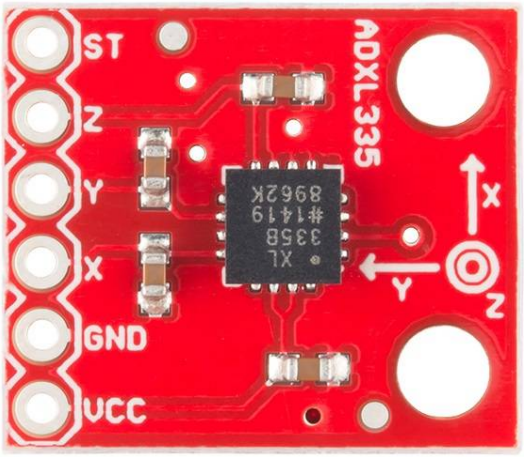


Fig 12:- 3-Axis analog Accelerometer

Fig 13:- 3-Axis digital Accelerometer

Range of Accelerometer:

Most accelerometers will have a selectable range of forces they can measure. These ranges can vary from ±1g up to ±250g. Typically, the smaller the range, the more sensitive the readings will be from the accelerometer. For example, to measure small vibrations on a table top, using a small-range accelerometer will provide more detailed data than using a 250g range (which is more suited for rockets).

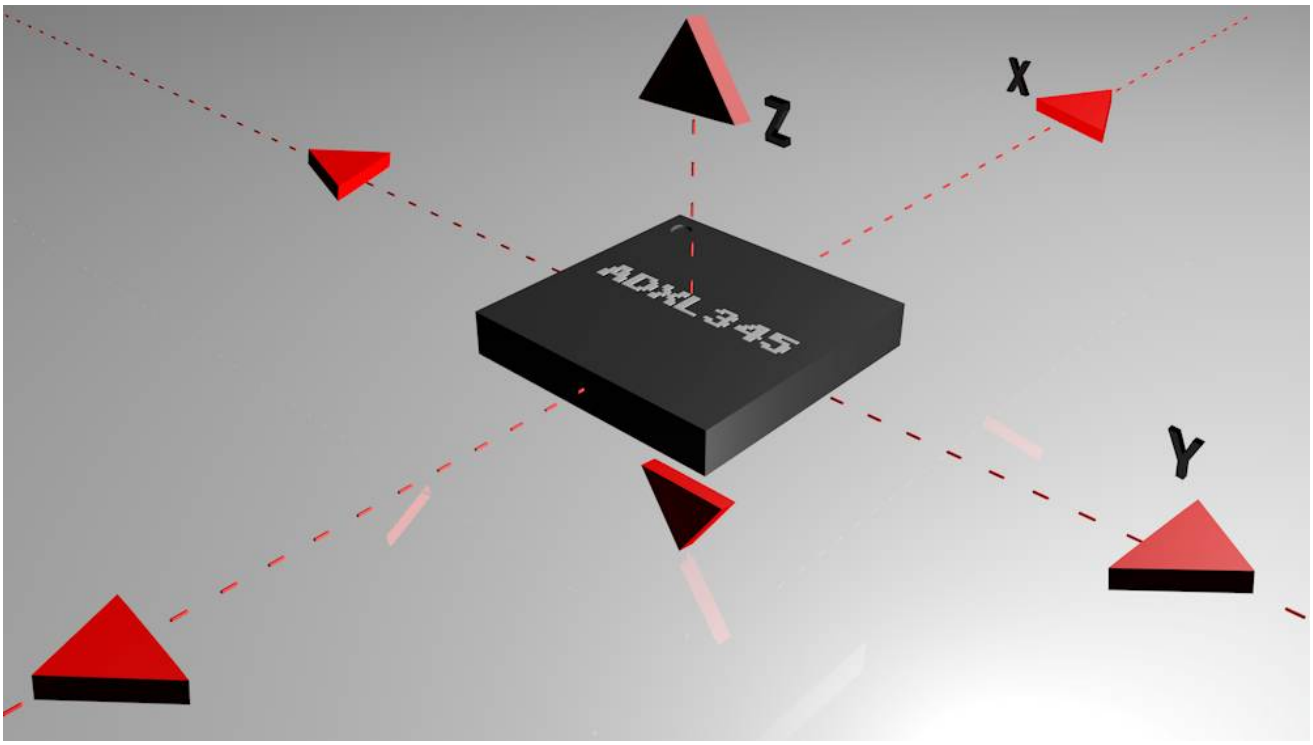


Fig 14:- ADXL345 axis

Communication Interface:

Accelerometers will communicate over an analog, digital, or pulse-width modulated connection interface.

- Accelerometers with an analog interface show accelerations through varying voltage levels. These values generally fluctuate between ground and the supply voltage level. An ADC on a microcontroller can then be used to read this value. These are generally less expensive than digital accelerometers.

- Accelerometers with a digital interface can either communicate over SPI or I2C communication protocols. These tend to have more functionality and be less susceptible to noise than analog accelerometers.
- Accelerometers that output data over pulse-width modulation (PWM) output square waves with a known period, but a duty cycle that varies with changes in acceleration.

Power:

Accelerometers are generally low-power devices. The required current typically falls in the micro (µ) or mille-amp range, with a supply voltage of 5V or less. The current consumption can vary depending on the settings (e.g., power saving mode versus standard operating mode). These different modes can make accelerometers well suited for battery powered applications.

V.1.4. Wi-Fi module ESP8266:-

The ESP8266 Wi-Fi Module is a self-contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to your Wi-Fi network. The ESP8266 is capable of either hosting an application or offloading all Wi-Fi networking functions from another application processor. Each ESP8266 module comes pre-programmed with an AT command set firmware, meaning, you can simply hook this up to your Arduino device and get about as much Wi-Fi-ability as a Wi-Fi Shield offers (and that's just out of the box)! The ESP8266 module is an extremely cost effective board with a huge, and ever growing, community.

This module has a powerful enough on-board processing and storage capability that allows it to be integrated with the sensors and other application specific devices through its GPIOs with minimal development up-front and minimal loading during runtime. Its high degree of on-chip integration allows for minimal external circuitry, including the front-end module, is designed to occupy minimal PCB area. The ESP8266 supports APSD for VoIP applications and Bluetooth co-existence interfaces, it contains a self-calibrated RF allowing it to work under all operating conditions, and requires no external RF parts.

There is an almost limitless fountain of information available for the ESP8266, all of which has been provided by amazing community support. In the Documents section below you will find many resources to aid you in using the ESP8266, even instructions on how to transforming this module into an IoT (Internet of Things) solution, but the ESP8266 Module is not capable of 5-3V logic shifting and will require an external Logic Level Converter. Please do not power it directly from 5V.

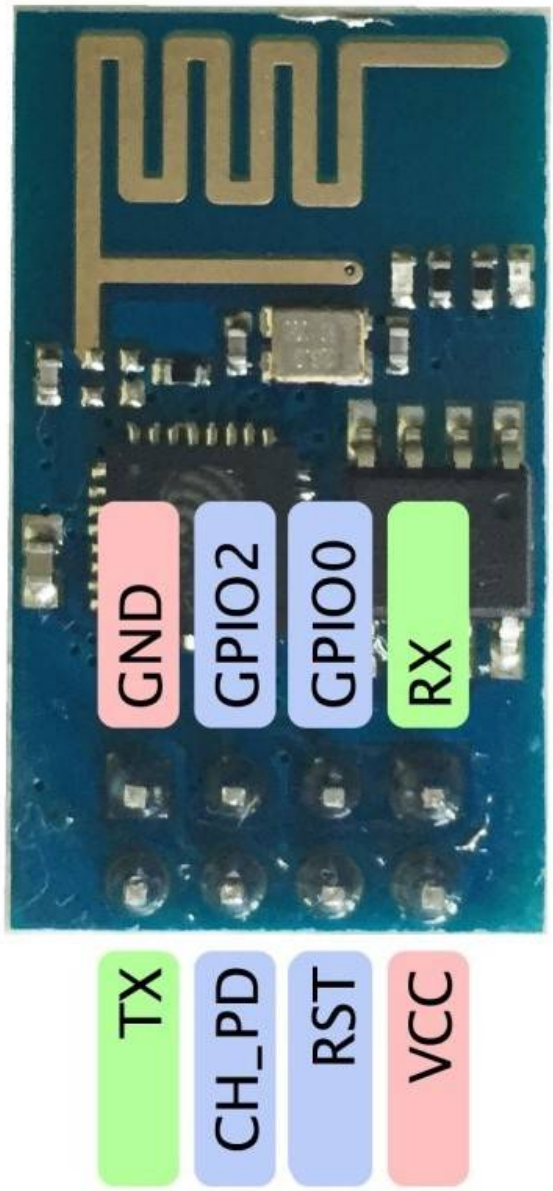


Fig 15:- ESP 8266 Wi-Fi Module

5.2. SOFTWARE USED:

5.2.1. Arduino IDE (Integrated Development Environment):

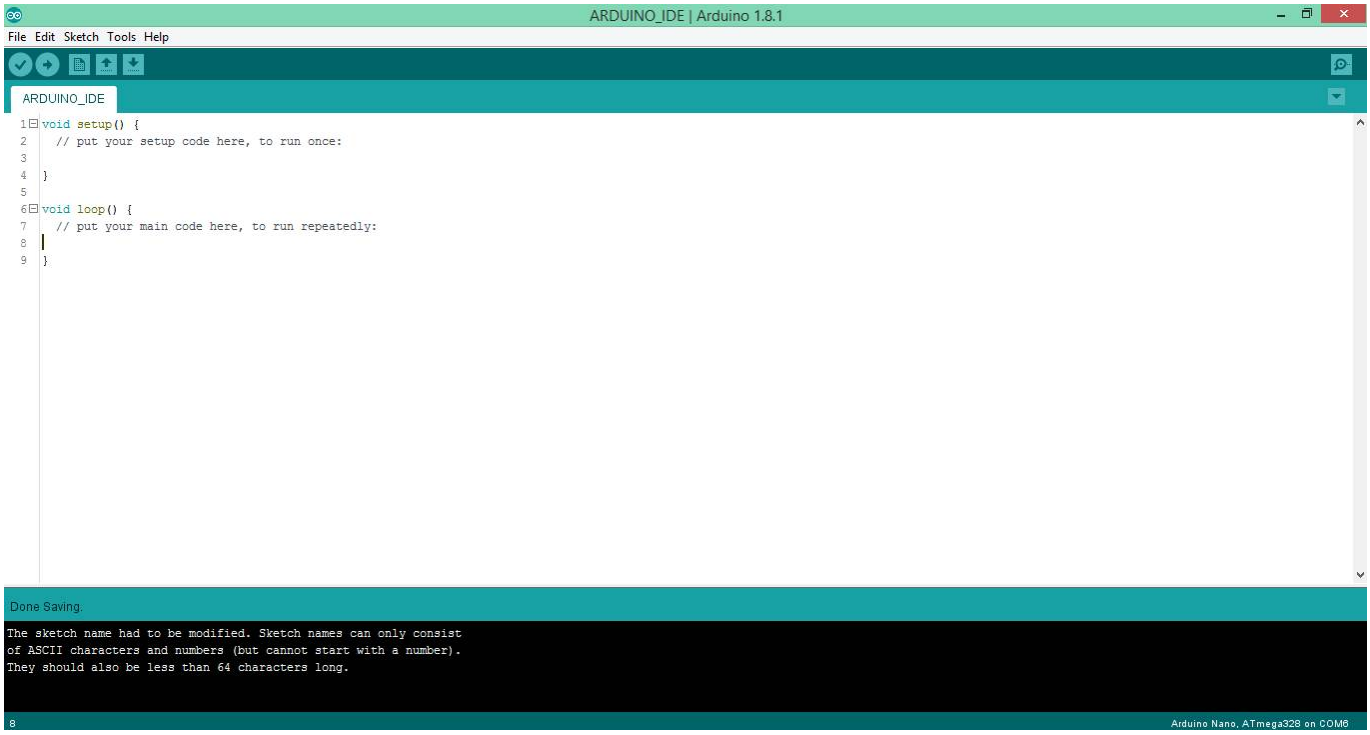


Fig 16:- Arduino IDE (Integrated Development Environment)

The Arduino Software (IDE) allows us to write programs and upload them to the board. The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software. This software can be used with any Arduino board.

The Arduino IDE has a feature that can be a great help in debugging sketches or controlling Arduino from your computer's keyboard.

The Serial Monitor is a separate pop-up window that acts as a separate terminal that communicates by receiving and sending Serial Data. See the icon on the far right of the image above.

Serial Data is sent over a single wire (but usually travels over USB in our case) and consists of a series of 1's and 0's sent over the wire. Data can be sent in both directions (In our case on two wires).

5.2.2. Arduino IDE as Serial Monitor:-

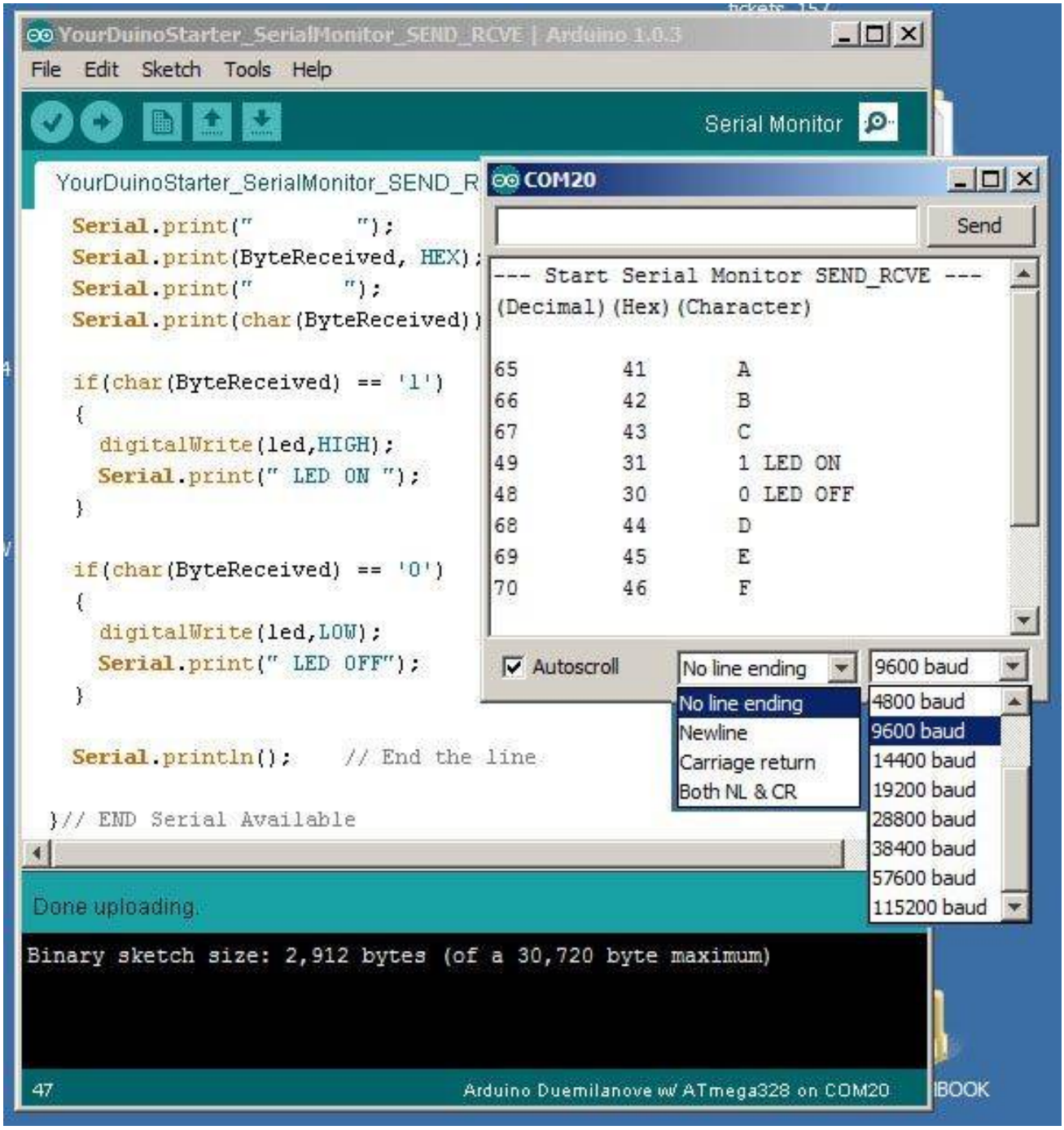


Fig 17:- Arduino IDE (Integrated Development Environment) with Serial Monitor

The Arduino IDE has a feature that can be a great help in debugging sketches or controlling Arduino from computer's keyboard.

The Serial Monitor is a separate pop-up window that acts as a separate terminal that communicates by receiving and sending Serial Data. See the icon on the far right of the image above.

Serial Data is sent over a single wire (but usually travels over USB in our case) and consists of a series of 1's and 0's sent over the wire. Data can be sent in both directions (In our case on two wires).

SETUP:
In Setup you need to begin Serial Communications and set the Baud Rate (speed) that data will be transferred at. That looks like this:

```
Serial.begin(9600); // Other baud rates can be used...
Serial.println("My Sketch has started");
```
The second line is optional...

LOOP:
Here we can print helpful info to the Serial Monitor. Examples:

```
Serial.println("Top of loop");
Serial.println("Reading Temperature Sensor");
Serial.print("LoopCounter value = ");
Serial.println(LoopCounter);
```

5.2.3. Docklight Serial monitor:-

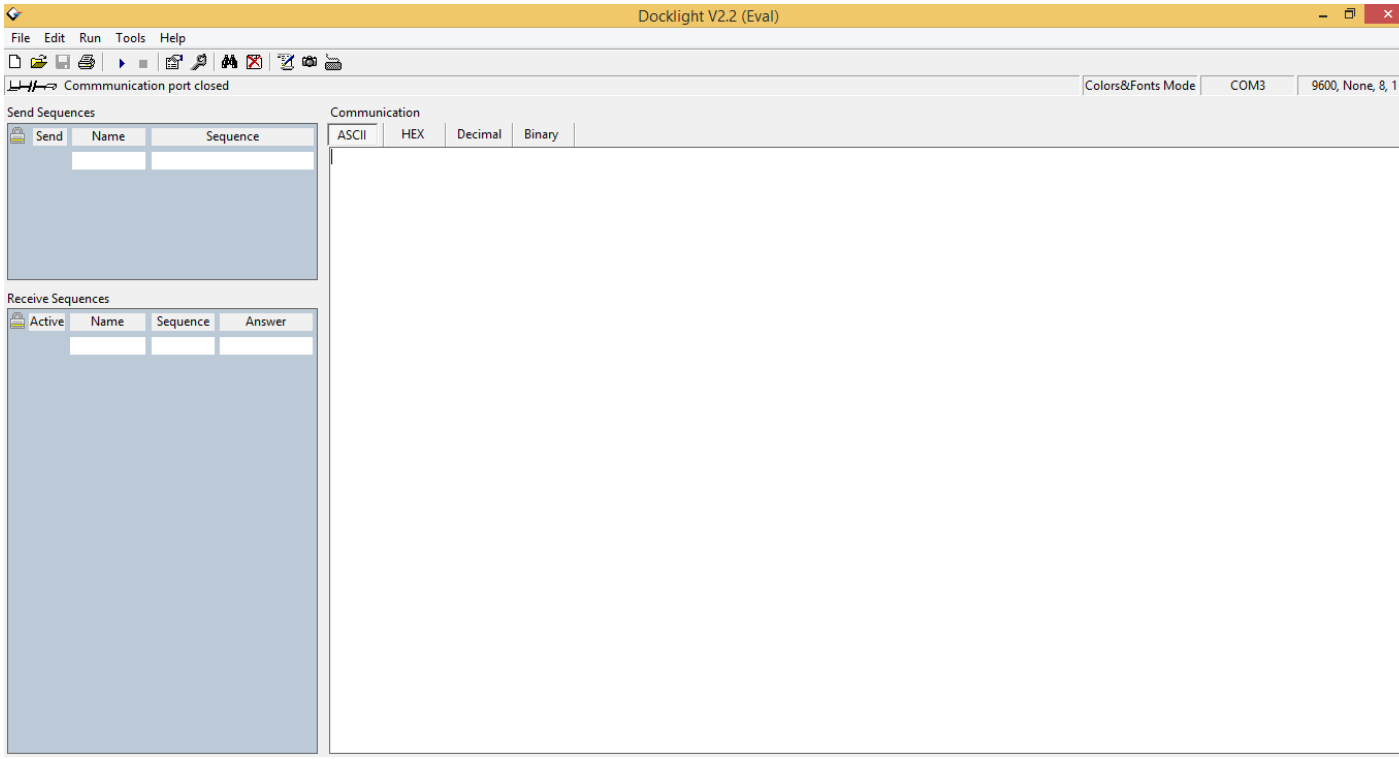


Fig 18:- Docklight Serial Monitor

Docklight is a testing, analysis and simulation tool for serial communication protocols. It allows you to monitor the communication between two serial devices or to test the serial communication of a single device.

VI. UART COMMUNICATION WITH Wi-Fi ESP 8266

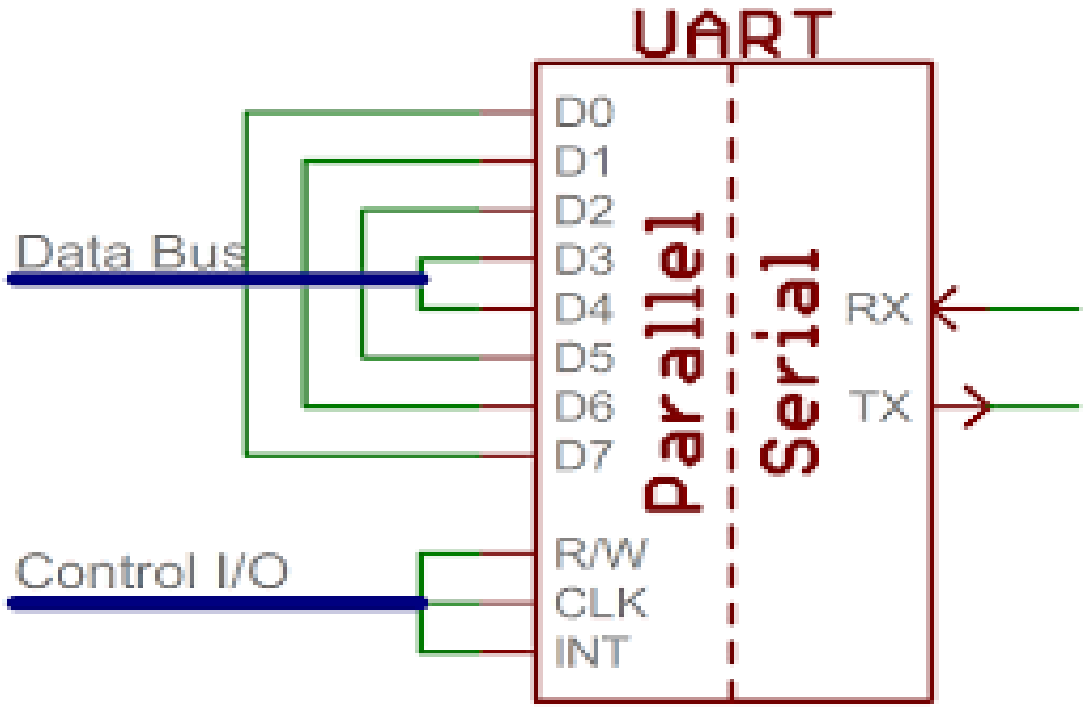


Fig 19:- Parallel to Serial Communication

The universal asynchronous receiver/transmitter (UART) takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Each UART contains a shift register which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is less costly than parallel transmission through multiple wires. UARTs are commonly used in conjunction with communication standards such as TIA. UART is usually an individual (or part of an) integrated circuit (IC) used for serial communication over a computer or peripheral device serial port. UARTs are now commonly included in microcontrollers. A dual UART, or DUART, combines two UARTs into a single chip.

The ESP8266 is a low cost Serial-to-Wi-Fi module that interfaces nicely to any microcontroller. However, a word of caution -- it is highly undocumented (primary reason for writing this document), and more importantly, it is frequently updated and not backward compatible. A good example is how newer versions use 9600 baud rate, while older versions used 57600-115200 baud rates.

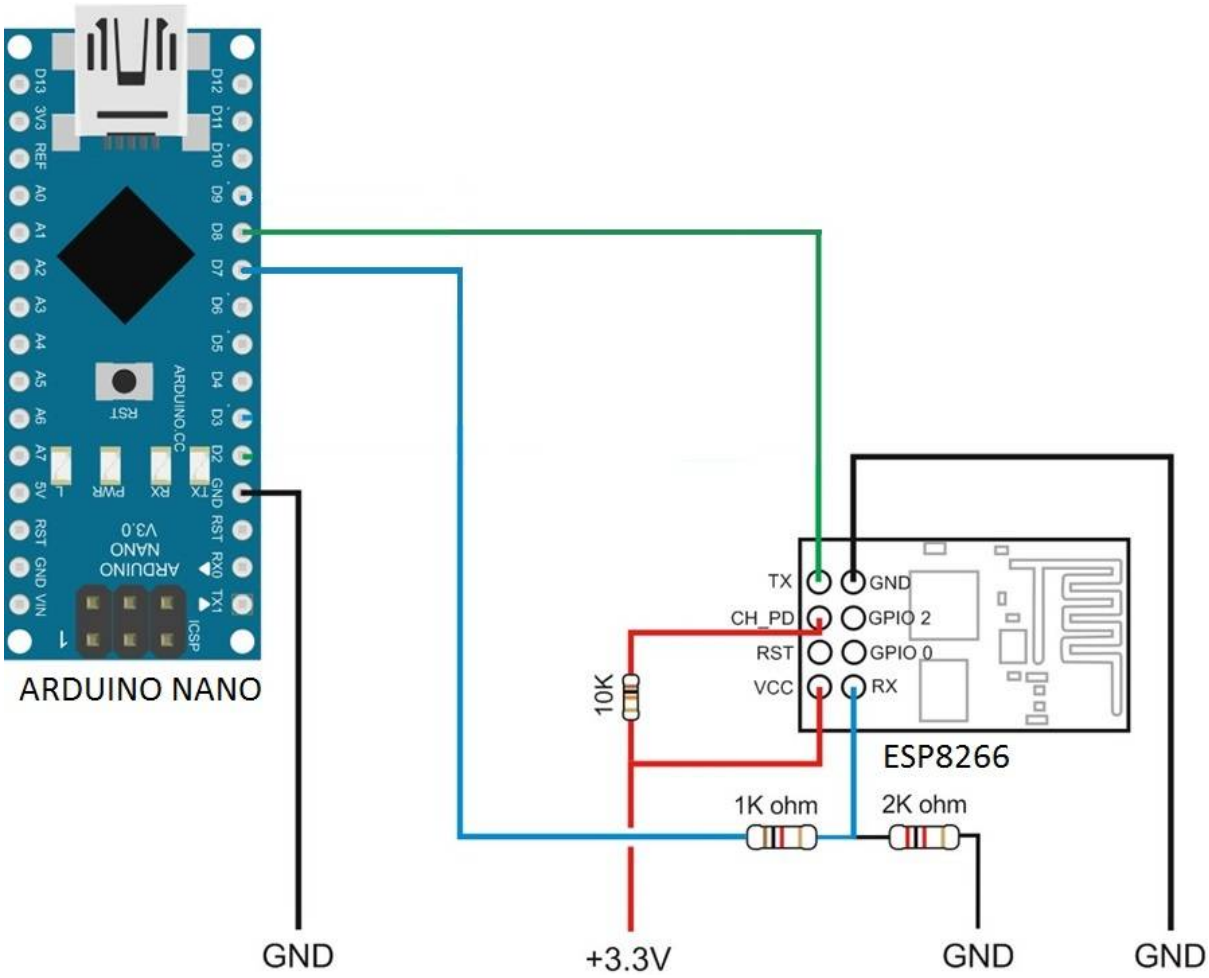


Fig 20:- Arduino Nano Interfacing with ESP8266 Module

Program 1:-

```
#include<SoftwareSerial.h>
SoftwareSerial wifi(7, 8);//Rx,Tx
void setup()
{
  wifi.begin(9600);
  Serial.begin(9600);
  wifi.println("AT");
  Serial.println(wifi.readString());
}
```

```
void loop()
{
  // put your main code here, to run repeatedly:
  while (wifi.available())
  {
    Serial.println("reading ......");
    String st = "";
    st = wifi.readString();
    for (int i = 0; st[i] != NULL; i++)
{
      Serial.print("st[");
      Serial.print(i);
      Serial.print("]= ");
      Serial.println(st[i]);
    }
    Serial.print(st);
    Serial.println();
    st = "";
  }
  while (Serial.available())
{
    Serial.println("writing...");
    String st1 = "";
    st1 = Serial.readString();
    Serial.println(st1);
    wifi.println(st1);
    st1 = "";
  }
}
```

OUTPUT:-

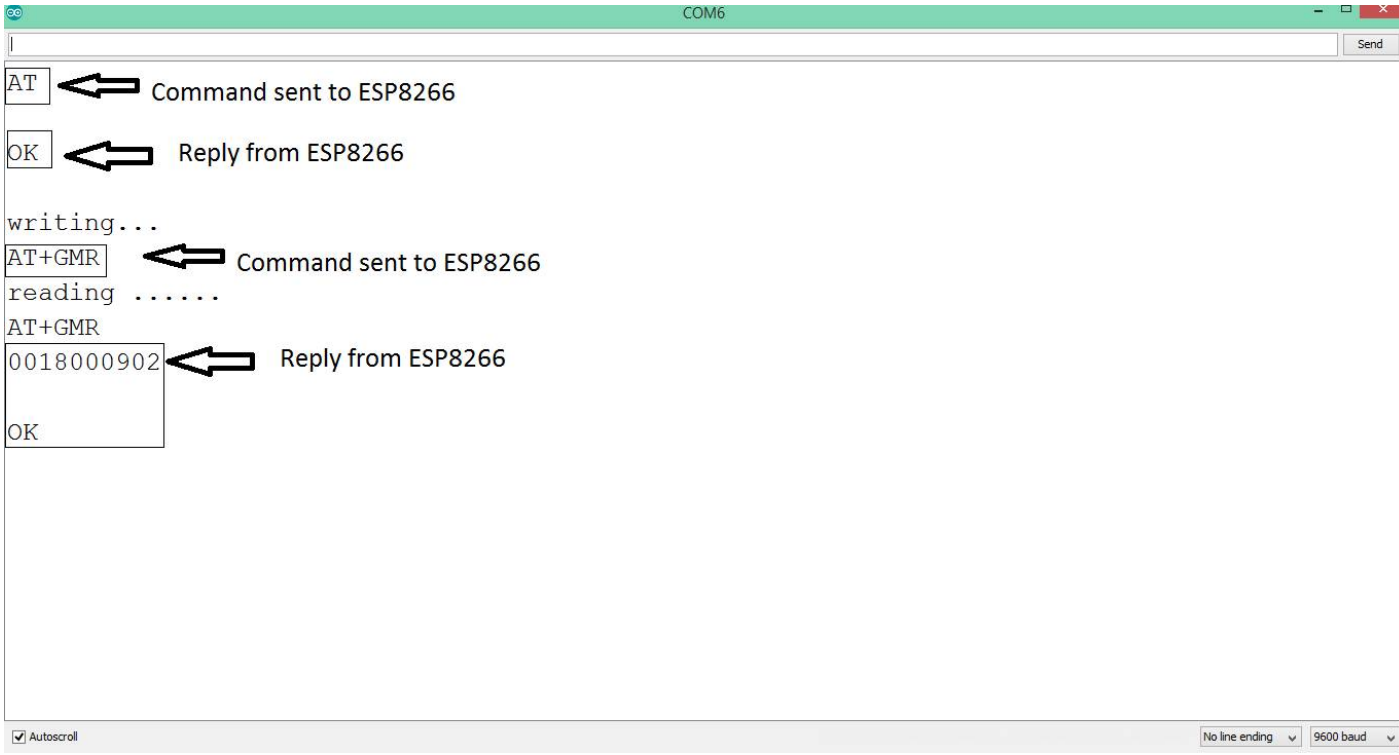


Fig 21:- OUTPUT of Program1

Major Applications:

First, it is important to understand how the board works. The ESP8266 has a full TCP/UDP stack support. It can also be easily configured as a web server. The module accepts commands via a simple serial interface. It then responds back with the operation's outcome (assuming everything is running correctly). Also, once the device is connected and is set to accept connections, it will send unsolicited messages whenever a new connection or a new request is issued.

Major fields of ESP8266EX applications to Internet-of-Things include:

• Home Appliances

• Home Automation

• Smart Plug and lights

• Mesh Network

• Industrial Wireless Control

• Baby Monitors

• IP Cameras

• Sensor Networks

• Wearable Electronics

• Wi-Fi Location-aware Devices

• Security ID Tags

• Wi-Fi Position System Beacons

VII. WIFI 'AT' COMMUNICATION

Wi-Fi AT command ESP8266 offers a complete and self-contained Wi-Fi networking solution, allowing it to either host the application or to offload all Wi-Fi networking functions from another application processor. When ESP8266 hosts the application, and when it is the only application processor in the device, it is able to boot up directly from an external flash. It has integrated cache to improve the performance of the system in such applications, and to minimize the memory requirements. Alternately, serving as a Wi-Fi adapter, wireless internet access can be added to any microcontroller-based design with simple connectivity through UART interface or the CPU AHB bridge interface.ESP8266 on-board processing and storage capabilities allow it to be integrated with the sensors and other application specific devices through its GPIOs with minimal development up-front and minimal loading during runtime. With its high degree of on-chip integration, which includes the antenna switch, power management converters, it requires minimal external circuitry, and the entire solution, including front-end module, is designed to occupy minimal PCB area.

Sophisticated system-level features include fast sleep/wake context switching for energy-efficient VoIP, adaptive radio biasing for low-power operation, advance signal processing, and spur cancellation and radio co-existence features for common cellular, Bluetooth, DDR, LVDS, LCD interference mitigation. A really cheap and easy way to connect any small microcontroller platform (for example Arduino) wirelessly to Internet. The ESP8266 is now one of the leading platforms for the Internet of Things. It's super cheap, and super easy to work with. This is a serial module with a built-in TCP/IP stack, so you can use it standalone. We can use AT commands to connect with Wi-Fi networks and open TCP connections without need to have TCP/IP stack running in your own microcontroller: You can simply connect any microcontroller to ESP module and start pushing data up to internet.

The ESP8266 wireless Wi-Fi modules can be driven via the serial interface using the standard AT commands. Here is a list of some basic AT commands that can be used.

Table :-

Basic	
Command	**Description**
AT	Test AT startup
AT+RST	Restart module
AT+GMR	View version info
AT+GSLP	Enter deep-sleep mode
ATE	AT commands echo or not
AT+RESTORE	Factory Reset
AT+UART	UART configuration, [@deprecated]
AT+UART_CUR	UART current configuration
AT+UART_DEF	UART default configuration, save to flash
AT+SLEEP	Sleep mode
AT+RFPOWER	Set maximum value of RF TX Power
AT+RFVDD	Set RF TX Power according to VDD33

Some AT commands and their outputs are given below:-

- **AT – Test AT startup**

  The type of this command is "executed". It is used to test the setup function of your wireless Wi-Fi module.

AT - Test AT startup	
Response	OK
Parameters	null

- **AT+RST – Restart module**

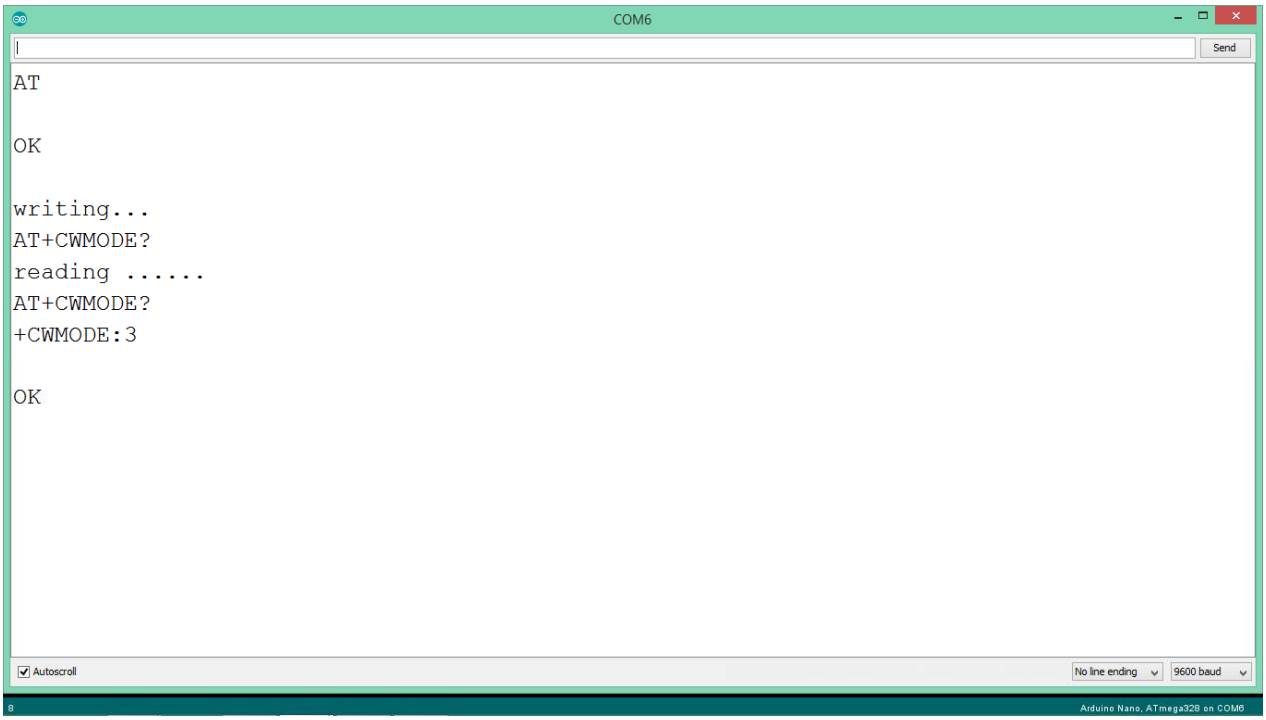
  The type of this command is "executed". It's used to restart the module.

AT+RST - Restart module	
Response	OK
Parameters	null

- **AT+CWMODE – Wi-Fi mode**

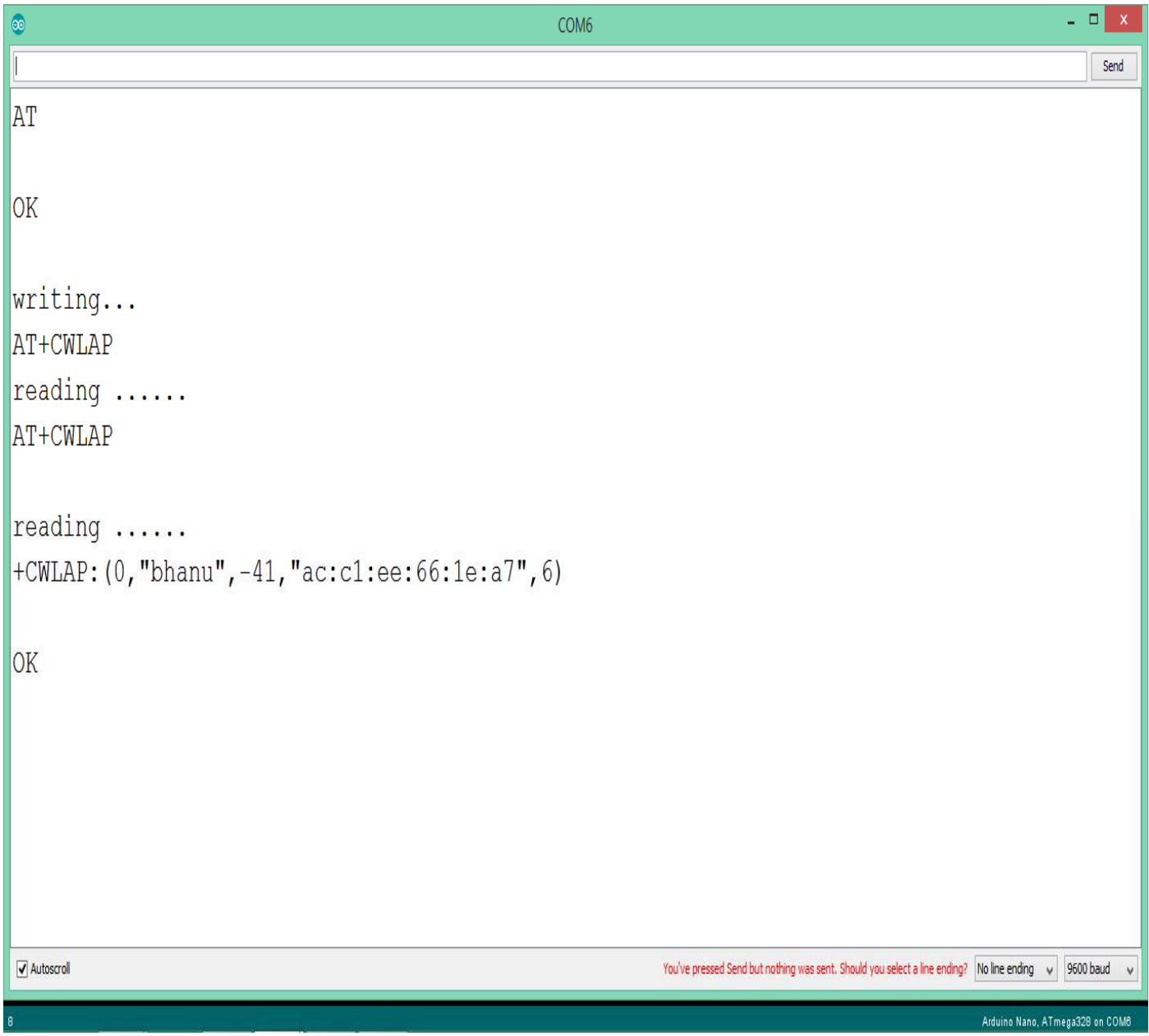
  The function of this AT command is to get the value scope of Wi-Fi mode, including station mode, softAP mode, and station+softAP mode, enquiry about the information of Wi-Fi mode, or set the Wi-Fi mode

AT+CWMODE - WiFi mode	
This command is deprecated. Please use AT+CWMODE_CUR or AT+CWMODE_DEF instead.	
Command	**AT+CWMODE=?**
Response	+CWMODE:(value scope of <mode>) OK
Parameters	Please refer to AT command settings.
Command	**AT+CWMODE?**
Response	+CWMODE:<mode> OK
Parameters	Please refer to AT command settings.
Command	**AT+CWMODE=<mode>**
Response	OK
Parameters	<mode> 1 : station mode 2 : softAP mode 3 : softAP + station mode
Notes	This setting will be stored in the flash system parameter area. It won't be erased even when the power is off and restarted.


COM6
Send
AT
OK
writing...
AT+CWMODE?
reading ......
AT+CWMODE?
+CWMODE:3
OK
Autoscroll
No line ending
9600 baud
Arduino Nano, ATmega328 on COM6

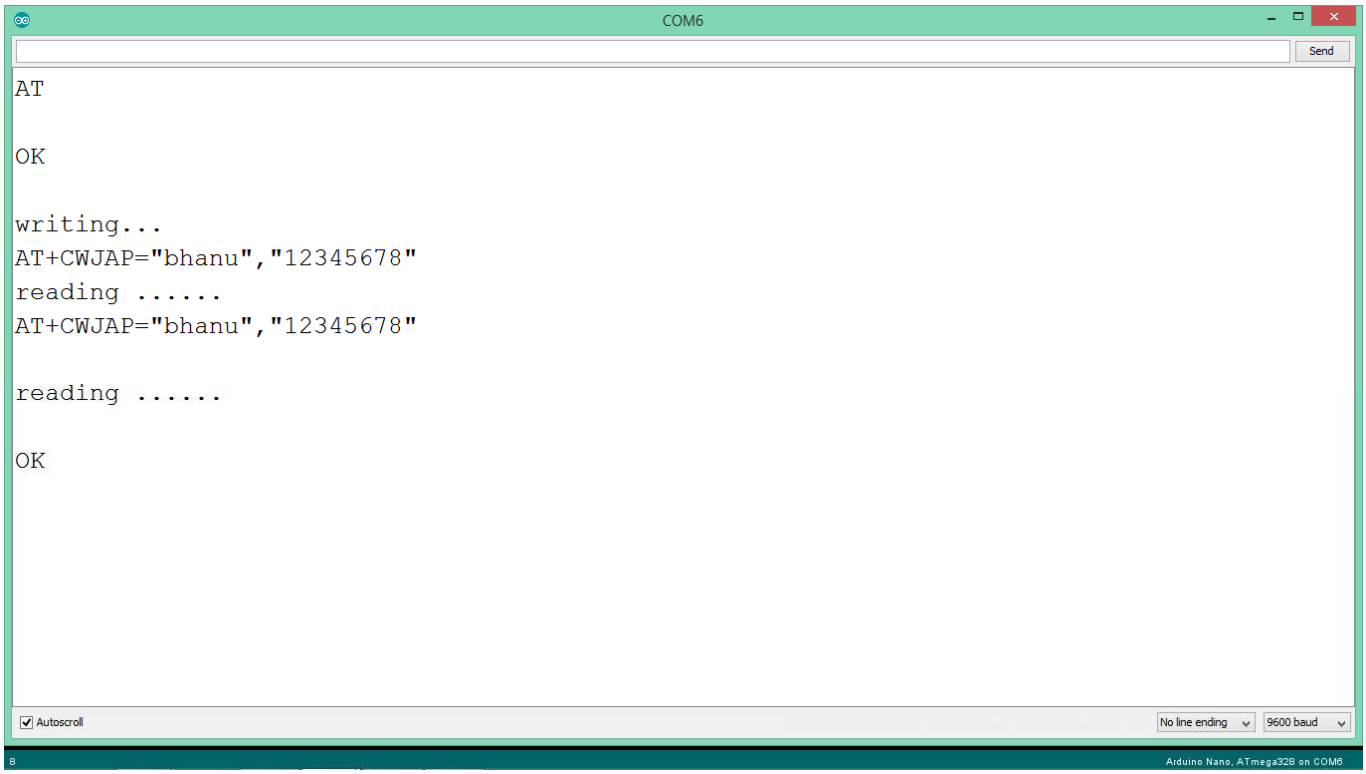
❖ **AT+CWLAP – List available Access Points**

AT+CWLAP - Lists available APs	
Example	• AT+CWLAP List of all available AP's detected by ESP8266 • AT+CWLAP="WiFi", "ca:d7:19:d8:a6:44", 6 Find AP with specific SSID and MAC at specific channel. • AT+CWLAP="WiFi" Find AP with specific SSID
Command	**AT+CWLAP=<ssid>[, <mac>, <ch>]**
Response	+CWLAP:<ecn>, <ssid>, <rssi>, <mac>, <ch>, <freq offset>, <freq calibration> OK
Parameters	<ecn> 0 : OPEN 1 : WEP 2 : WPA_PSK 3 : WPA2_PSK 4 : WPA_WPA2_PSK 5 : WPA2_Enterprise (AT can NOT connect to WPA2_Enterprise AP) <ssid> string, SSID of AP <rssi> signal strength <mac> string, MAC address <freq offset> frequency offset of AP, unit: KHz. The value of <freq offset> / 2.4 to get the value as ppm <freq calibration> calibration for frequency offset
Command	**AT+CWLAP**
Response	+CWLAP:<ecn>, <ssid>, <rssi>, <mac>, <ch>, <freq offset>, <freq calibration> OK
Parameters	The same as above


COM6
Send
AT
OK
writing...
AT+CWLAP
reading ......
AT+CWLAP
reading ......
+CWLAP:(0,"bhanu",-41,"ac:c1:ee:66:1e:a7",6)
OK
Autoscroll
You've pressed Send but nothing was sent. Should you select a line ending?
No line ending
9600 baud
Arduino Nano, ATmega328 on COM6

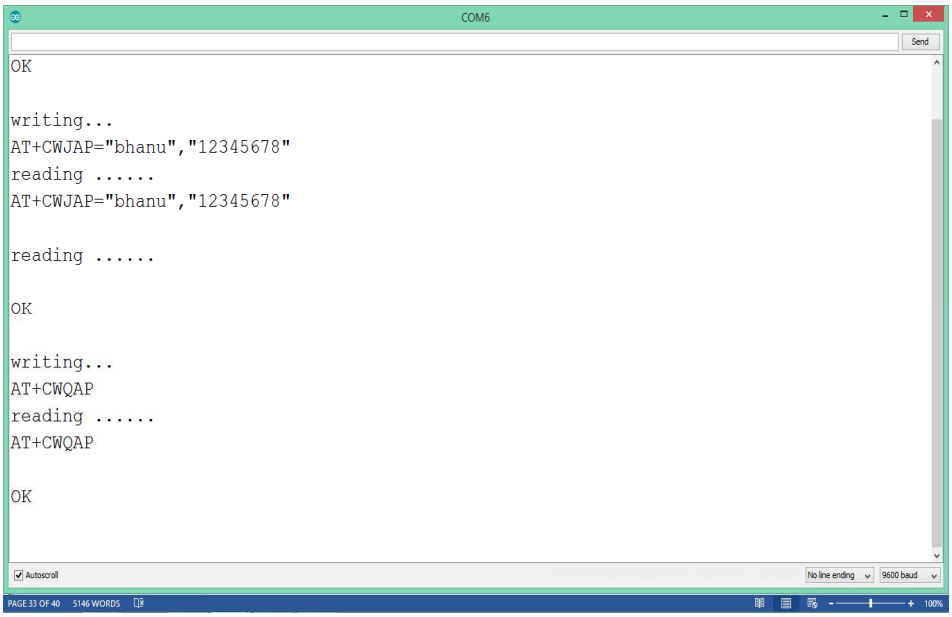
❖ **AT+CWJAP – Connect to AP (Access Piont)**

Example	• AT+CWJAP ="abc", "0123456789" • If SSID is "ab\, c" and password is "0123456789"\" AT+CWJAP ="ab\\\, c", "0123456789\"\\" • If several APs have the same SSID as "abc", target AP can be found by bssid: AT+CWJAP ="abc", "0123456789", "ca:d7:19:d8:a6:44"
Command	**AT+CWJAP?**
Response	+CWJAP:<ssid>, <bssid>, <channel>, <rssi> OK
Parameters	<ssid> string, AP's SSID
Command	**AT+CWJAP=<ssid>, <pwd>[, <bssid>]**
Response	OK or +CWJAP:<error code> FAIL
Parameters	<ssid> string, AP's SSID <pwd> string, MAX: 64 bytes ASCII [<bssid>] string, AP's MAC address, for several APs may have the same SSID <error code> only for reference, it is not reliable 1 : connection timeout 2 : wrong password 3 : cannot find target AP 4 : connection failed This command requires station mode to be active. Escape character syntax is needed if "SSID" or "password" contains any special characters (' , ' or ' " 'or ' \ ')
Notes	Configuration changes will be stored in flash system parameter area.



❖ AT+CWQAP – Disconnect from AP

AT+CWQAP - Disconnect from AP	
Command	AT+ CWQAP
Response	OK
Parameters	null



❖ **AT+CWSAP – Configuration of softAP mode**

AT+ CWSAP - Configuration of softAP mode [@deprecated]. Please use AT+CWSAP_CUR or AT+CWSAP_DEF instead.	
Example	AT+CWSAP="ESP8266", "1234567890", 5, 3
Command	**AT+CWSAP?**
Response	+CWSAP:<ssid>, <pwd>, <chl>, <ecn>, <max conn>, <ssid hidden>
Parameters	<ssid> string, ESP8266 softAP's SSID <pwd> string, range: 8 ~ 64 bytes ASCII <chl> channel ID <ecn> 0 : OPEN 2 : WPA_PSK 3 : WPA2_PSK 4 : WPA_WPA2_PSK <max conn> maximum count of stations that are allowed to connect to ESP8266 soft-AP range: [1, 4] <ssid hidden> Broadcast SSID by default 0 : broadcast SSID of ESP8266 soft-AP 1 : do not broadcast SSID of ESP8266 soft-AP
Command	**AT+CWSAP=<ssid>, <pwd>, <chl>, <ecn>[, <max conn>][, <ssid hidden>]**
Response	OK
Parameters	The same as above.
Notes	This CMD is only available when softAP is active. ESP8266 softAP does not support WEP. Configuration changes will be stored in flash system parameter area.

❖ **AT+CWLIF – IP of stations**

This command is used to get the IP of stations that are connected to ESP8266 softAP.

AT+ CWLIF- IP of stations which are connected to ESP8266 softAP	
Response	<IP addr>, <mac> OK
Parameters	<IP addr> IP address of stations which are connected to ESP8266 softAP <mac> MAC address of stations which are connected to ESP8266 softAP
Notes	This command cannot get static IP, it is only available if DHCP is enabled.

❖ **AT+CIPSTA**

Set IP address of station Only after ESP8266 station is connected to an AP, station IP can be obtained and inquired. Configuration changes will be stored in flash user parameter area.

AT+ CIPSTA - Set IP address of ESP8266 station [@deprecated]. Please use AT+CIPSTA_CUR or AT+CIPSTA_DEF instead.	
Example	AT+CIPSTA="192.168.6.100", "192.168.6.1", "255.255.255.0"
Command	**AT+CIPSTA?**
Response	+CIPSTA:<IP> OK
Parameters	<IP> string, IP address of ESP8266 station
Command	**AT+CIPSTA=<IP>[, <gateway>, <netmask>]**
Response	OK
Parameters	<IP> string, IP address of ESP8266 station [<gateway>] gateway [<netmask>] netmask
Notes	This configuration interacts with AT+CWDHCP related AT commands: • If static IP is enabled, DHCP will be disabled; • If DHCP is enabled, static IP will be disabled; • This will depend on the last configuration.

❖ **AT+CIPSTART – Establish TCP connection, UDP transmission or SSL connection**

AT+CIPSTART - Function 1: Establish TCP connection	
Example	AT+CIPSTART="TCP", "iot.espressif.cn", 8000 AT+CIPSTART="TCP", "192.168.101.110", 1000
Single connection (AT+CIPMUX=0)	**AT+CIPSTART=** **<type>, <remote IP>, <remote port>[, <TCP keep alive>]**
Multiple connection (AT+CIPMUX=1)	**AT+CIPSTART=<link ID>,** **<type>, <remote IP>, <remote port>[, <TCP keep alive>]**
Response	OK or ERROR If TCP is already connected, returns ALREADY CONNECT
Parameters	<link ID> ID of network connection (0~4), used for multi-connection <type> string, "TCP" or "UDP" <remote IP> string, remote IP address <remote port> string, remote port number [<TCP keep alive>] optional, detection time interval when TCP is kept alive, this function is closed by default. 0 : disable TCP keep-alive 1 ~ 7200 : detection time interval, unit: second

❖ **AT+CIPSEND – Send data**

AT+CIPSEND - Send data	
Single connection	(+CIPMUX=0) **AT+CIPSEND=<length>**
Multiple connection	(+CIPMUX=1) **AT+CIPSEND=<link ID>, <length>**
UDP Transmission	**AT+CIPSEND=[<link ID>,]<length>[, <remote IP>, <remote port>]**
Response	Wrap return ">" after set command. Begins receiving serial data, when data length is met, starts transmission of data. If connection cannot be established or gets disconnected during data transfer, returns ERROR If data is transmitted successfully, returns SEND OK
Parameters	<link ID> ID of the connection (0~4), for multi-connect <length> data length, MAX 2048 bytes [<remote IP>] optional, UDP transmission can set remote IP when sending data [<remote port>] optional, UDP transmission can set remote port when sending data
Command	**AT+CIPSEND**
Response	Wrap return ">" after execute command. Enters transparent transmission, 20ms interval between each packet, maximum 2048 bytes per packet. When single packet containing "+++" is received, it returns to normal command mode. Please wait at least 1 second before sending next AT command. This command can only be used in transparent transmission mode which requires single connection. For UDP transparent transmission, <UDP mode> has to be 0 in command "AT+CIPSTART"

VIII. Wi-Fi P2P (PEER TO PEER) COMMUNICATION

Wi-Fi peer-to-peer (P2P) allows Android 4.0 (API level 14) or later devices with the appropriate hardware to connect directly to each other via Wi-Fi without an intermediate access point (Android's Wi-Fi P2P framework complies with the Wi-Fi Alliance's Wi-Fi Direct™ certification program). Using these APIs, you can discover and connect to other devices when each device supports Wi-Fi P2P, then communicate over a speedy connection across distances much longer than a Bluetooth connection. This is useful for applications that share data among users, such as a multiplayer game or a photo sharing application.

The Wi-Fi P2P APIs consist of the following main parts:

- Methods that allow you to discover, request, and connect to peers are defined in the Wi-Fi P2P class.

- Listeners that allow you to be notified of the success or failure of Wi-Fi P2P method calls. When calling Wi-Fi P2P methods, each method can receive a specific listener passed in as a parameter.

- Intents that notify you of specific events detected by the Wi-Fi P2P framework, such as a dropped connection or a newly discovered peer.

Wi-Fi Direct negotiates the link with a system that assigns each device a limited wireless access point. The "pairing" of Wi-Fi Direct devices can be set up to require the proximity of a near field communication signal, or a button press on one or all the devices.

IX. I2C COMMUNICATION WITH ACCELEROMETER

The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 g. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I2C digital interface. The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 mg/LSB) enables measurement of inclination changes less than 1.0°. Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion and if the acceleration on any axis exceeds a user-set level. Tap sensing detects single and double taps. Free-fall sensing detects if the device is falling. These functions can be mapped to one of two interrupt output pins. An integrated, patent pending 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor intervention.

PIN CONFIGURATION AND FUNCTION DESCRIPTION

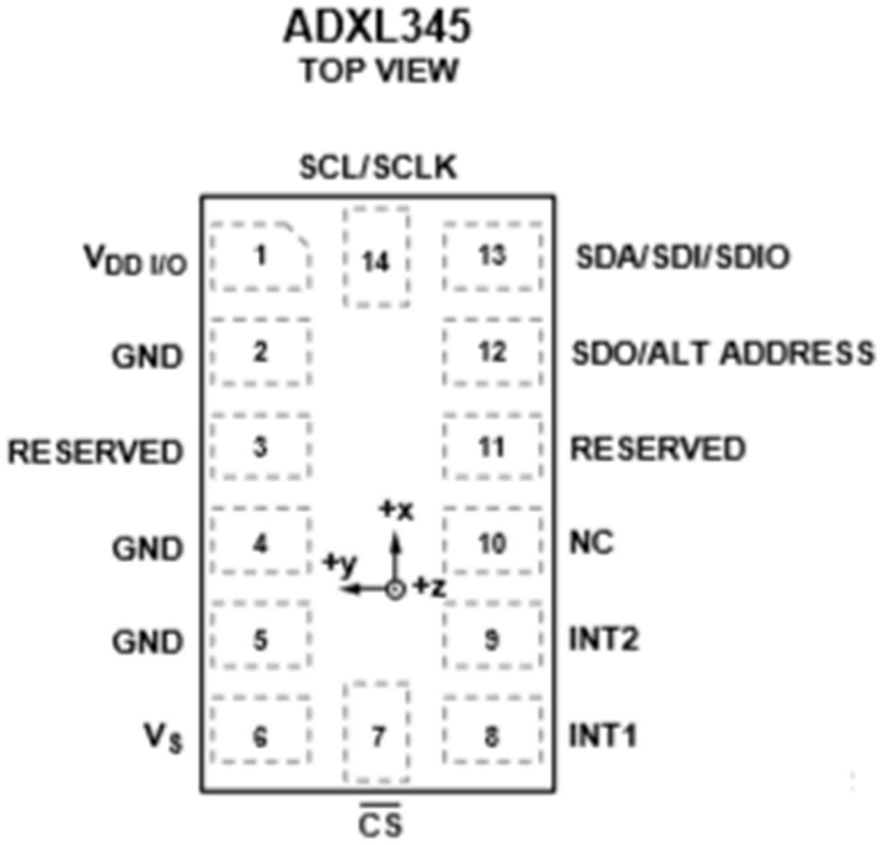


Table: -

Pin No.	Mnemonic	Description
1	$V_{DD\ I/O}$	Digital Interface Supply Voltage.
2	GND	Must be connected to ground.
3	Reserved	Reserved. This pin must be connected to V_S or left open.
4	GND	Must be connected to ground.
5	GND	Must be connected to ground.
6	V_S	Supply Voltage.
7	$\overline{CS}$	Chip Select.
8	INT1	Interrupt 1 Output.
9	INT2	Interrupt 2 Output.
10	NC	Not Internally Connected.
11	Reserved	Reserved. This pin must be connected to ground or left open.
12	SDO/ALT ADDRESS	Serial Data Output/Alternate I^2C Address Select.
13	SDA/SDI/SDIO	Serial Data (I^2C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
14	SCL/SCLK	Serial Communications Clock.

Table: - REGISTER MAP

Address					
Hex	**Dec**	**Name**	**Type**	**Reset Value**	**Description**
0x00	0	DEVID	R	11100101	Device ID.
0x01 to 0x01C	1 to 28	Reserved			Reserved. Do not access.
0x1D	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold.
0x1E	30	OFSX	R/$\overline{W}$	00000000	X-axis offset.
0x1F	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset.
0x20	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset.
0x21	33	DUR	R/$\overline{W}$	00000000	Tap duration.
0x22	34	Latent	R/$\overline{W}$	00000000	Tap latency.
0x23	35	Window	R/$\overline{W}$	00000000	Tap window.
0x24	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold.
0x25	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold.
0x26	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time.
0x27	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold.
0x29	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time.
0x2A	42	TAP_AXES	R/$\overline{W}$	00000000	Axis control for tap/double tap.
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of tap/double tap.
0x2C	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control.
0x2D	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control.
0x2E	46	INT_ENABLE	R/$\overline{W}$	00000000	Interrupt enable control.
0x2F	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control.
0x30	48	INT_SOURCE	R	00000010	Source of interrupts.
0x31	49	DATA_FORMAT	R/$\overline{W}$	00000000	Data format control.
0x32	50	DATAX0	R	00000000	X-Axis Data 0.
0x33	51	DATAX1	R	00000000	X-Axis Data 1.
0x34	52	DATAY0	R	00000000	Y-Axis Data 0.
0x35	53	DATAY1	R	00000000	Y-Axis Data 1.
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0.
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1.
0x38	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control.
0x39	57	FIFO_STATUS	R	00000000	FIFO status.

Table: - Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	LINK	AUTO_SLEEP	MEASURE	SLEEP	WAKEUP	

Link Bit

Link Bit A setting of 1 in the link bit with both the activity and inactivity functions enabled delays the start of the activity function until inactivity is detected. After activity is detected, inactivity detection begins, preventing the detection of activity. This bit serially links the activity and inactivity functions. When this bit is set to 0, the inactivity and activity functions are concurrent. Additional information can be found in the Link Mode section.

When clearing the link bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the link bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

AUTO_SLEEP Bit

AUTO_SLEEP Bit If the link bit is set, a setting of 1 in the AUTO_SLEEP bit sets the ADXL345 to switch to sleep mode when inactivity is detected (that is, when acceleration has been below the THRESH_INACT value for at least the time indicated by TIME_INACT). A setting of 0 disables automatic switching to sleep mode.

When clearing the AUTO_SLEEP bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the AUTO_SLEEP bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

Measure Bit

A setting of 0 in the measure bit places the part into standby mode, and a setting of 1 places the part into measurement mode. The ADXL345 powers up in standby mode with minimum power consumption. Bits set to 1 in this register indicate that their respective functions have triggered an event, whereas a value of 0 indicates that the corresponding event has not occurred. The DATA_READY, watermark, and overrun bits are always set if the corresponding events occur, regardless of the INT_ENABLE register settings, and are cleared by reading data from the DATAX, DATAY, and DATAZ registers. The DATA_READY and watermark bits may require multiple reads, as indicated in the FIFO mode descriptions in the FIFO section. Other bits, and the corresponding interrupts, are cleared by reading the INT_SOURCE register.

Program:-

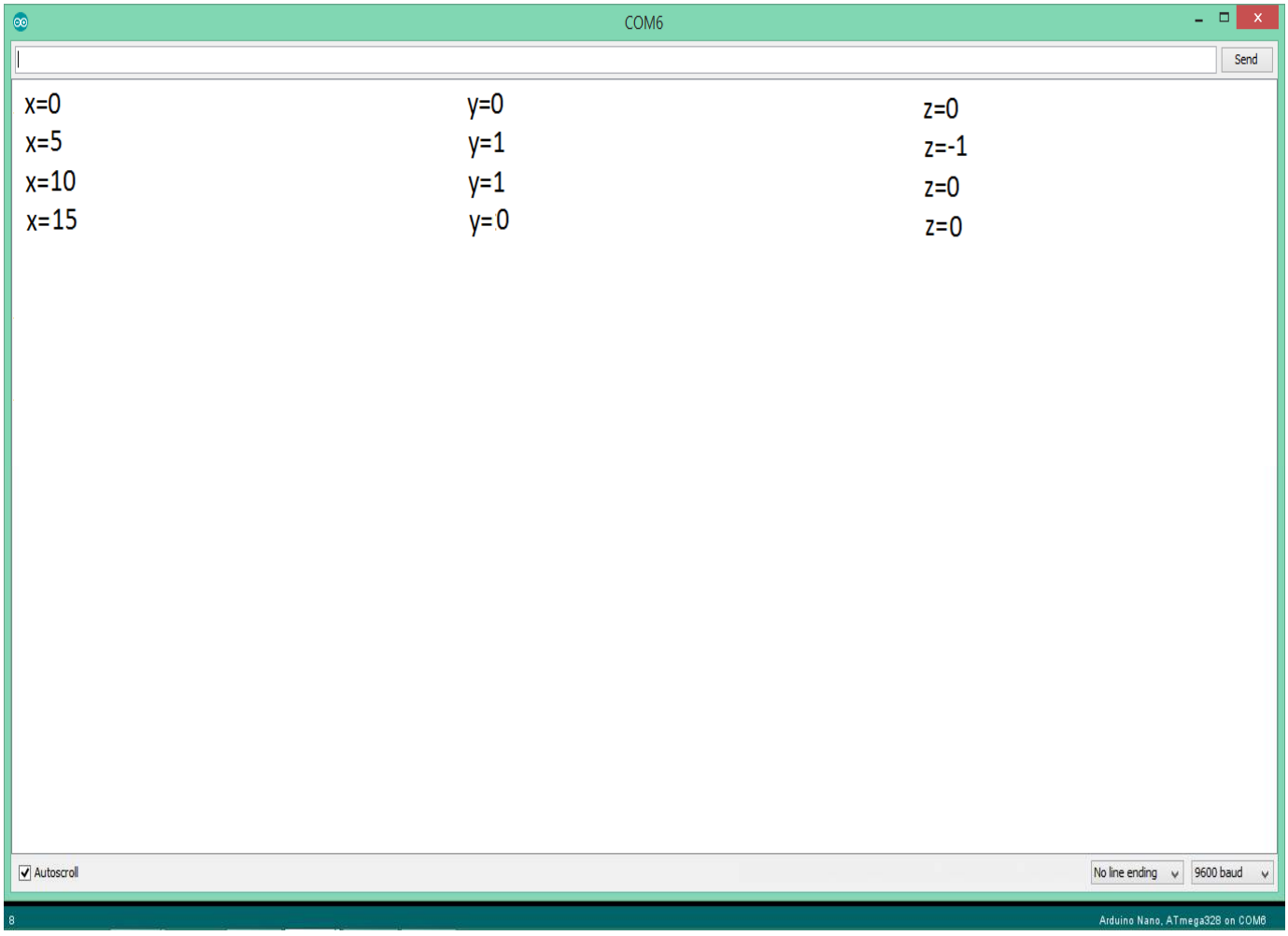
```
#include<Wire.h>
#define accel_module (0x53)
byte values[6] ;
char output[512];
void setup() {
```

```
  Wire.begin();
  Serial.begin(9600);
  Wire.beginTransmission(accel_module);
  Wire.write(0x2D);
  Wire.write(0);
  Wire.endTransmission();
  Wire.beginTransmission(accel_module);
  Wire.write(0x2D);
  Wire.write(16);
  Wire.endTransmission();
  Wire.beginTransmission(accel_module);
  Wire.write(0x2D); Wire.write(8);
  Wire.endTransmission();
}

void loop() {
  int xyzregister = 0x32;
  int x, y, z;
  Wire.beginTransmission(accel_module);
  Wire.write(xyzregister);
  Wire.endTransmission();
  Wire.beginTransmission(accel_module);
  Wire.requestFrom(accel_module, 6);
  int i = 0;
  while (Wire.available()) {
    values[i] = Wire.read();
    i++;
  }
  Wire.endTransmission();
  x = (((int)values[1]) << 8) | values[0];
```

```
  y = (((int)values[3]) << 8) | values[2];
  z = (((int)values[5]) << 8) | values[4];
  sprintf(output, "\tX= %d\t\t\t\t Y= %d\t\t\t\t Z= %d\t\t", x, y, z);
  Serial.print(output);
  Serial.write(12);
  delay(500);
}
```

Output:-



FUTURE SCOPE

i. Driver's safety warning system

ii. Deceleration indicating system

iii. Vehicle longitudinal control and collision avoidance system for an automated highway system

iv. Method, apparatus and system for transmitting and receiving data in a moving linear chain

v. Deceleration magnitude detecting and signalling device

vi. Method and apparatus for automatic vehicle event detection, characterization and reporting

vii. Process and device for indicating braking power or delay in cars

viii. Motor vehicle early warning system

ix. Panic stop, deceleration warning system

x. Vehicle collision warning system

xi. Automatic following travel system

xii. Obstruction detection method for vehicle

xiii. Inter vehicle communication system

xiv. Systems and methods for insurance based on monitored characteristics of an autonomous drive mode selection system

XI. CONCLUSION

The system which is the design and construction of an anti-collision system for vehicles was designed considering some factors such as economy, availability of components and research materials, efficiency, compatibility, portability and also durability. The performance of the system after test met design specifications. The general operation of the system and performance is dependent on the presence of two moving cars as they get closer to each other. However, it should be stated here that the system was aimed at fabricating prototype, a replica of the actual thing. It is economically viable to undertake certain system this way since testing would not cost so much. Any desire to implement this design into a vehicle would require a laser detector. The problem of power supply would not arise due to the amount of battery power from the car battery. Also the operation of the system is dependent on how well the soldering is done, and the positioning of the components on the Vero board. The Wi-Fi P2P were make away from the power supply stage to prevent heat radiation which, might occur and affect the performance of the entire system. The construction was done in such a way that it makes maintenance and repairs an easy task and affordable for the user should there be any system breakdown. All components were soldered on one Vero-board which makes troubleshooting easier. In general, the system was designed, and the real time implementation done with a photo-type of the model. It leads to decreasing of accident rate.

XII. REFRENCES

i. Zungeru, A. M. et al., (2012). Design and Implementation of a Low Cost Digital Bus Passenger Counter. Innovative Systems Design and Engineering,

ii. Zungeru, A. M. et al., (2012). Design and Implementation of a Short Message Service Based Remote Controller. Computer Engineering and Intelligent systems,

iii. Theraja, B.L., & Theraja, A.K. (1999). A textbook of electrical technology, S. Chand and company, New delhi, India.

iv. Electronics, (2012). How stuffs work: How LIDAR Work. Online available at: http://electronics.howstuffworks.com/lidar.htm