**IOT based Garbage Management System**

**ABSTRACT:**

Nowadays we can see that most of the garbage’s across the roadside are overloaded because the wastes are not collected periodically. It creates unhygienic condition for the people and creates bad odor around the surroundings. This leads in spreading some deadly diseases & human illness. To overcome this problem we are going to implement a project called IOT based Garbage Management System. In this system we are going to use NODE MCU,Ultrasonic sensor , GPRS and GPS. The vision of our project is to make the surroundings clean in smart way.GSM and GPS based Garbage and waste collection bins overflow indicator Short description: ULTRASONIC sensor will be placed UPPER the garbage bin or dustbin. When the SENSOR reaches to the threshold value, a sms will be sent to the respective Municipal / Government authority person. Then that person can send the collection vehicle to collect the full garbage bins or dustbins.

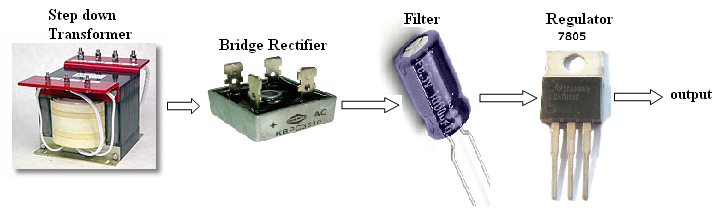
**DESCRIPTION:**

In our city many times we see that the garbage bins or dustbins placed at public places are overflowing. It creates unhygienic conditions for people. Also it creates ugliness to that place. At the same time bad smell is also spread.

To avoid all such situations we are going to implement a project called Garbage collection bin overflow indicator using GSM technology. In this project we are going to place a ULTRASONIC sensor UPPER the dustbin. When the SENSOR reaches to the threshold value, a sms will be sent to the respective Municipal / Government authority person. Then that person can send the collection vehicle to collect the full garbage bins or dustbins.

We have observed that the municipal officer or the government authorized person will monitor the status of dustbin. Or generally we see that they have a regular schedule of picking up these garbage bins or dustbins. This schedule varies as per the population of that place. It can be once in a day or twice in a day or in some cases once in two days. However we see that in case there is some festival or some function, lots of garbage material is generated by people in that particular area. In such cases the garbage dustbin gets immediately full and then it overflows which creates many problems. So in situations, with help of our project the government authority person can get SMS immediately. So they will get SMS before their periodic interval visit of picking up the dustbin. Then they can go and pick up the dustbins.

**Power Supply:**

****

**Block diagram: RECEIVER SECTION:**

**NIDEMCU**

**ESP8266**

**Power supply**

**LCD DISPLAY (2X16)**

**GPRS**

**module**

**MAX2332**

**HC SR04**

**ULTRASONIC**

**SENSOR**

**GPS**

**module**

**The major building blocks of this project are:**

1. Regulated Power Supply.

2. Microcontroller. NODEMCU

3. LCD with driver.

4. Microcontroller.

5. Crystal Oscillator.

6. LED indicators.

7. GPS MODULE.

8. GSM MODULE

9.HCSR04 ULTRASONIC SENSOR

**Software’s used:**

1. ARDUINO compiler for Embedded C programming.

2. Express SCH for Circuit design.

4. Proteus for hardware simulation.

SCHEMATIC DIAGRAM:



**INTRODUCTION TO EMBEDDED SYSTEMS:**

Embedded systems are electronic devices that incorporate microprocessors with in Their implementations. The main purposes of the microprocessors are to simplify the system design and provide flexibility. Having a microprocessor in the device means that removing the bugs, making modifications, or adding new features are only matters of rewriting the software that controls the device. Or in other words embedded computer systems are electronic systems that include a microcomputer to perform a specific dedicated application. The computer is hidden inside these products. Embedded systems are ubiquitous. Every week millions of tiny computer chips come pouring out of factories finding their way into our everyday products.

**Embedded systems** are self-contained programs that are embedded within a piece of hardware. Whereas a regular computer has many different applications and software that can be applied to various tasks, embedded systems are usually set to a specific task that cannot be altered without physically manipulating the circuitry. Another way to **think of an embedded** **system** is as a computer system that is created with optimal efficiency, thereby allowing it to complete specific functions as quickly as possible.

**Embedded systems designers** usually have a significant grasp of hardware technologies. They used specific programming languages and software to **develop embedded systems** and manipulate the equipment. When searching online, companies offer embedded systems development kits and other **embedded systems tools** for use by engineers and businesses.

**Embedded systems technologies** are usually fairly expensive due to the necessary development time and built in efficiencies, but they are also highly valued in specific industries. Smaller businesses may wish to hire a consultant to determine what sort of **embedded systems** will add value to your organization.

## Characteristics

Two major areas of differences are cost and power consumption. Since many embedded systems are produced in the tens of thousands to millions of units range, reducing cost is a major concern. Embedded systems often use a (relatively) slow processor and small memory size to minimize costs.

The slowness is not just clock speed. The whole architecture of the computer is often intentionally simplified to lower costs. For example, embedded systems often use peripherals controlled by synchronous serial interfaces, which are ten to hundreds of times slower than comparable peripherals used in PCs.

Programs on an embedded system often must run with real-time constraints with limited hardware resources: often there is no disk drive, operating system, keyboard or screen.

A flash drive may replace rotating media, and a small keypad and LCD screen may be used instead of a PC's keyboard and screen.

Firmware is the name for software that is embedded in hardware devices, e.g. in one or more ROM/Flash memory IC chips.

Embedded systems are routinely expected to maintain 100% reliability while running continuously for long periods, sometimes measured in years. Firmware is usually developed and tested too much stricter requirements than is general-purpose software, which can usually be easily restarted if a problem occurs.

### Platform

There are many different CPU architectures used in embedded designs. This in contrast to the desktop computer market, which as of this writing (2003) is limited to just a few competing architectures, mainly the Intel/AMD x86, and the Apple/Motorola/IBM PowerPC, used in the Apple Macintosh.

One common configuration for embedded systems is the *system on a chip*, an application-specific integrated circuit, for which the CPU was purchased as intellectual property to add to the IC's design.

### Tools

Like a typical computer programmer, embedded system designers use compilers, assemblers and debuggers to develop an embedded system.

Those software tools can come from several sources:

Software companies that specialize in the embedded market Ported from the GNU software development tools.

Sometimes, development tools for a personal computer can be used if the embedded processor is a close relative to a common PC processor.

Embedded system designers also use a few software tools rarely used by typical computer programmers.

Some designers keep a utility program to turn data files into code, so that they can include any kind of data in a program.

Most designers also have utility programs to add a checksum or CRC to a program, so it can check its program data before executing it.

### Operating system

They often have no operating system, or a specialized embedded operating system (often a real-time operating system), or the programmer is assigned to port one of these to the new system.

### Debugging

Debugging is usually performed with an in-circuit emulator, or some type of debugger that can interrupt the micro controller’s internal microcode.

The microcode interrupt lets the debugger operate in hardware in which only the CPU works. The CPU-based debugger can be used to test and debug the electronics of the computer from the viewpoint of the CPU. This feature was pioneered on the PDP-11.

Developers should insist on debugging which shows the high-level language, with breakpoints and single stepping, because these features are widely available. Also, developers should write and use simple logging facilities to debug sequences of real-time events.

PC or mainframe programmers first encountering this sort of programming often become confused about design priorities and acceptable methods. Mentoring, code-reviews and ego less programming are recommended.

### Design of embedded systems

The electronics usually uses either a microprocessor or a micro controller. Some large or old systems use general-purpose mainframes computers or minicomputers.

### Start-up

All embedded systems have start-up code. Usually it disables interrupts, sets up the electronics, tests the computer (RAM, CPU and software), and then starts the application code. Many embedded systems recover from short-term power failures by restarting (without recent self-tests). Restart times under a tenth of a second are common.

Many designers have found one of more hardware plus software-controlled LEDs useful to indicate errors during development (and in some instances, after product release, to produce troubleshooting diagnostics). A common scheme is to have the electronics turn off the LED(s) at reset, whereupon the software turns it on at the first opportunity, to prove that the hardware and start-up software have performed their job so far. After that, the software blinks the LED(s) or sets up light patterns during normal operation, to indicate program execution progress and/or errors. This serves to reassure most technicians/engineers and some users.

### The control loop

In this design, the software simply has a loop. The loop calls subroutines. Each subroutine manages a part of the hardware or software. Interrupts generally set flags, or update counters that are read by the rest of the software.

A simple API disables and enables interrupts. Done right, it handles nested calls in nested subroutines, and restores the preceding interrupt state in the outermost enable. This is one of the simplest methods of creating an exokernel.

Typically, there's some sort of subroutine in the loop to manage a list of software timers, using a periodic real time interrupt. When a timer expires, an associated subroutine is run, or flag is set.

Any expected hardware event should be backed-up with a software timer. Hardware events fail about once in a trillion times. That's about once a year with modern hardware. With a million mass-produced devices, leaving out a software timer is a business disaster.

State machines may be implemented with a function-pointer per state-machine (in C++, C or assembly, anyway). A change of state stores a different function into the pointer. The function pointer is executed every time the loop runs.

Many designers recommend reading each IO device once per loop, and storing the result so the logic acts on consistent values.

Many designers prefer to design their state machines to check only one or two things per state. Usually this is a hardware event, and a software timer.

Designers recommend that hierarchical state machines should run the lower-level state machines before the higher, so the higher run with accurate information.

Complex functions like internal combustion controls are often handled with multi-dimensional tables. Instead of complex calculations, the code looks up the values. The software can interpolate between entries, to keep the tables small and cheap.

One major weakness of this system is that it does not guarantee a time to respond to any particular hardware event. Careful coding can easily assure that nothing disables interrupts for long. Thus interrupt code can run at very precise timings. Another major weakness of this system is that it can become complex to add new features. Algorithms that take a long time to run must be carefully broken down so only a little piece gets done each time through the main loop.

This system's strength is its simplicity, and on small pieces of software the loop is usually so fast that nobody cares that it is not predictable.

Another advantage is that this system guarantees that the software will run. There is no mysterious operating system to blame for bad behavior.

### User interfaces

User interfaces for embedded systems vary wildly, and thus deserve some special comment. Designers recommend testing the user interface for usability at the earliest possible instant. A quick, dirty test is to ask an executive secretary to use cardboard models drawn with magic markers, and manipulated by an engineer. The videotaped result is likely to be both humorous and very educational. In the tapes, every time the engineer talk, the interface has failed: It would cause a service call.

Exactly one person should approve the user interface. Ideally, this should be a customer, the major distributor or someone directly responsible for selling the system. The decision maker should be able to decide. The problem is that a committee will never make up its mind, and neither will some people. Not doing this causes avoidable, expensive delays. A usability test is more important than any number of opinions.

Interface designers at PARC, Apple Computer, Boeing and HP minimize the number of types of user actions. For example, use two buttons (the absolute minimum) to control a menu system (just to be clear, one button should be "next menu entry" the other button should be "select this menu entry"). A touch-screen or screen-edge buttons also minimize the types of user actions.

Another basic trick is to minimize and simplify the type of output. Designs should consider using a status light for each interface plug, or failure condition, to tell what failed. A cheap variation is to have two light bars with a printed matrix of errors that they select- the user can glue on the labels for the language that she speaks.

For example, Boeing's standard test interface is a button and some lights. When you press the button, all the lights turn on. When you release the button, the lights with failures stay on. The labels are in Basic English.

For another example, look at a small computer printer. You might have one next to your computer. Notice that the lights are labeled with stick-on labels that can be printed in any language. Really look at it.

Designers use colors. Red means the users can get hurt- think of blood. Yellow means something might be wrong. Green means everything's OK.

Another essential trick is to make any modes absolutely clear on the user's display.

If an interface has modes, they must be reversible in an obvious way.

Most designers prefer the display to respond to the user. The display should change immediately after a user action. If the machine is going to do anything, it should start within 7 seconds, or give progress reports.

If a design needs a screen, many designers use plain text. It can be sold as a temporary expedient. Why is it better than pictures? Users have been reading signs for years. A GUI is pretty and can do anything, but typically adds a year from artist, approval and translator delays and one or two programmers to a project's cost, without adding any value. Often, a clever GUI actually confuses users.

If a design needs to point to parts of the machine (as in copiers), these are labeled with numbers on the actual machine, that are visible with the doors closed.

A network interface is just a remote screen. It needs the same caution as any other user interface.

One of the most successful general-purpose screen-based interfaces is the two menu buttons and a line of text in the user's native language. It's used in pagers, medium-priced printers, network switches, and other medium-priced situations that require complex behavior from users.

When there's text, there are languages. The default language should be the one most widely understood. Right now this is English. French and Spanish follow.

Most designers recommend that one use the native character sets, no matter how painful. People with peculiar character sets feel coddled and loved when their language shows up on machinery they use.

Text should be translated by professional translators, even if native speakers are on staff. Marketing staff have to be able to tell foreign distributors that the translations are professional.

A foreign organization should give the highest-volume distributor the duty to review and correct any translations in his native language. This stops critiques by other native speakers, who tend to believe that no foreign organization will ever know their language as well as they.

1. **ARDUINO**
   1. **What is Arduino?**

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board.

Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can communicate with software running on your computer (e.g. Flash, Processing, MaxMSP.) The boards can be assembled by hand or purchased preassembled; the open-source IDE can be downloaded for free.

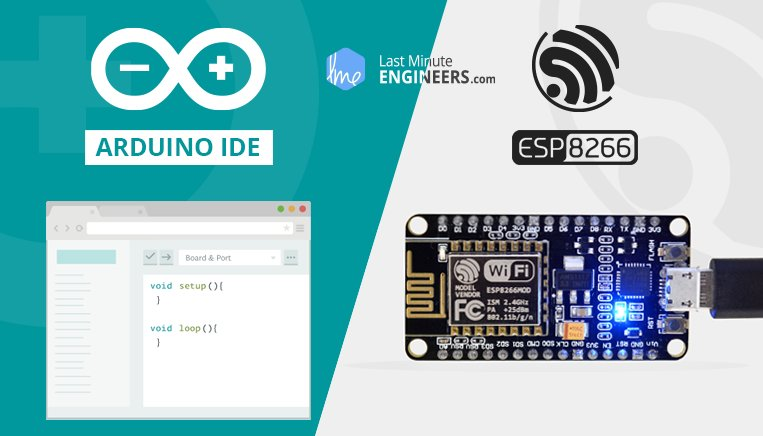
The Arduino programming language is an implementation of Wiring, a similar physical computing platform, which is based on the Processing multimedia programming environment.

* 1. **Why Arduino?**

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handy board, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

* **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50
* **Cross-platform** - The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
* **Simple, clear programming environment** - The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino
* **Open source and extensible software**- The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
* **Open source and extensible hardware** - The Arduino is based on Atmel's ATMEGA8 and ATMEGA168/ATMEGA2560 microcontrollers. The plans for the modules are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money

# Insight Into ESP8266 NodeMCU Features & Using It With Arduino IDE



The Internet of Things (IoT) has been a trending field in the world of technology. It has changed the way we work. Physical objects and the digital world are connected now more than ever. Keeping this in mind, [Espressif Systems](https://www.espressif.com/) (A Shanghai-based Semiconductor Company) has released an adorable, bite-sized WiFi enabled microcontroller – **ESP8266**, at an unbelievable price! For less than $3, it can monitor and control things from anywhere in the world – **perfect for just about any IoT project**.

## ESP-12E Module

The development board equips the ESP-12E module containing ESP8266 chip having **Tensilica Xtensa® 32-bit LX106 RISC microprocessor** which operates at **80 to 160 MHz** adjustable clock frequency and supports **RTOS**.

**ESP-12E Chip**

* Tensilica Xtensa® 32-bit LX106
* 80 to 160 MHz Clock Freq.
* 128kB internal RAM
* 4MB external flash
* 802.11b/g/n Wi-Fi transceiver

There’s also **128 KB RAM and 4MB of Flash memory** (for program and data storage) just enough to cope with the large strings that make up web pages, JSON/XML data, and everything we throw at IoT devices nowadays.

The ESP8266 Integrates **802.11b/g/n HT40 Wi-Fi transceiver**, so it can not only connect to a WiFi network and interact with the Internet, but it can also set up a network of its own, allowing other devices to connect directly to it. This makes the ESP8266 NodeMCU even more versatile.

## Power Requirement

As the operating voltage range of ESP8266 is **3V to 3.6V**, the board comes with a LDO voltage regulator to keep the voltage steady at 3.3V. It can reliably supply up to 600mA, which should be more than enough when ESP8266 pulls as much as **80mA during RF transmissions**. The output of the regulator is also broken out to one of the sides of the board and labeled as 3V3. This pin can be used to supply power to external components.

**Power Requirement**

* Operating Voltage: 2.5V to 3.6V
* On-board 3.3V 600mA regulator
* 80mA Operating Current
* 20 µA during Sleep Mode

**Power to the ESP8266 NodeMCU**is supplied via the **on-board MicroB USB connector**. Alternatively, if you have a regulated 5V voltage source, the **VIN pin** can be used to directly supply the ESP8266 and its peripherals.

Warning:

The ESP8266 requires a 3.3V power supply and 3.3V logic levels for communication. The GPIO pins are not 5V-tolerant! If you want to interface the board with 5V (or higher) components, you’ll need to do some level shifting.

## Peripherals and I/O

The ESP8266 NodeMCU has total **17 GPIO pins** broken out to the pin headers on both sides of the development board. These pins can be assigned to all sorts of peripheral duties, including:

* **ADC channel** – A 10-bit ADC channel.
* **UART interface** – UART interface is used to load code serially.
* **PWM outputs** – PWM pins for dimming LEDs or controlling motors.
* **SPI, I2C & I2S interface** – SPI and I2C interface to hook up all sorts of sensors and peripherals.
* **I2S interface** – I2S interface if you want to add sound to your project.

**Multiplexed I/Os**

* 1 ADC channels
* 2 UART interfaces
* 4 PWM outputs
* SPI, I2C & I2S interface

Thanks to the ESP8266’s **pin multiplexing feature** (Multiple peripherals multiplexed on a single GPIO pin). Meaning a single GPIO pin can act as PWM/UART/SPI.

## On-board Switches & LED Indicator

The ESP8266 NodeMCU features two buttons. One marked as **RST** located on the top left corner is the Reset button, used of course to reset the ESP8266 chip. The other **FLASH** button on the bottom left corner is the download button used while upgrading firmware.

**Switches & Indicators**

* RST – Reset the ESP8266 chip
* FLASH – Download new programs
* Blue LED – User Programmable

The board also has a**LED indicator** which is user programmable and is connected to the D0 pin of the board.

## Serial Communication

The board includes CP2102 USB-to-UART Bridge Controller from [Silicon Labs](http://www.silabs.com/), which converts USB signal to serial and allows your computer to program and communicate with the ESP8266 chip.

**Serial Communication**

* CP2102 USB-to-UART converter
* 4.5 Mbps communication speed
* Flow Control support

If you have an older version of CP2102 driver installed on your PC, we recommend upgrading now.

[CP2102 Driver](https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers)

ESP8266 NodeMCU Pinout

The ESP8266 NodeMCU has total 30 pins that interface it to the outside world. The connections are as follows:

For the sake of simplicity, we will make groups of pins with similar functionalities.

Power Pins There are four power pins viz. one VIN pin & three 3.3V pins. The VIN pin can be used to directly supply the ESP8266 and its peripherals, if you have a regulated 5V voltage source. The 3.3V pins are the output of an on-board voltage regulator. These pins can be used to supply power to external components.

GND is a ground pin of ESP8266 NodeMCU development board.

I2C Pins are used to hook up all sorts of I2C sensors and peripherals in your project. Both I2C Master and I2C Slave are supported. I2C interface functionality can be realized programmatically, and the clock frequency is 100 kHz at a maximum. It should be noted that I2C clock frequency should be higher than the slowest clock frequency of the slave device.

GPIO Pins ESP8266 NodeMCU has 17 GPIO pins which can be assigned to various functions such as I2C, I2S, UART, PWM, IR Remote Control, LED Light and Button programmatically. Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, it can also be set to edge-trigger or level-trigger to generate CPU interrupts.

ADC Channel The NodeMCU is embedded with a 10-bit precision SAR ADC. The two functions can be implemented using ADC viz. Testing power supply voltage of VDD3P3 pin and testing input voltage of TOUT pin. However, they cannot be implemented at the same time.

UART Pins ESP8266 NodeMCU has 2 UART interfaces, i.e. UART0 and UART1, which provide asynchronous communication (RS232 and RS485), and can communicate at up to 4.5 Mbps. UART0 (TXD0, RXD0, RST0 & CTS0 pins) can be used for communication. It supports fluid control. However, UART1 (TXD1 pin) features only data transmit signal so, it is usually used for printing log.

SPI Pins ESP8266 features two SPIs (SPI and HSPI) in slave and master modes. These SPIs also support the following general-purpose SPI features:

* 4 timing modes of the SPI format transfer
* Up to 80 MHz and the divided clocks of 80 MHz
* Up to 64-Byte FIFO

SDIO Pins ESP8266 features Secure Digital Input/Output Interface (SDIO) which is used to directly interface SD cards. 4-bit 25 MHz SDIO v1.1 and 4-bit 50 MHz SDIO v2.0 are supported.

PWM Pins The board has 4 channels of Pulse Width Modulation (PWM). The PWM output can be implemented programmatically and used for driving digital motors and LEDs. PWM frequency range is adjustable from 1000 μs to 10000 μs, i.e., between 100 Hz and 1 kHz.

Control Pins are used to control ESP8266. These pins include Chip Enable pin (EN), Reset pin (RST) and WAKE pin.

* EN pin – The ESP8266 chip is enabled when EN pin is pulled HIGH. When pulled LOW the chip works at minimum power.
* RST pin – RST pin is used to reset the ESP8266 chip.
* WAKE pin – Wake pin is used to wake the chip from deep-sleep.

ESP8266 Development Platforms

Now, let’s move on to the interesting stuff!

There are a variety of development platforms that can be equipped to program the ESP8266. You can go with [Espruino](https://www.espruino.com/) – JavaScript SDK and firmware closely emulating Node.js, or use [Mongoose OS](https://mongoose-os.com/) – An operating system for IoT devices (recommended platform by Espressif Systems and Google Cloud IoT) or use a software development kit (SDK) provided by Espressif or one of the platforms listed on [WiKiPedia](https://en.wikipedia.org/wiki/ESP8266" \l "SDKs).

Fortunately, the amazing ESP8266 community took the IDE selection a step further by creating an Arduino add-on. If you’re just getting started programming the ESP8266, this is the environment we recommend beginning with, and the one we’ll document in this tutorial.

This ESP8266 add-on for Arduino is based on the amazing work by [Ivan Grokhotkov](https://github.com/igrr) and the rest of the ESP8266 community. Check out the [ESP8266 Arduino GitHub repository](https://github.com/esp8266/Arduino) for more information.

Installing the ESP8266 Core on Windows OS

Let’s proceed with installing ESP8266 Arduino core.

The first thing is having latest Arduino IDE (Arduino 1.6.4 or higher) installed on your PC. If don’t have it, we recommend upgrading now.

[Latest Arduino IDE](https://www.arduino.cc/en/Main/Software)

To begin, we’ll need to update the board manager with a custom URL. Open up Arduino IDE and go to File > Preferences. Then, copy below URL into the Additional Board Manager URLs text box situated on the bottom of the window:

http://arduino.esp8266.com/stable/package\_esp8266com\_index.json

Hit OK. Then navigate to the Board Manager by going to Tools > Boards > Boards Manager. There should be a couple new entries in addition to the standard Arduino boards. Filter your search by typing esp8266. Click on that entry and select Install.

The board definitions and tools for the ESP8266 include a whole new set of gcc, g++, and other reasonably large, compiled binaries, so it may take a few minutes to download and install (the archived file is ~110MB). Once the installation has completed, a small *INSTALLED* text will appear next to the entry. You can now close the Board Manager.

Arduino Example: Blink

To make sure ESP8266 Arduino core and the NodeMCU are properly set up, we’ll upload the simplest sketch of all – The Blink!

We will use the on-board LED for this test. As mentioned earlier in this tutorial, D0 pin of the board is connected to on-board Blue LED & is user programmable. Perfect!

Before we get to uploading sketch & playing with LED, we need to make sure that the board is selected properly in Arduino IDE. Open Arduino IDE and select NodeMCU 0.9 (ESP-12 Module) option under your Arduino IDE > Tools > Board menu.

Now, plug your ESP8266 NodeMCU into your computer via micro-B USB cable. Once the board is plugged in, it should be assigned a unique COM port. On Windows machines, this will be something like COM#, and on Mac/Linux computers it will come in the form of /dev/tty.usbserial-XXXXXX. Select this serial port under the Arduino IDE > Tools > Port menu. Also select the Upload Speed : 115200

Warning:

More attention needs to be given to selecting board, choosing COM port and selecting Upload speed. You may get espcomm\_upload\_mem error while uploading new sketches, if failed to do so.

Once you are done, try the example sketch below.

void setup()

{

pinMode(D0, OUTPUT);

}

void loop()

{

digitalWrite(D0, HIGH);

delay(500);

digitalWrite(D0, LOW);

delay(500);

}

Once the code is uploaded, LED will start blinking. You may need to tap the RST button to get your ESP8266 to begin running the sketch.

### 2.1 Architecture of GSM Network:

A GSM network is composed of several functional entities, whose functions and interfaces are specified. Figure 1 shows the layout of a generic GSM network. The GSM network can be divided into three broad parts. The Mobile Station is carried by the subscriber. The Base Station Subsystem controls the radio link with the Mobile Station. The Network Subsystem, the main part of which is the Mobile services Switching Center (MSC), performs the switching of calls between the mobile users, and between mobile and fixed network users. The MSC also handles the mobility management operations. Not shown are the Operations

A GSM network is composed of several functional entities, whose functions and interfaces are specified. Figure shows the layout of a generic GSM network. The GSM network can be divided into three broad parts. Subscriber carries the Mobile Station. The Base Station Subsystem controls the radio link with the Mobile Station. The Network Subsystem, the main part of which is the Mobile services Switching Center (MSC), performs the switching of calls between the mobile users, and between mobile and fixed network users. The MSC also handles the mobility management operations. Not shown is the Operations intendance Center, which oversees the proper operation and setup of the network. The Mobile Station and the Base Station Subsystem communicate across the Um interface, also known as the air interface or radio link. The Base Station Subsystem communicates with the Mobile services Switching Center across the A interface.

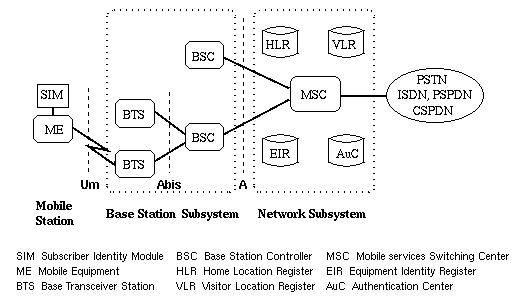


Fig 2.1 General architecture of a GSM network

**Mobile Station:**

The mobile station (MS) consists of the mobile equipment (the terminal) and a smart card called the Subscriber Identity Module (SIM). The SIM provides personal mobility, so that the user can have access to subscribed services irrespective of a specific terminal. By inserting the SIM card into another GSM terminal, the user is able to receive calls at that terminal, make calls from that terminal, and receive other subscribed services. The mobile equipment is uniquely identified by the International Mobile Equipment Identity (IMEI). The SIM card contains the International Mobile Subscriber Identity (IMSI) used to identify the subscriber to the system, a secret key for authentication, and other information. The IMEI and the IMSI are independent, thereby allowing personal mobility. The SIM card may be protected against unauthorized use by a password or personal identity number.

**Base Station Subsystem:**

The Base Station Subsystem is composed of two parts, the Base Transceiver Station (BTS) and the Base Station Controller (BSC). These communicate across the standardized Abis interface, allowing (as in the rest of the system) operation between components made by different suppliers. The Base Transceiver Station houses the radio transceivers that define a cell and handles the radio-link protocols with the Mobile Station. In a large urban area, there will potentially be a large number of BTSs deployed, thus the requirements for a BTS are ruggedness, reliability, portability, and minimum cost. The Base Station Controller manages the radio resources for one or more BTSs. It handles radio-channel setup, frequency hopping, and handovers, as described below. The BSC is the connection between the mobile station and the Mobile service Switching Center (MSC).

##### Network Subsystem

The central component of the Network Subsystem is the Mobile services Switching Center (MSC). It acts like a normal switching node of the PSTN or ISDN, and additionally provides all the functionality needed to handle a mobile subscriber, such as registration, authentication, location updating, handovers, and call routing to a roaming subscriber. These services are provided in conjunction with several functional entities, which together form the Network Subsystem. The MSC provides the connection to the fixed networks (such as the PSTN or ISDN). Signalling between functional entities in the Network Subsystem uses Signalling System Number 7 (SS7), used for trunk signalling in ISDN and widely used in current public networks.

The Home Location Register (HLR) and Visitor Location Register (VLR), together with the MSC, provide the call-routing and roaming capabilities of GSM. The HLR contains all the administrative information of each subscriber registered in the corresponding GSM network, along with the current location of the mobile. The location of the mobile is typically in the form of the signaling address of the VLR associated with the mobile as a distributed database station. The actual routing procedure will be described later. There is logically one HLR per GSM network, although it may be implemented

The Visitor Location Register (VLR) contains selected administrative information from the HLR, necessary for call control and provision of the subscribed services, for each mobile currently located in the geographical area controlled by the VLR. Although each functional entity can be implemented as an independent unit, all manufacturers of switching equipment to date implement the VLR together with the MSC, so that the geographical area controlled by the MSC corresponds to that controlled by the VLR, thus simplifying the signalling required. Note that the MSC contains no information about particular mobile stations --- this information is stored in the location registers. The other two registers are used for authentication and security purposes. The Equipment Identity Register (EIR) is a database that contains a list of all valid mobile equipment on the network, where each mobile station is identified by its International Mobile Equipment Identity (IMEI). An IMEI is marked as invalid if it has been reported stolen or is not type approved. The Authentication Center (AuC) is a protected database that stores a copy of the secret key stored in each subscriber's SIM card, which is used for authentication and encryption over the radio channel.

## 2.2 GSM Network Operators

T-Mobile and Cingular operate GSM networks in the United States on the 1,900 MHz band. GSM networks in other countries operate at 900, 1,800, or 1,900 MHz

### 2.3 GSM carrier frequencies

GSM networks operate in a number of different carrier frequency ranges (separated into [GSM frequency ranges](http://en.wikipedia.org/wiki/GSM_frequency_ranges) for 2G and [UMTS frequency bands](http://en.wikipedia.org/wiki/UMTS_frequency_bands) for 3G), with most [2G](http://en.wikipedia.org/wiki/2G) GSM networks operating in the 900 MHz or 1800 MHz bands. Where these bands were already allocated, the 850 MHz and 1900 MHz bands were used instead (for example in [Canada](http://en.wikipedia.org/wiki/Canada) and the [United States](http://en.wikipedia.org/wiki/United_States)). In rare cases the 400 and 450 MHz frequency bands are assigned in some countries because they were previously used for first-generation systems. Most [3G](http://en.wikipedia.org/wiki/3G) networks in Europe operate in the 2100 MHz frequency band. Regardless of the frequency selected by an operator, it is divided into [timeslots](http://en.wikipedia.org/wiki/Time_division_multiplexing) for individual phones to use. This allows eight full-rate or sixteen half-rate speech channels per [radio frequency](http://en.wikipedia.org/wiki/Radio_frequency). These eight radio timeslots (or eight [burst](http://en.wikipedia.org/wiki/Burst_transmission) periods) are grouped into a [TDMA](http://en.wikipedia.org/wiki/Time_division_multiple_access) frame. Half rate channels use alternate frames in the same timeslot. The channel data rate for all 8 channels is 270.833 Kbit/s, and the frame duration is 4.615 ms. The transmission power in the handset is limited to a maximum of 2 watts in GSM850/900 and 1 watt in GSM1800/1900.

**2.4 GSM ATCommands**

AT Commands are used to perform different operations is GSM module

**Short message commands**

**2.4.1Preferred Message Format +CMGF**

**Description:**

The message formats supported are *text mode* and *PDU mode*. In PDU mode, a complete SMS Message including all header information is given as a binary string (in hexadecimal format).Therefore, only the following set of characters is allowed: {‘0’,’1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’,’9’, ‘A’,‘B’,’C’,’D’,’E’,’F’}. Each pair or characters are converted to a byte (e.g.: ‘41’ is converted to theASCII character ‘A’, whose ASCII code is 0x41 or 65). In Text mode, all commands and responses are in ASCII characters. The format selected is stored in EEPROM by the +CSAS command.

Syntax : Command syntax: AT+CMGF

|  |  |
| --- | --- |
| Command | Possible responses |
| AT+CMGF?  Note :possible message format | +CMGF=1  Ok  Note :Text mode |

**2.4.2 Send message + CMGS**

To send a message in text mode **CMGS** command used

**Description:**

The <address> field is the address of the terminal to which the message is sent. To send the message, simply type, <ctrl-Z> character (ASCII 26). The text can contain all existing characters except <ctrl-Z> and <ESC> (ASCII 27). This command can be aborted using the <ESC> character when entering text. In PDU mode, only hexadecimal characters are used (‘0’…’9’,’A’…’F’).

**Syntax:**

In SMS text mode, the syntax of the +CMGS AT command is: (Optional parameters are enclosed in square brackets.)

+CMGS=address[,address\_type]<CR>sms\_message\_body<Ctrl+z>Before we discuss each of the parameters, let's see an example that gives you some idea of how an actual command line should look like:AT+CMGS="+85291234567",145<CR>This is an example for illustrating the syntax of the +CMGS AT command in SMS text mode.<Ctrl+z>

The address Parameter

The first parameter of the +CMGS AT command, *address*, specifies the destination address to send the SMS message to. Usually it is a mobile number formatted using the typical ISDN / telephony numbering plan (ITU E.164/E.163). For example, "+85291234567", "91234567", etc. Note that the value passed to the *address* parameter should be a string, i.e. it should be enclosed in double quotes.

The address\_type Parameter

The second parameter of the +CMGS AT command, *address\_type*, specifies the type of the address assigned to the *address* parameter. Two values are commonly used. They are 129 and 145:

As *address\_type* is an optional parameter, it can be omitted. If you do so, the GSM/GPRS modem or mobile phone will use the default value of the *address\_type* parameter, which is:

* 129 if the value of *address* does not start with a "+" character. For example, "85291234567".
* 145 if the value of *address* starts with a "+" character. For example, "+85291234567".

The <CR> Character

*<CR>*, which represents the carriage return character, follows the *address\_type* parameter. When the GSM/GPRS modem or mobile phone receives the carriage return character, it will send back a prompt formed by these four characters: the carriage return character, the linefeed character, the ">" character and the space character.

The sms\_message\_body Parameter

The third parameter of the +CMGS AT command, *sms\_message\_body*, specifies the body of the SMS message to be sent. Entering the *<Esc>* character will cancel the +CMGS AT command.

The <Ctrl+z> Character

When you finish entering the SMS message body, you have to enter the *<Ctrl+z>* character to mark the end of the SMS message body. The GSM/GPRS modem or mobile phone will then attempt to send the SMS message to the SMS center

**5.5.3 Read message +CMGR**

The AT command +CMGR (command name in text: Read Message) is used to read a message from a message storage area. The location of the message to be read from the message storage area is specified by an index number. The message to be retrieved by the AT command +CMGR does not necessarily have to be an SMS message. It can be of other message types such as status reports and cell broadcast messages, but we will only focus on SMS messages here.

### Format of the Information Response of the +CMGR AT Command in SMS Text Mode

If the GSM/GPRS modem or mobile phone successfully reads the SMS message from message storage, it will return an information response to the computer / PC. In SMS text mode, the format of the information response of the +CMGR AT command is different for different message types. In the sections that follow, we assume the message to be read is an SMS message but not of other message types like status reports and cell broadcast messages.

#### Incoming SMS Messages and Outgoing SMS Messages

If the SMS message retrieved is an SMS message received from the SMS center (i.e. incoming SMS message), the information response of the +CMGR AT command in SMS text mode has the following format: (Optional fields are enclosed in square brackets.)

+CMGR:message\_status,address,[address\_text],service\_center\_time\_stamp[,address\_type,TPDU\_first\_octet,protocol\_identifier,data\_coding\_scheme,service\_center\_address,service\_center\_address\_type,sms\_message\_body\_length]<CR><LF>sms\_message\_body

For incoming SMS messages:

+CMGR:"RECREAD","+85291234567",,"07/04/20,10:08:02+32",145,4,0,0,"+85290000000",145,49  
It is easy to read text messages via AT Commands

**4. GLOBAL POSITION SYSTEM**

### 4 Introduction

The Global Positioning System (GPS) is the most significant recent advance in navigation and positioning technology .In the past, the stars was used for navigation. Today’s world requires greater accuracy .The new constellation of with radius equal to the distance to the satellite. If two satellites are used, then the receiver must be on the surface of both spheres, which is the intersection of the two spheres or the perimeter of a circle. If a third satellite is used, then the location of the user is narrowed down to the two points where the three spheres intersect. Three measurements are enough for land receivers since the lower of the two points would be selected. But when in the air or space, four satellites are needed; the intersection of all four spheres will be the receiver’s location. When more than four satellites are used, greater accuracy can be achieved.

Global Positioning System satellites transmit signals to equipment on the ground, GPS receivers passively receive satellite signals; they do not transmit. GPS receivers require unobstructed views of the sky, so they are used only outdoors and they often do not perform well within forested areas or near tall buildings. GPS operations depend on a very accurate time reference.

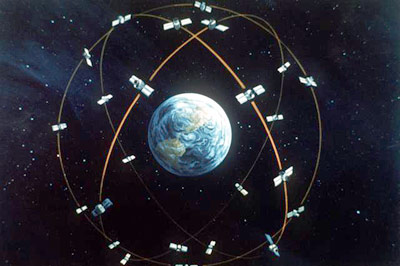
****

Figure 4.1 Satellite System

**4.1 Satellites in Space**:

The complete GPS space system includes 24 satellites, 11,000 nautical miles above the Earth, which take 12 hours each to go around the Earth once (one orbit). Satellites are equipped with very precise clocks that keep accurate time to within three nanoseconds - that’s 0.000000003, or three billionths, of a second. This precision timing is important because the receiver must determine exactly how long it takes for signals to travel from each GPS satellite. The receiver uses this information to calculate its position. The first GPS satellite was launched in 1978. The first 10 satellites were developmental satellites, called Block 1 and the first product for civilian consumers appeared in the mid 1980’s

#### 4.2. NMEA input:

Some units also support an NMEA input mode. While not too many programs support this mode it does provide a standardized way to update or add waypoint and route data. Note that there is no handshaking or commands in NMEA mode so you just send the data in the correct sentence and the unit will accept the data and add or overwrite the information in memory.

The most important NMEA sentences include the GGA which provides the current Fix data, the RMC which provides the minimum gps sentences information, and the GSA which provides the Satellite status data.

**GGA** - essential fix data which provide 3D location and accuracy data.

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,\*47

**GSA** - GPS DOP and active satellites. This sentence provides details on the nature of the fix. It includes the numbers of the satellites being used in the current solution and the DOP

$GPGSA,A,3,04,05,,09,12,,,24,,,,,2.5,1.3,2.1\*39

**RMC** - NMEA has its own version of essential Gps pvt (position, velocity, time) data. It is called RMC, The Recommended Minimum, which will look similar to:

$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W\*6A

**4.3 GPS Applications:**

An One of the most significant and unique feature of the GPS is the fact that the positioning signals available to users in any position worldwide at any time With a fully operational GPS system, there are multiple applications,rangingfromsurveying,mapping,and navigation to GIS data capture. The GPS will soon be apart of the overall utility of technology. There are countlessGPSapplications,a few importantances are covered in the following passage   
4**.3.1** **Surveying and Mapping:** The high precision of GPS carrier phase measurements, together with appropriate adjustment algorithms, provides an adequate tool for a variety of tasks for surveying and mapping. Using DGPs methods,accurate and timely mapping of almost anything canbe carriedout.The GPS is used to map cut blocks,road alignments,and environmental hazards suchas landslides,forestfires,andoilspills.Applications,such as cadastral mapping, needing a high degree of accuracy also can be carried out using high grade GPS receivers. Continuous kinematic techniques can be used for topographic surveys and accurate linear mapping.

4**.3.2. Navigation:** Navigation using GPS can save countless hours in the field. Any feature, even if it is under water, can be located up to one hundred meters simply by scaling coordinates from a map, entering waypoints, and going directly to the site. Examples include road intersections, corner posts, plot canters, accident sites, geological formations, and so on. GPS navigation in helicopters, in vehicles, or in a ship can provide an easy means of navigation with substantial savings.

**4.3.3.** **Remote Sensing and GIS:** It is also possible to integrate GPS positioning into remote-sensing methods such as photogrammetric and aerial scanning, magnetometer, and video technology. Using DGPS or kinematic techniques, depending upon the accuracy required, real time or post-processing will provide positions for the sensor which can be projected to the ground, instead of having ground control projected to an image. GPS are becoming very effective tools for GIS data capture. The GIS user community benefits from the use of GPS for location data capture in various GIS applications. The GPS can easily be linked to a laptop computer in the field, and, with appropriate software, users can also have all their data on a common base with every little distortion.

**Liquid Crystal Display (LCD):**

Liquid crystal displaya type of display used in digital watches and many portable computers.



LCD displays utilize two sheets of polarizing material with a liquid crystal solution between them. An electric current passed through the liquid causes the crystals to align so that light cannot pass through them. Each crystal, therefore, is like a shutter, either allowing light to pass through or blocking the light.

The liquid crystals can be manipulated through an applied electric voltage so that light is allowed to pass or is blocked.

By carefully controlling where and what wavelength (color) of light is allowed to pass, the LCD monitor is able to display images. A back light provides LCD monitor’s brightness.

Other advances have allowed LCD’s to greatly reduce liquid crystal cell response times.

Response time is basically the amount of time it takes for a pixel to “change colors”. In reality response time is the amount of time it takes a liquid crystal cell to go from being active to inactive

Here the LCD is used at both the Transmitter as well as the receiver side.

The input which we give to the microcontroller is displayed on the LCD of the transmitter side and the message sent is received at the receiver side which displays at the receiver end of the LCD and the corresponding operation is performed

They make complicated equipment easier to operate. LCDs come in many shapes and sizes but the most common is the 16 character x 4 line display with no backlight.

It requires only 11 connections – eight bits for data (which can be reduced to four if necessary) and three control lines (we have only used two here). It runs off a 5V DC supply and only needs about 1mA of current.

The display contrast can be varied by changing the voltage into pin 3 of the display,

**Pin description of LCD:**

****

From this description, the interface is a parallel bus, allowing simple and fast reading/writing of data to and from the LCD. This waveform will write an ASCII Byte out to the LCD's screen.

# PIN DESCRIPTIONS

**Vcc, Vss and Vee**

While Vcc and Vss provide +5V and ground respectively, Vee is used for controlling LCD contrast.

|  |  |  |  |
| --- | --- | --- | --- |
| **PIN** | **SYMBOL** | **I/O** | **DESCRIPTION** |
| 1 | Vss | -- | Ground |
| 2 | Vcc | -- | +5V power supply |
| 3 | Vee | -- | Power supply to control contrast |
| 4 | RS | I | RS=0 to select command register  RS=1 to select data register |
| 5 | R/W | I | R/W=0 for write  R/W=1 for read |
| 6 | EN | I/O | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

The ASCII code to be displayed is eight bits long and is sent to the LCD either four or eight bits at a time.

If four bit mode is used, two "nibbles" of data (Sent high four bits and then low four bits with an "E" Clock pulse with each nibble) are sent to make up a full eight bit transfer.

The "E" Clock is used to initiate the data transfer within the LCD.

Deciding how to send the data to the LCD is most critical decision to be made for an LCD interface application.

Eight-bit mode is best used when speed is required in an application and at least ten I/O pins are available.

The "R/S" bit is used to select whether data or an instruction is being transferred between the microcontroller and the LCD.

If the Bit is set, then the byte at the current LCD "Cursor" Position can be reader written.

When the Bit is reset, either an instruction is being sent to the LCD or the execution status of the last instruction is read back

**INTERFACING LCD WITH CONTROLLER:**

# P1.0

**P1.1**

**P1.2**

**P1.3**

**P1.4**

**P1.5**

**P1.6**

**P1.7**

**P3.7**

**P3.6**

**P3.5**

## DB0

**DB1**

**DB2**

**DB3**

**DB4**

**DB5**

**DB6**

## DB7

**EN**

**RS**

**RW**

**8052**

**μC**

## HD44780

**LCD**

**Interfacing a LCD with a microcontroller.**

**Advantages:**

LCD interfacing with 8051 is a real-world application. In recent years the LCD is finding widespread use replacing LEDs (seven segment LEDs or other multisegment LEDs).

This is due to following reasons:

1. The declining prices of LCDs.

2. The ability to display numbers, characters and graphics. This is in contrast to LEDs, which are limited to numbers and a few characters. An intelligent LCD display of two lines, 20 characters per line, which is interfaced to the 8051.

3. Incorporation of a refreshing controller into the LCD, thereby relieving the CPU to keep displaying the data.

4. Ease of programming for characters and graphics.

**Basic commands of LCD:**

When LCD is powered up, the display should show a series of dark squares, possibly only on part of display.

These characters are actually in their off state, so the contrast control should be adjusted anti-clockwise until the squares are just visible.

The display module resets itself to an initial state when power is applied, which curiously the display has blanked off so that even if characters are entered, they cannot be seen.

It is therefore necessary to issue a command at this point, to switch the display on.

**Prototype circuit:**

For a LCD module to be used effectively in any piece of equipment, a microprocessor or a micro controller is usually required to drive it.

However, before attempting to wire the two together, some initial experiments can be performed by connecting a series of switches to the pins of the module.

This can be a quite beneficial step, if even you are thoroughly conversant with the workings of microprocessors.

**Circuit description of LCD experiment:**

The circuit can be wired up on a “plug-in-style” prototyping board, using dual-in-line switches for the data lines (S1-S8)

A toggle switch for the RS input (S10) and a momentary action switch (or macro switch) for usage.

Most of the LCD modules conform to a standard interface specification. A 14pin access is provided having eight data lines, three control lines and three power lines.

The connections are laid out in one of the two common configurations, either two rows of seven pins, or a single row of 14 pins.

One of the, pins are numbered on the LCD’s print circuit board (PCB), but if not, it is quite easy to locate pin1.

Since this pin is connected to ground, it often has a thicker PCB track, connected to it, and it is generally connected to metalwork at same point.

**Pin description:**

**G +5V -5v**

**1 2 3**

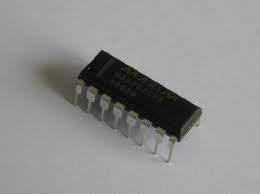
**07 08 09 10 11 12 13 14 4 5 6**

D0 D1 D2 D3 D4 D5 D6 D7 RS R/W EN

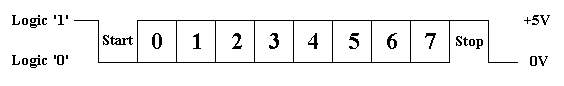
The LCD plays a major role in the entire operation as it has the ability to display the certain data that the user has entitled.

LCD display varies from input to input as there is no specific outline for it to operate.

**MAX 232**

****

**RS-232 WAVEFORM**

  
TTL/CMOS Serial Logic Waveform

The diagram above shows the expected waveform from the UART when using the common 8N1 format. 8N1 signifies 8 Data bits, No Parity and 1 Stop Bit. The RS-232 line, when idle is in the Mark State (Logic 1). A transmission starts with a start bit which is (Logic 0). Then each bit is sent down the line, one at a time. The LSB (Least Significant Bit) is sent first. A Stop Bit (Logic 1) is then appended to the signal to make up the transmission.

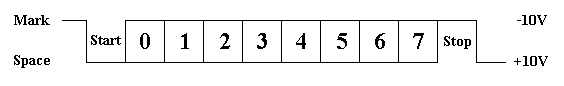
The data sent using this method, is said to be framed. That is the data is framed between a Start and Stop Bit.

**RS-232 Voltage levels**

* +3 to +25 volts to signify a "Space" (Logic 0)
* -3 to -25 volts for a "Mark" (logic 1).
* Any voltage in between these regions (i.e. between +3 and -3 Volts) is undefined.

The data byte is always transmitted least-significant-bit first.

The bits are transmitted at specific time intervals determined by the baud rate of the serial signal.  This is the signal present on the RS-232 Port of your computer, shown below.

  
RS-232 Logic Waveform

**2.3.2 RS-232 LEVEL CONVERTER**

Standard serial interfacing of microcontroller (TTL) with PC or any  [RS232C](http://www.arcelect.com/rs232.htm) Standard device , requires TTL to RS232 Level converter . A [MAX232](http://www.bsc.nodak.edu/electron/rs232.htm) is used for this purpose. It provides 2-channel RS232C port and requires external 10uF capacitors.

The driver requires a single supply of +5V.

|  |  |
| --- | --- |
| MAX232 | MAX232A |

 MAX-232 includes a Charge Pump, which generates +10V and -10V from a single 5v supply.

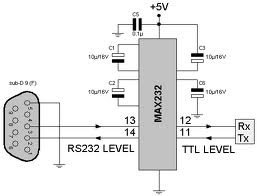
**Serial communication**

When a processor communicates with the outside world, it provides data in byte sized chunks. Computers transfer data in two ways: parallel and serial. In parallel data transfers, often more lines are used to transfer data to a device and 8 bit data path is expensive. The serial communication transfer uses only a single data line instead of the 8 bit data line of parallel communication which makes the data transfer not only cheaper but also makes it possible for two computers located in two different cities to communicate over telephone.

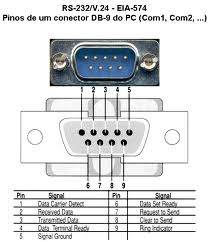
Serial data communication uses two methods, asynchronous and synchronous. The synchronous method transfers data at a time while the asynchronous transfers a single byte at a time. There are some special IC chips made by many manufacturers for data communications. These chips are commonly referred to as UART (universal asynchronous receiver-transmitter) and USART (universal synchronous asynchronous receiver transmitter). The AT89C51 chip has a built in UART.

In asynchronous method, each character is placed between start and stop bits. This is called framing. In data framing of asynchronous communications, the data, such as ASCII characters, are packed in between a start and stop bit. We have a total of 10 bits for a character: 8 bits for the ASCII code and 1 bit each for the start and stop bits. The rate of serial data transfer communication is stated in bps or it can be called as baud rate.

To allow the compatibility among data communication equipment made by various manufacturers, and interfacing standard called RS232 was set by the Electronics industries Association in 1960. Today RS232 is the most widely used I/O interfacing standard. This standard is used in PCs and numerous types of equipment. However, since the standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible. In RS232, a 1 bit is represented by -3 to -25V, while a 0 bit is represented +3 to +25 V, making -3 to +3 undefined. For this reason, to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to connect the TTL logic levels to RS232 voltage levels and vice versa. MAX232 ICs are commonly referred to as line drivers.



The RS232 cables are generally referred to as DB-9 connector. In labeling, DB-9P refers to the plug connector (male) and DB-9S is for the socket connector (female). The simplest connection between a PC and microcontroller requires a minimum of three pin, TXD, RXD, and ground. Many of the pins of the RS232 connector are used for handshaking signals. They are bypassed since they are not supported by the UART chip.

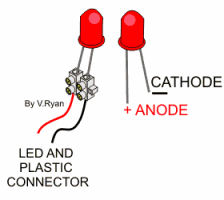


IBM PC/ compatible computers based on x86(8086, 80286, 386, 486 and Pentium) microprocessors normally have two COM ports. Both COM ports have RS232 type connectors. Many PCs use one each of the DB-25 and DB-9 RS232 connectors. The COM ports are designated as COM1 and COM2. We can connect the serial port to the COM 2 port of a PC for serial communication experiments. We use a DB9 connector in our arrangement.

**Light Emitting Diode (LED):**

A light-emitting diode (LED) is a semiconductor diode that emits incoherent narrow spectrum light when electrically biased in the forward direction of the pn-junction, as in the common LED circuit. This effect is a form of electroluminescence

While sending a message in the form of bits such as 1,the data is sent to the receiver side correspondingly the LED glows representing the data is being received simultaneously when we send 8 as a data the LED gets off .



**Color Coding**

Color Potential Difference

Infrared 1.6 V

Red 1.8 V to 2.1 V

Orange 2.2 V

Yellow 2.4 V

Green 2.6 V

Blue 3.0 V to 3.5 V

White 3.0 V to 3.5 V

Ultraviolet 3.5V



**ADVANTAGES**

* LEDs have many advantages over other technologies like lasers. As compared to laser diodes or IR sources
* LEDs have several advantages over conventional incandescent lamps. For one thing, they don't have a filament that will burn out, so they last much longer. Additionally, their small plastic bulb makes them a lot more durable. They also fit more easily into modern electronic circuits.
* The main advantage is **efficiency**. In conventional incandescent bulbs, the light-production process involves generating a lot of heat (the filament must be warmed). This is completely wasted energy, unless you're using the lamp as a heater, because a huge portion of the available electricity isn't going toward producing visible light.
* LEDs generate very little heat. A much higher percentage of the electrical power is going directly to generating light, which cuts down on the electricity demands considerably.
* LEDs offer advantages such as lower cost and longer service life. Moreover LEDs have very low power consumption and are easy to maintain. Many functions can be assigned to a robot easily using different colors of LEDs availible.

**Disadvantages of LEDs**

* LED performance largely depends on the ambient temperature of the operating environment.
* LEDs must be supplied with the correct current.
* LEDs do not approximate a "point source" of light, so cannot be used in applications needing a highly collimated beam.

But the disadvantages are quite negligible in this project as the negative properties of LEDs do not apply and the advantages far exceed the limitations. So we prefer to use the LED as our light source.

**POWER SUPPLY**

All digital circuits require regulated power supply. In this article we are going to learn how to get a regulated positive supply from the mains supply.

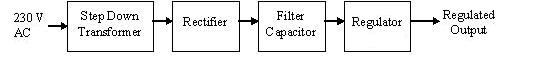
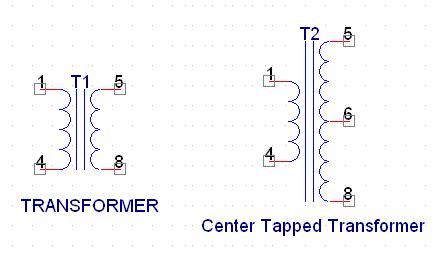


Figure 1 shows the basic block diagram of a fixed regulated power supply. Let us go through each block.

**TRANSFORMER**



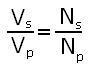
A transformer consists of two coils also called as “WINDINGS” namely PRIMARY & SECONDARY.

They are linked together through inductively coupled electrical conductors also called as CORE. A changing current in the primary causes a change in the Magnetic Field in the core & this in turn induces an alternating voltage in the secondary coil. If load is applied to the secondary then an alternating current will flow through the load. If we consider an ideal condition then all the energy from the primary circuit will be transferred to the secondary circuit through the magnetic field.

So Image

Image

The secondary voltage of the transformer depends on the number of turns in the Primary as well as in the secondary.

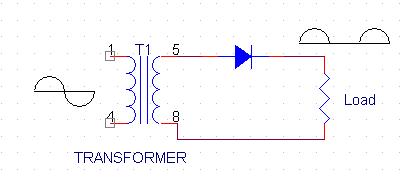


**Rectifier**

A rectifier is a device that converts an AC signal into DC signal. For rectification purpose we use a diode, a diode is a device that allows current to pass only in one direction i.e. when the anode of the diode is positive with respect to the cathode also called as forward biased condition & blocks current in the reversed biased condition.

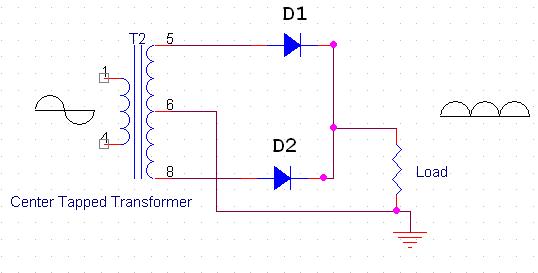
Rectifier can be classified as follows:

**1)      Half Wave rectifier.**



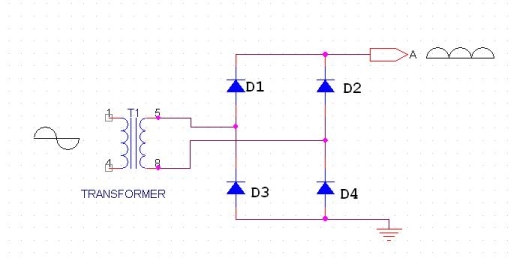
This is the simplest type of rectifier as you can see in the diagram a half wave rectifier consists of only one diode. When an AC signal is applied to it during the positive half cycle the diode is forward biased & current flows through it. But during the negative half cycle diode is reverse biased & no current flows through it. Since only one half of the input reaches the output, it is very inefficient to be used in power supplies.

**2)      Full wave rectifier.**



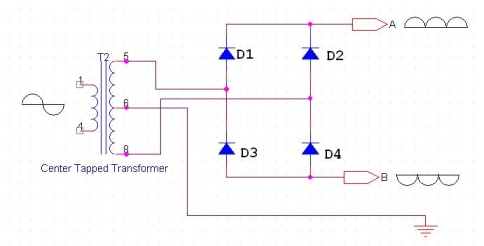
Half wave rectifier is quite simple but it is very inefficient, for greater efficiency we would like to use both the half cycles of the AC signal. This can be achieved by using a center tapped transformer i.e. we would have to double the size of secondary winding & provide connection to the center. So during the positive half cycle diode D1 conducts & D2 is in reverse biased condition. During the negative half cycle diode D2 conducts & D1 is reverse biased. Thus we get both the half cycles across the load.

One of the disadvantages of Full Wave Rectifier design is the necessity of using a center tapped transformer, thus increasing the size & cost of the circuit. This can be avoided by using the Full Wave Bridge Rectifier.

**3)      Bridge Rectifier.**

As the name suggests it converts the full wave i.e. both the positive & the negative half cycle into DC thus it is much more efficient than Half Wave Rectifier & that too without using a center tapped transformer thus much more cost effective than Full Wave Rectifier.

Full Bridge Wave Rectifier consists of four diodes namely D1, D2, D3 and D4. During the positive half cycle diodes D1 & D4 conduct whereas in the negative half cycle diodes D2 & D3 conduct thus the diodes keep switching the transformer connections so we get positive half cycles in the output.

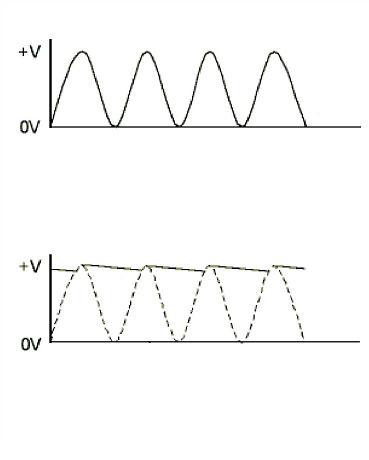


If we use a center tapped transformer for a bridge rectifier we can get both positive & negative half cycles which can thus be used for generating fixed positive & fixed negative voltages.

**FILTER CAPACITOR**

Even though half wave & full wave rectifier give DC output, none of them provides a constant output voltage. For this we require to smoothen the waveform received from the rectifier. This can be done by using a capacitor at the output of the rectifier this capacitor is also called as “FILTER CAPACITOR” or “SMOOTHING CAPACITOR” or “RESERVOIR CAPACITOR”. Even after using this capacitor a small amount of ripple will remain.

We place the Filter Capacitor at the output of the rectifier the capacitor will charge to the peak voltage during each half cycle then will discharge its stored energy slowly through the load while the rectified voltage drops to zero, thus trying to keep the voltage as constant as possible.



If we go on increasing the value of the filter capacitor then the Ripple will decrease. But then the costing will increase. The value of the Filter capacitor depends on the current consumed by the circuit, the frequency of the waveform & the accepted ripple.



Where,

Vr= accepted ripple voltage.( should not be more than 10% of  the voltage)

I= current consumed by the circuit in Amperes.

F= frequency of the waveform. A half wave rectifier has only one peak in one cycle so F=25hz

Whereas a full wave rectifier has Two peaks in one cycle so F=100hz.

**VOLTAGE REGULATOR**

A Voltage regulator is a device which converts varying input voltage into a constant regulated output voltage. Voltage regulator can be of two types

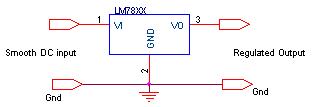
1)      Linear Voltage Regulator

      Also called as Resistive Voltage regulator because they dissipate the excessive voltage resistively as heat.

2)      Switching Regulators.

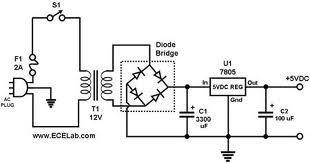
      They regulate the output voltage by switching the Current ON/OFF very rapidly. Since their output is either ON or OFF it dissipates very low power thus achieving higher efficiency as compared to linear voltage regulators. But they are more complex & generate high noise due to their switching action. For low level of output power switching regulators tend to be costly but for higher output wattage they are much cheaper than linear regulators.

The most commonly available Linear Positive Voltage Regulators are the 78XX series where the XX indicates the output voltage. And 79XX series is for Negative Voltage Regulators.



 After filtering the rectifier output the signal is given to a voltage regulator. The maximum input voltage that can be applied at the input is 35V.Normally there is a 2-3 Volts drop across the regulator so the input voltage should be at least 2-3 Volts higher than the output voltage. If the input voltage gets below the Vmin of the regulator due to the ripple voltage or due to any other reason the voltage regulator will not be able to produce the correct regulated voltage.

#### 3 Circuit diagram:



**Fig 2.3. Circuit Diagram of power supply**

#### IC 7805:

7805 is an integrated three-terminal positive fixed linear voltage regulator. It supports an input voltage of 10 volts to 35 volts and output voltage of 5 volts. It has a current rating of 1 amp although lower current models are available. Its output voltage is fixed at 5.0V. The 7805 also has a built-in current limiter as a safety feature. 7805 is manufactured by many companies, including National Semiconductors and Fairchild Semiconductors.

The 7805 will automatically reduce output current if it gets too hot.The last two digits represent the voltage; for instance, the 7812 is a 12-volt regulator. The 78xx series of regulators is designed to work in complement with the 79xx series of negative voltage regulators in systems that provide both positive and negative regulated voltages, since the 78xx series can't regulate negative voltages in such a system.

The 7805 & 78 is one of the most common and well-known of the 78xx series regulators, as it's small component count and medium-power regulated 5V make it useful for powering TTL devices.

**Table 2.1. Specifications of IC7805**

|  |  |
| --- | --- |
| **SPECIFICATIONS** | **IC 7805** |
| Vout | 5V |
| Vein - Vout Difference | 5V - 20V |
| Operation Ambient Temp | 0 - 125°C |
| Output Imax | 1A |

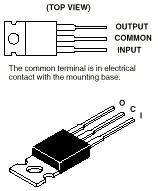
A variable regulated power supply, also called a variable bench power supply, is one where you can continuously adjust the output voltage to your requirements. Varying the output of the power supply is the recommended way to test a project after having double checked parts placement against circuit drawings and the parts placement guide.

This type of regulation is ideal for having a simple variable bench power supply. Actually this is quite important because one of the first projects a hobbyist should undertake is the construction of a variable regulated power supply. While a dedicated supply is quite handy e.g. 5V or 12V, it's much handier to have a variable supply on hand, especially for testing.

Most digital logic circuits and processors need a 5 volt power supply. To use these parts we need to build a regulated 5 volt source. Usually you start with an unregulated power supply ranging from 9 volts to 24 volts DC (A 12 volt power supply is included with the [Beginner Kit](http://www.iguanalabs.com/1stled.htm) and the [Microcontroller Beginner Kit](http://www.iguanalabs.com/mbkit.htm).). To make a 5 volt power supply, we use a LM7805 voltage regulator IC (Integrated Circuit). The IC is shown below.

**POWER SUPPLY**:

In my project I used9 volts transformer for continuous power supply. Why I am using this means to continuous power will come. Otherwise If I use a battery some times the total currents will loss so that’s way I am using A.C Transformer. A.C transformer is giving the input to Bridge Rectifier. Bridge Rectifier converts A.C to D.C. After that we are using one filter capacitor 1000uf/25v electrolytic capacitor .We connecting this capacitor in parallel section. The main purpose of this capacitor is if there is any alternate peaks we need to reduce that peaks. Nothing but a filtering that repull’s. After that we are using LM7805 Regulator Most digital logic circuits and processors need a 5 volt power supply. To use these parts we need to build a regulated 5 volt source. We make a 5 volt power supply, The LM7805 is simple to use. First connect the positive lead of our unregulated DC power supply Input pin, connect the negative lead to the Common pin and then when we turn on the power, we get a 5 volt supply from the Output pin. Here we are using one red color led to indicate the power.





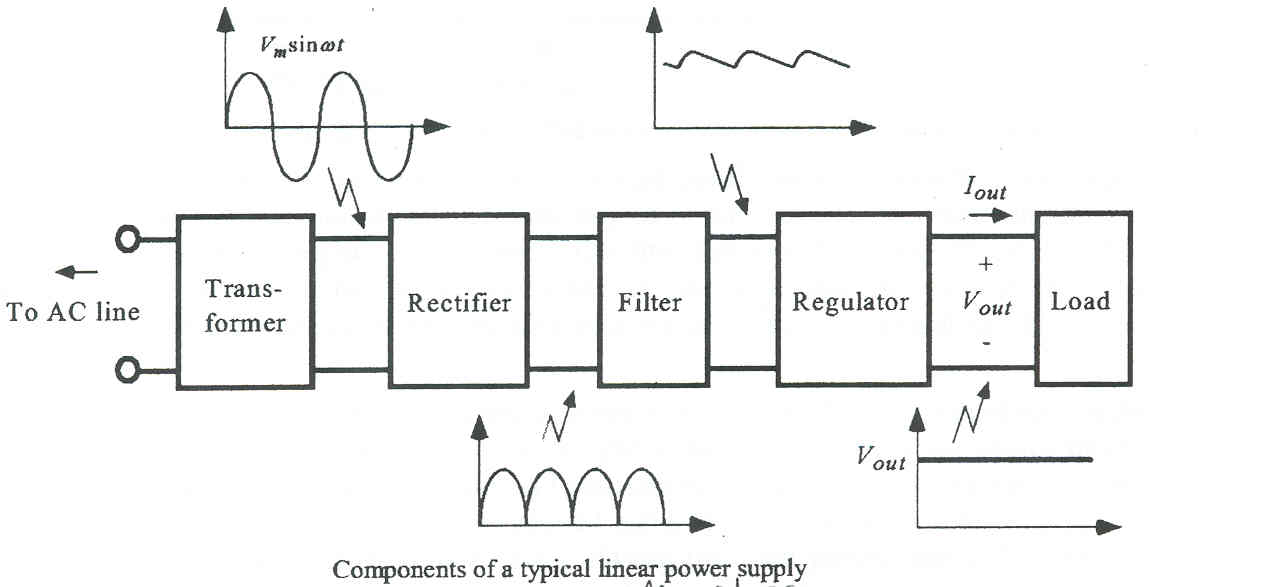
We are using General purpose PCB. To mount or place the components on the PCB. After that

The LM7805 is simple to use. You simply connect the positive lead of your unregulated DC power supply (anything from 9VDC to 24VDC) to the Input pin, connect the negative lead to the Common pin and then when you turn on the power, you get a 5 volt supply from the Output pin.

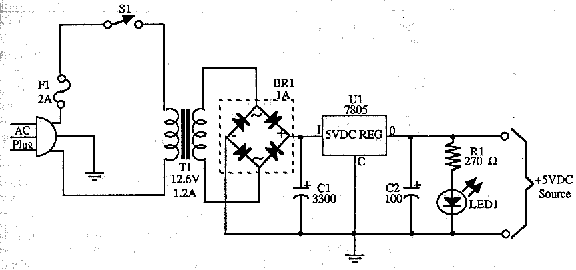
**7.2 CIRCUIT FEATURES:**

* Brief description of operation: Gives out well regulated +5V output, output current capability of 100 mA
* Circuit protection: Built-in overheating protection shuts down output when regulator IC gets too hot
* Circuit complexity: Very simple and easy to build
* Circuit performance: Very stable +5V output voltage, reliable operation
* Availability of components: Easy to get, uses only very common basic components
* Design testing: Based on datasheet example circuit, I have used this circuit succesfully as part of many electronics projects
* Applications: Part of electronics devices, small laboratory power supply
* Power supply voltage: Unreglated DC 8-18V power supply
* Power supply current: Needed output current + 5 mA
* Component costs: Few dollars for the electronics components + the input transformer cost

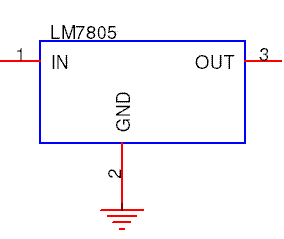
**7.3 BLOCK DIAGRAM:**



**7.4 CIRCUIT DIAGRAM:**



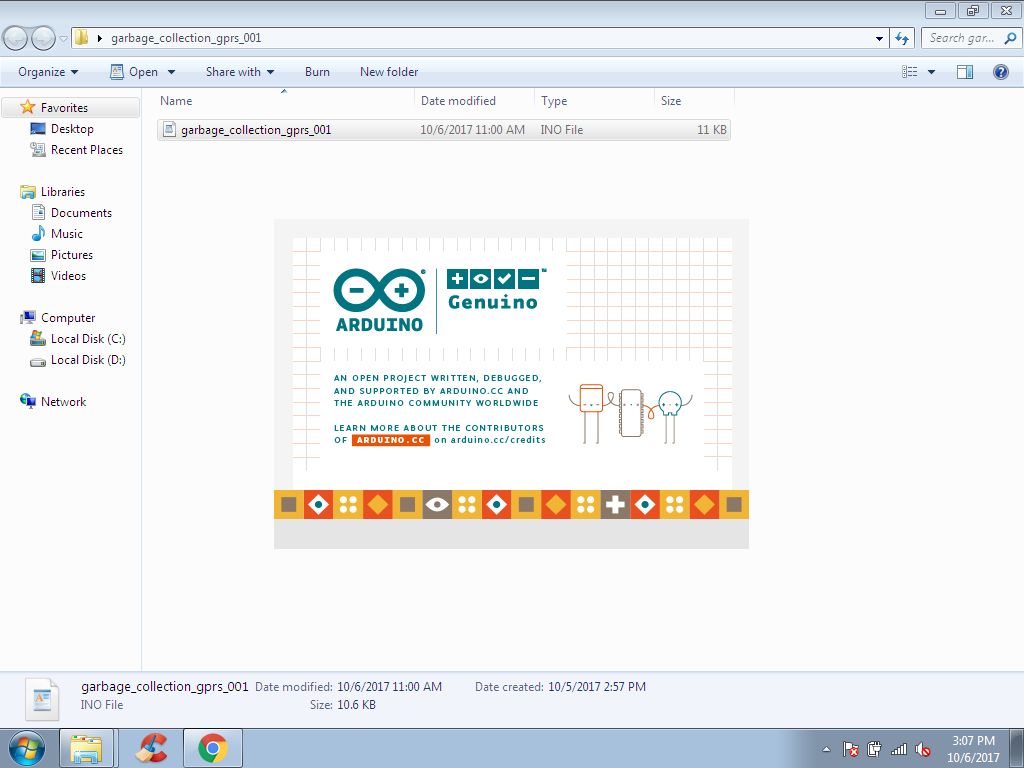
The power supply unit is used to provide constant 5 V dc supply to the peripherals. The 230 V ac is converted into 9 V ac by using a transformer and then a bridge rectifier rectifies it to a 9 V dc with ac ripples. This is then filtered by electrolytic capacitors used across the rectifier output. LM7805 regulator is employed to obtain a constant 5 V dc at the output.

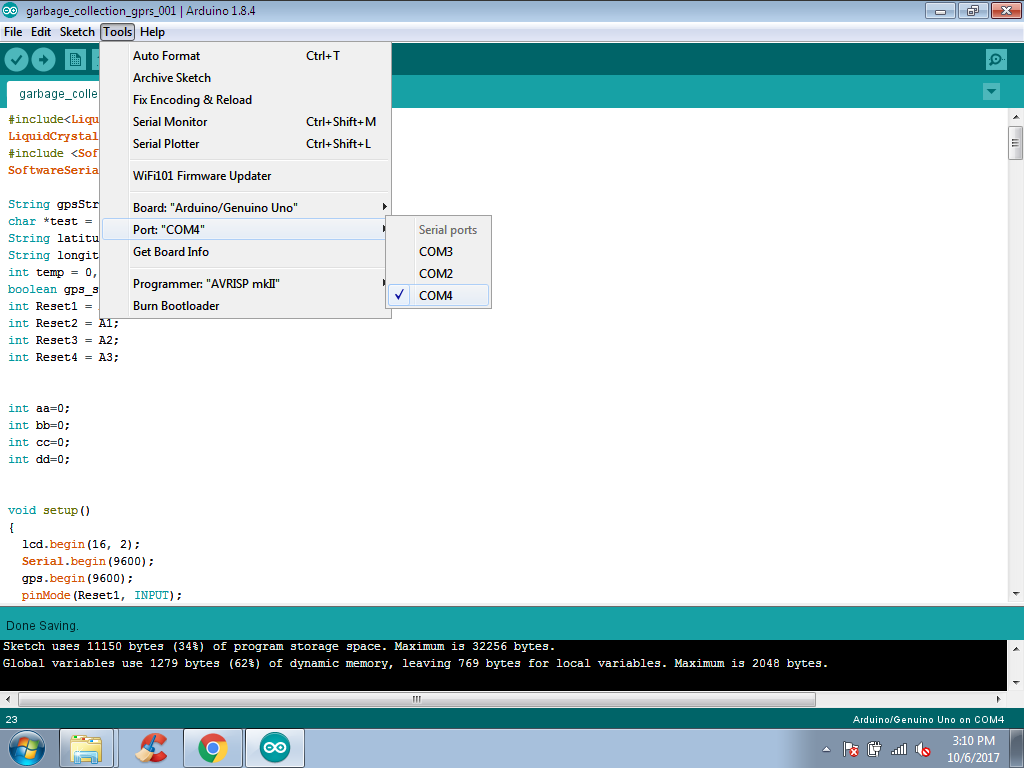


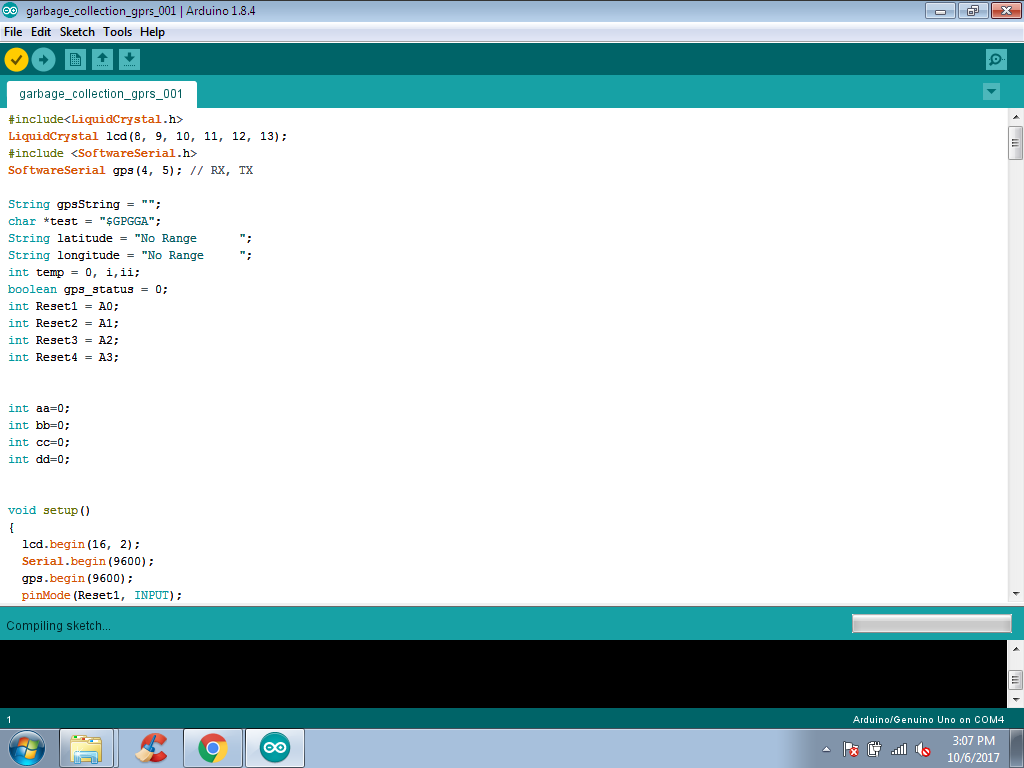
**Regulator LM7805.**

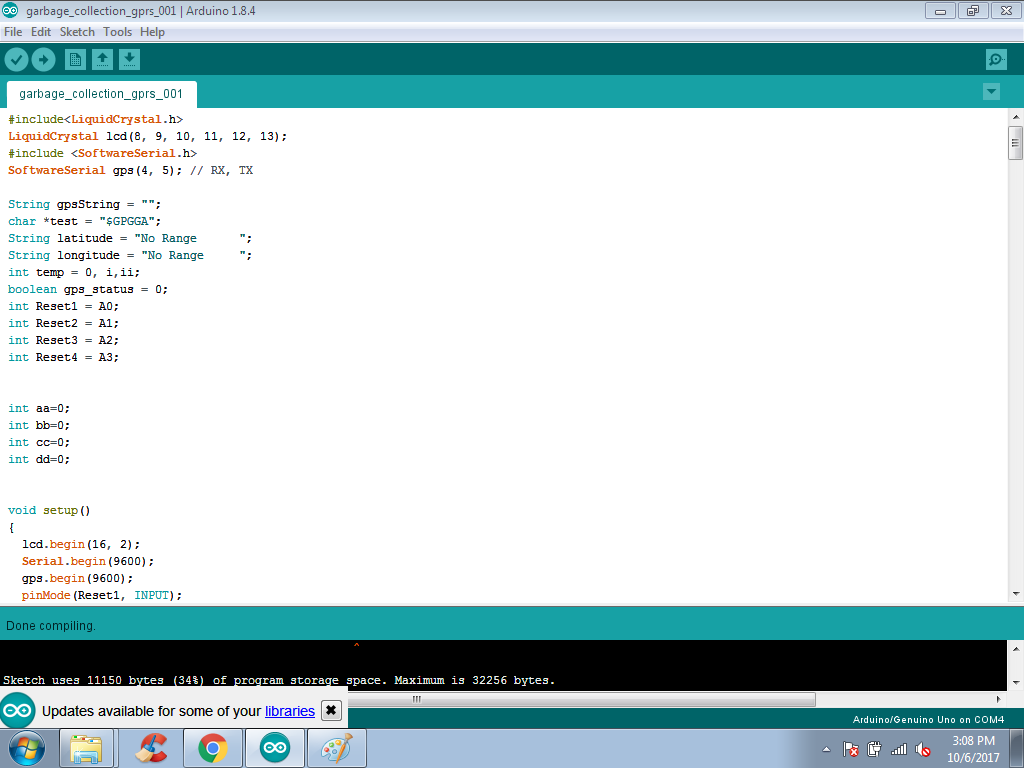
This is a digital IC used as a regulator for regulating the output. LM7805 provides a constant 5V dc supply as output. For this it needs at least 8V dc as input.

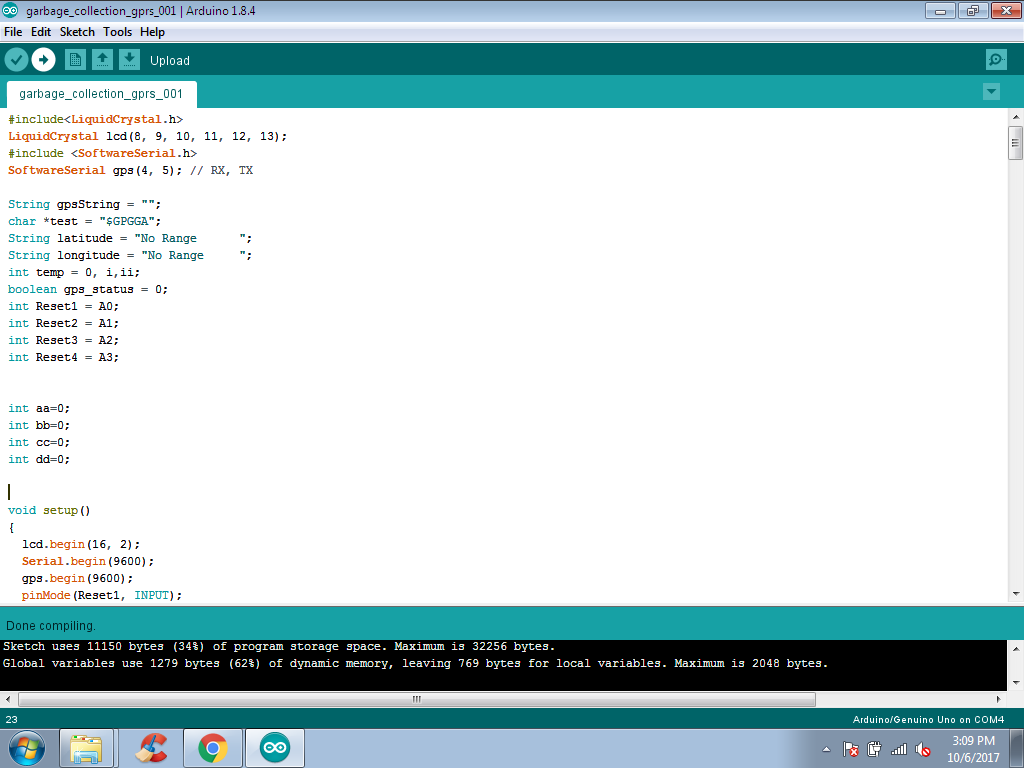
**SOFTWARE COMPILER**

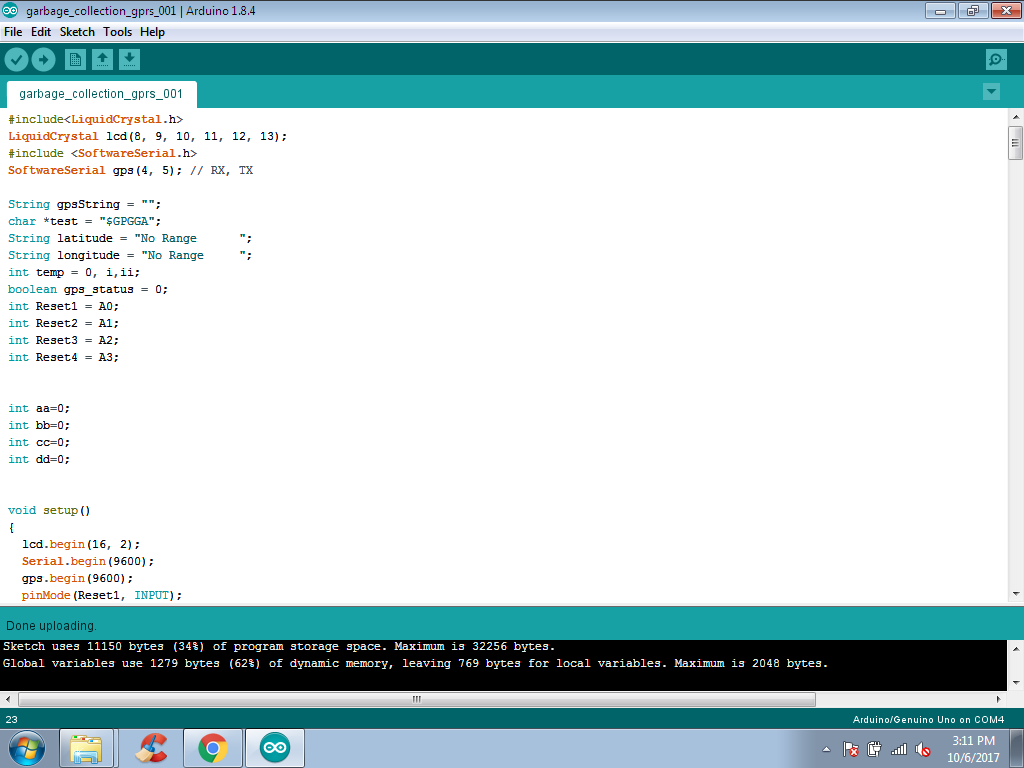












**SOFTWARE CODE**

#include <ESP8266WiFi.h>

#include <WiFiClient.h>

#include <ESP8266WebServer.h>

#include <ESP8266HTTPClient.h>

#include<LiquidCrystal.h>

LiquidCrystal lcd(16, 5, 4, 0, 2, 14);

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#define trigPin1 12

#define echoPin1 13

long duration, distance, FIRSTSensor;

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <SoftwareSerial.h>

SoftwareSerial gps(15, 9); // RX, TX

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int i=0;

int gps\_status=0;

float latitude=0;

float logitude=0;

String gpsString="";

char \*test="$GPRMC";

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

const char\* ssid = "project"; // Input your wifi network name

const char\* password = "project1234"; // Input your wifi password

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

const char \*host = "http://iotvehicle.co.in";

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

String aa;

int ee=0;

String bb;

int gg=0;

String cc;

////////////////////////////////////////////////////////////////

void initModule(String cmd, char \*res, int t)

{

while(1)

{

Serial.println(cmd);

delay(100);

while(Serial.available()>0)

{

if(Serial.find(res))

{

Serial.println(res);

delay(t);

return;

}

else

{

Serial.println("Error");

}

}

delay(t);

}

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

void setup()

{

lcd.begin(16, 2);

delay(1000);

Serial.begin(9600);

gps.begin(9600);

WiFi.mode(WIFI\_OFF);

delay(1000);

WiFi.mode(WIFI\_STA);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED)

{

delay(500);

lcd.print(".");

}

pinMode(trigPin1, OUTPUT);

pinMode(echoPin1, INPUT);

lcd.clear();

lcd.setCursor(0,0);lcd.print(" IOT BASED ");

lcd.setCursor(0,1);lcd.print("SMART DUSTBIN ");

delay (5000);lcd.clear();

lcd.clear();

lcd.setCursor(0,0);lcd.print("Using NodeMCU ");

lcd.setCursor(0,1);lcd.print("ESP8266");

delay (5000);lcd.clear();

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

lcd.setCursor(0,0);lcd.print("Waiting For GPS");

lcd.setCursor(0,1);lcd.print(" Signal ");

get\_gps();show\_coordinate();

delay(5000);lcd.clear();

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

lcd.setCursor(0,0);lcd.print("Initializing");

lcd.setCursor(0,1);lcd.print("GSM MODEM");delay(1000);

initModule("AT","OK",1000);

initModule("AT+CPIN?","READY",1000);

initModule("AT+CMGF=1","OK",1000);

initModule("AT+CNMI=2,2,0,0,0","OK",1000);

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

lcd.setCursor(0,0);lcd.print("Initialized");

lcd.setCursor(0,1);lcd.print("Successfully");delay(2000);lcd.clear();

get\_gps();show\_coordinate();delay(5000);lcd.clear();

}

void loop()

{

SonarSensor(trigPin1, echoPin1);

FIRSTSensor = distance;

///////////////////////////////////////////////////////////////////////////////////////

lcd.setCursor(0,0);lcd.print("S1:");lcd.setCursor(4,0);lcd.print(FIRSTSensor);delay(500);

///////////////////////////////////////////////////////

if((FIRSTSensor >= 1) & (FIRSTSensor <= 4))

{cc="01";aa="3";lcd.setCursor(0,1);lcd.print("GARBAGE BIN FULL");delay(500);gg=gg+1;bb="95";

if(gg==5){ get\_gps();show\_coordinate();delay(2000);lcd.clear();tracking1();delay(500);}}

if((FIRSTSensor >= 5) & (FIRSTSensor <= 8))

{cc="01";aa="6";lcd.setCursor(0,1);lcd.print("GARBAGE BIN 85% ");delay(500);gg=0;bb="85";}

if((FIRSTSensor >= 9) & (FIRSTSensor <= 12))

{cc="01";aa="10";lcd.setCursor(0,1);lcd.print("GARBAGE BIN 70% ");delay(500);gg=0;bb="70";}

if((FIRSTSensor >= 13) & (FIRSTSensor <= 16))

{cc="01";aa="15";lcd.setCursor(0,1);lcd.print("GARBAGE BIN 50% ");delay(500);gg=0;bb="50";}

if((FIRSTSensor >= 17) & (FIRSTSensor <= 20))

{cc="01";aa="18";lcd.setCursor(0,1);lcd.print("GARBAGE BIN 25% ");delay(500);gg=0;bb="25";}

if((FIRSTSensor >= 21) & (FIRSTSensor <= 24))

{cc="01";aa="22";lcd.setCursor(0,1);lcd.print("GARBAGE BIN 10% ");delay(500);gg=0;bb="10";}

if((FIRSTSensor >= 25) & (FIRSTSensor <= 200))

{cc="01";aa="30";lcd.setCursor(0,1);lcd.print("GARBAGE BIN LOW ");delay(500);gg=0;bb="5";}

/////////////////////////////////////////////////////

ee=ee+1;

if(ee==5)

{

HTTPClient http; //Declare object of class HTTPClient

String postData,temp,lat,lon,gid,gct;

postData = "temp=" + aa + "&lat=" + latitude + "&lon=" + logitude + "&gid=" + cc + "&gct=" + bb;

http.begin("http://iotvehicle.co.in/iot\_waste\_dustbin/put\_data.php?"); //Specify request destination

http.addHeader("Content-Type", "application/x-www-form-urlencoded"); //Specify content-type header

int httpCode = http.POST(postData); //Send the request

String payload = http.getString(); //Get the response payload

Serial.print(httpCode); //Print HTTP return code

Serial.println(payload); //Print request response payload

http.end(); //Close connection

delay(3000); //Post Data at every 5 seconds

delay(500);lcd.clear();

ee=0;

}

}

///////////////////////////////////////////////////////////////////////////////////////////

void SonarSensor(int trigPin,int echoPin)

{

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

duration = pulseIn(echoPin, HIGH);

distance = (duration/2) / 29.1;

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void gpsEvent()

{

gpsString="";

while(1)

{

while (gps.available()>0) //Serial incoming data from GPS

{

char inChar = (char)gps.read();

gpsString+= inChar; //store incoming data from GPS to temparary string str[]

i++;

if (i < 7)

{

if(gpsString[i-1] != test[i-1]) //check for right string

{

i=0;

gpsString="";

}

}

if(inChar=='\r')

{

if(i>65)

{

gps\_status=1;

break;

}

else

{

i=0;

}

}

}

if(gps\_status)

break;

}

}

///////////////////////////////////////////////////////////////////////////////////////////

void get\_gps()

{

lcd.clear();

lcd.print("Getting GPS Data");

lcd.setCursor(0,1);

lcd.print("Please Wait.....");

gps\_status=0;

int x=0;

while(gps\_status==0)

{

gpsEvent();

int str\_lenth=i;

coordinate2dec();

i=0;x=0;

str\_lenth=0;

}

}

///////////////////////////////////////////////////////////////////////////////////////////

void show\_coordinate()

{

lcd.clear();

lcd.print("Lat:");

lcd.print(latitude);

lcd.setCursor(0,1);

lcd.print("Log:");

lcd.print(logitude);

delay(2000);

lcd.clear();

}

///////////////////////////////////////////////////////////////////////////////////////////

//$GPRMC,053508.00,A,1725.64574,N,07835.11697,E,0.041,,121217,,,D\*79

void coordinate2dec()

{

String lat\_degree="";

for(i=19;i<=20;i++)

lat\_degree+=gpsString[i];

String lat\_minut="";

for(i=21;i<=27;i++)

lat\_minut+=gpsString[i];

String log\_degree="";

for(i=32;i<=34;i++)

log\_degree+=gpsString[i];

String log\_minut="";

for(i=35;i<=41;i++)

log\_minut+=gpsString[i];

float minut= lat\_minut.toFloat();

minut=minut/60;

float degree=lat\_degree.toFloat();

latitude=degree+minut;

minut= log\_minut.toFloat();

minut=minut/60;

degree=log\_degree.toFloat();

logitude=degree+minut;

}

///////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////

void init\_sms1()

{Serial.println("AT+CMGF=1");delay(400);Serial.println("AT+CMGS=\"6383965677\""); delay(400);}

/////////////////////////////////////////////////////////////////////////////////////////////////////

void send\_data(String message)

{ Serial.print(message);delay(200);}

//////////////////////////////////////////////////////////////////////////////////////////////////////////

void lcd\_status()

{lcd.clear();lcd.print("Message Sent"); delay(3000); lcd.clear(); return;}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

void tracking1()

{

init\_sms1();lcd.clear();lcd.print("Sending SMS ");

send\_data("GARBAGE BIN FULL\n");

send\_data("PLZ COLLECT\n");

Serial.print("https://www.google.com/maps/place/");

Serial.print(latitude,6);Serial.print(",");Serial.print(logitude,6);Serial.write(26);delay(2000);lcd\_status();

Serial.write(26);delay(2000);

lcd\_status();

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////