**Smart Fire Detection Using OpenCV and Python With Arduino - GSM - Call Alert and SMS Notification**

**ABSTRACT:**

Internet of things is an interconnection of physical devices embedded with electronics, software, sensor which is capable of collecting data from the surrounding and sending data over internet is called IOT. The fire detection gathers all of the techniques and processes that contribute to early detection of a fire. We identify three main categories: Smoke detection, Flame detection and Temperature detection. Automatic fire alarm system provides real-time surveillance, monitoring and automatic alarm. An automatic fire alarm system based on wireless sensor networks is developed, which is designed for high-rise buildings. To provide early extinguishing of a fire disaster, large numbers of detectors which periodically measure smoke concentration or temperature are deployed in buildings. In this paper will we present the different techniques we had been already used to detect fire. Some of those techniques include fire detection using image processing and sensors, fire detection using CCTV technology, Fire detection using GSM and GPS with sms and call alert then motor pump is on

Key Words –ARDUINO microcontroller, Fire alarm system, Wireless sensor networks, Sensor etc AND GPRS MODEM.

**PURPOSE:**

The purpose of the project is to provide fire security for in home r any where . In case of any fire alert the gprs modem will ping the data from tower and send the loaction to mobile numbers. Open cv with python camera to detect the fire and send to Arduino.

**Block Diagram:**

**spinkler**

**Arduino MICRO CONTROLLER**

**POWER  
 SUPPLY**

**LCD**

**Python code detect fire**

**GSM MODEM**

**SIM800l**

**BUZZER**

**Led’s**

**Red/green**

**GSM:**

GSM (Global System for Mobile communications) is the technology that underpins most of the world's mobile phone networks. The GSM platform is a hugely successful wireless technology and an unprecedented story of global achievement and cooperation. GSM has become the world's fastest growing communications technology of all time and the leading global mobile standard, spanning 218 countries. GSM is an open, digital cellular technology used for transmitting mobile voice and data services. GSM operates in the 900MHz and 1.8GHz bands GSM supports data transfer speeds of up to 9.6 kbps, allowing the transmission of basic data services such as SMS.

**SOFTWARES:**

* Embedded C
* ARDUINO IDE
* Uc-Flash
* Express PCB

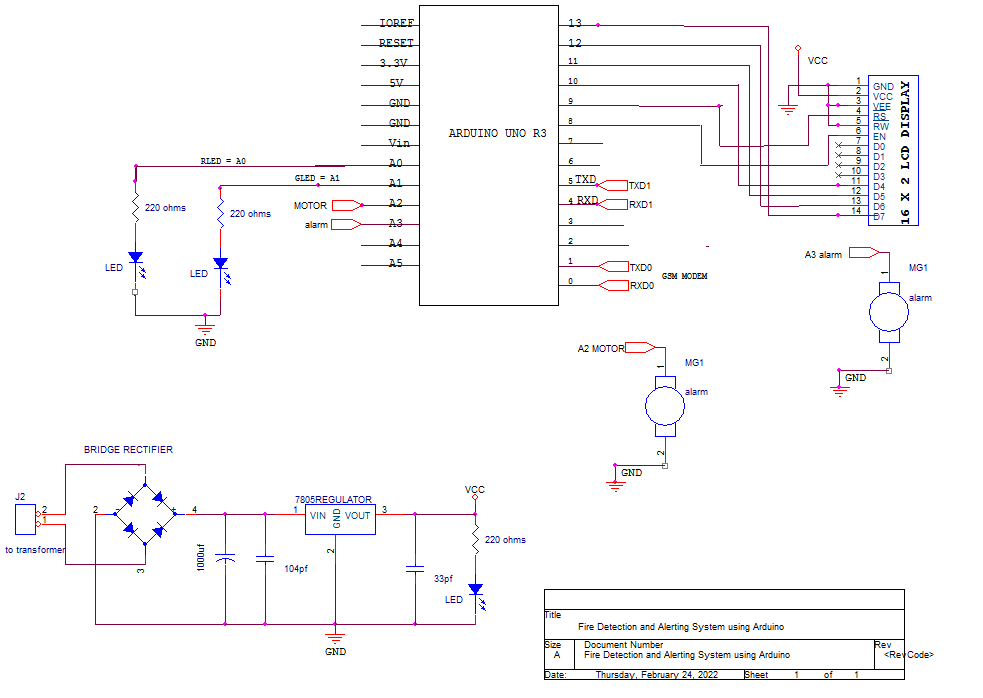
**HARDWARES:**

* Micro Controller
* Power Supply
* Gsm module
* Laptop
* Usb cable
* LCD
* Led’s
* SPINKLER

**RESULT: fire alert**

According to this project, we can provide safety system for home and also protect in where in emergency conditions

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

Overview:

The Uno is a microcontroller board based on the [ATmega328P.](http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf) It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.

"Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform; for an extensive list of current, past or outdated boards see the Arduino index of boards.

You can find [here](https://www.arduino.cc/en/Main/warranty) your board warranty informations.

Getting Started

You can find in the [Getting Started section](https://www.arduino.cc/en/Guide/HomePage) all the information you need to configure your board, use the [Arduino Software (IDE)](https://www.arduino.cc/en/Main/Software), and start tinker with coding and electronics.

Need Help?

* On the Software [on the Arduino Forum](https://forum.arduino.cc/index.php?board=63.0)
* On Projects [on the Arduino Forum](https://forum.arduino.cc/index.php?board=3.0)
* On the Product itself through [our Customer Support](https://store.arduino.cc/index.php?main_page=contact_us&language=en)

Technical specs

|  |  |
| --- | --- |
| Microcontroller | [ATmega328P](http://www.atmel.com/Images/doc8161.pdf) |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limit) | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) |
| PWM Digital I/O Pins | 6 |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 20 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 32 KB (ATmega328P) of which 0.5 KB used by bootloader |
| SRAM | 2 KB (ATmega328P) |
| EEPROM | 1 KB (ATmega328P) |
| Clock Speed | 16 MHz |
| Length | 68.6 mm |
| Width | 53.4 mm |
| Weight | 25 g |

**Basic Arduino code definitions:**

**setup( ):** A function present in every Arduino sketch. Run once before the loop( ) function. Often used to set pinmode to input or output. The setup( ) function looks like:

*void setup( ){*

*//code goes here*

*}*

**loop( ):** A function present in every single Arduino sketch. This code happens over and over again. The loop( ) is where (almost) everything happens. The one exception to this is setup( ) and variable declaration. ModKit uses another type of loop called “forever( )” which executes over Serial. The loop( ) function looks like:

*void loop( ) {*

*//code goes here*

*}*

**input:** A pin mode that intakes information.

**output:** A pin mode that sends information.

**HIGH:** Electrical signal present (5V for Uno). Also ON or True in boolean logic.

**LOW:** No electrical signal present (0V). Also OFF or False in boolean logic.

**digitalRead:** Get a HIGH or LOW reading from a pin already declared as an input.

**digitalWrite:** Assign a HIGH or LOW value to a pin already declared as an output.

**analogRead:** Get a value between or including 0 (LOW) and 1023 (HIGH). This allows you to get readings from analog sensors or interfaces that have more than two states.

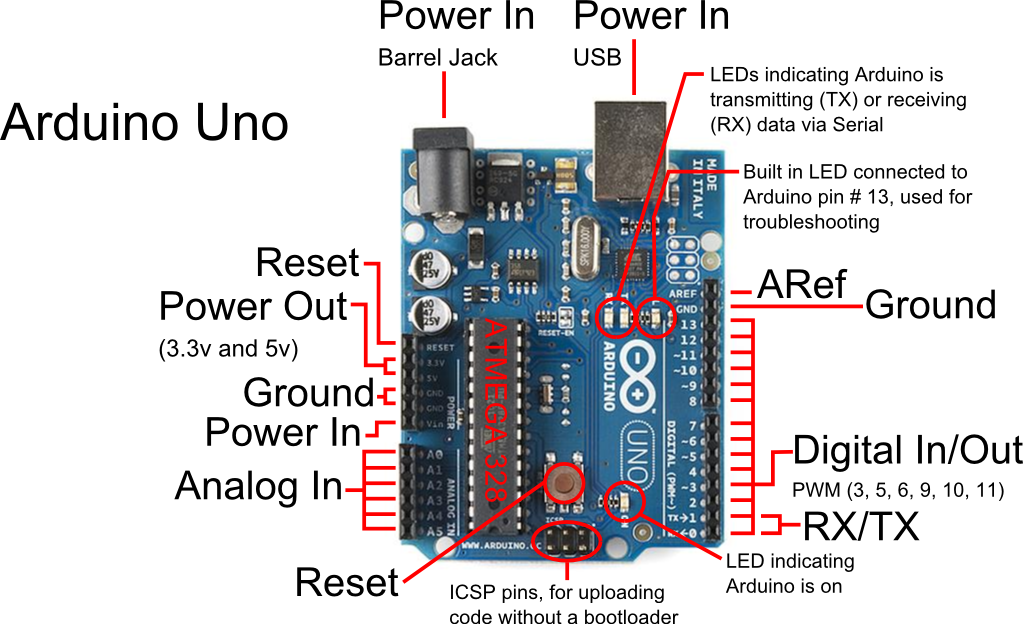
**analogWrite:** Assign a value between or including 0 (LOW) and 255 (HIGH). This allows you to set output to a PWM value instead of just HIGH or LOW.

**PWM:** Stands for Pulse-Width Modulation, a method of emulating an analog signal through a digital pin. A value between or including 0 and 255. Used with analogWrite.

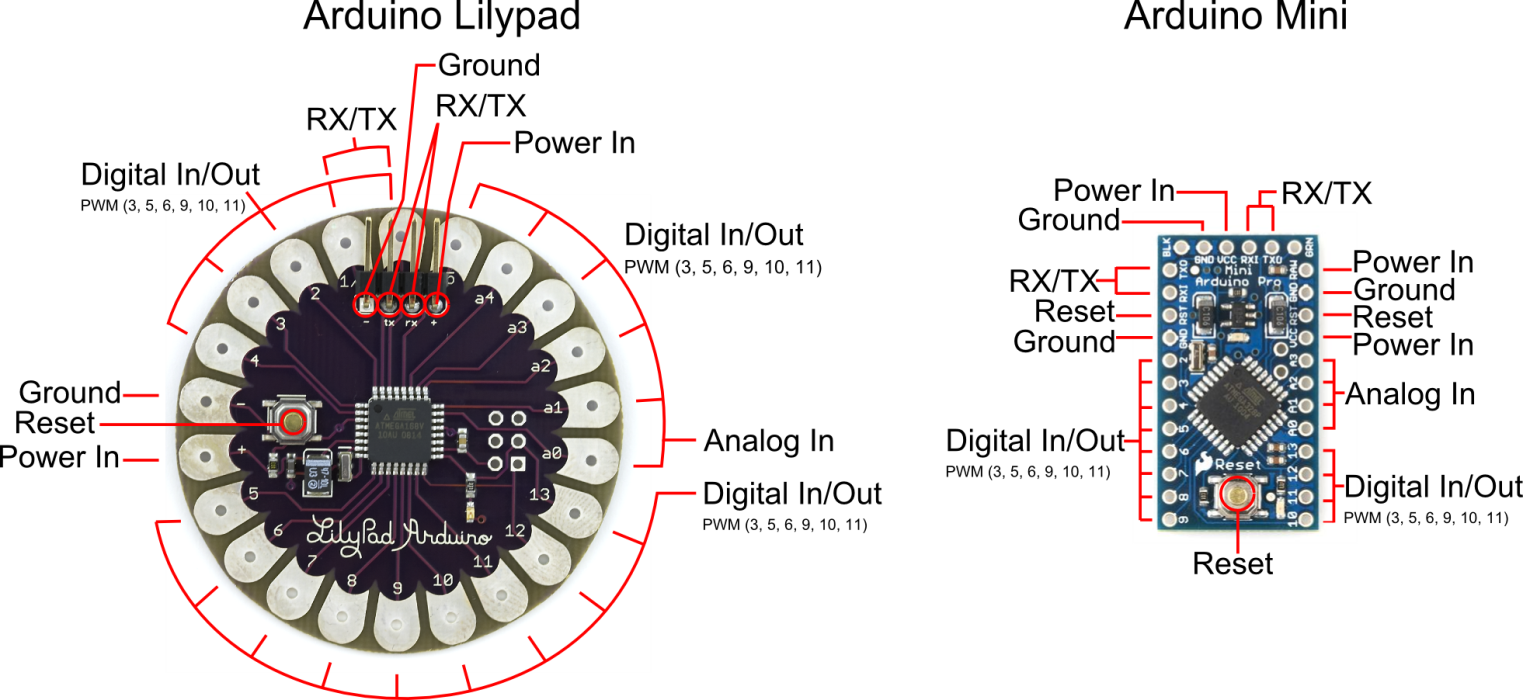
**Arduino Uno pin type definitions: (Take a look at your Arduino board)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Reset** | **3v3** | **5v** | **Gnd** | **Vin** | **Analog In** | **RX/TX** | **Digital** | **PWM(~)** | **AREF** |
| Resets Arduino sketch on board | 3.3 volts in and out | 5 volts in and out | Ground | Voltage in for sources  over 7V (9V - 12V) | Analog inputs, can also be used as Digital | Serial comm. Receive and Transmit | Input or output, HIGH or LOW | Digital pins with output option of PWM | External reference voltage used for analog |

**Basic Arduino Pin Reference Sheet**



These boards below use the same micro-controller, just in a different package. The Lilypad is designed for use with conductive thread instead of wire and the Arduino Mini is simply a smaller package without the USB, Barrel Jack and Power Outs. Other boards in the Arduino family can be found at <http://arduino.cc/en/Main/Hardware>.



**Voltage Dividers**

**What is a voltage divider?**

Voltage dividers are a way to produce a voltage that is a fraction of the original voltage.

**Why is a voltage divider useful?**

One of the ways this is useful is when you want to take readings from a circuit that has a voltage beyond the limits of your input pins. By creating a voltage divider you can be sure that you are getting an accurate reading of a voltage from a circuit. Voltage dividers are also used to provide an analog Reference signal.

**What is in a voltage divider?**

A voltage divider has three parts; two resistors and a way to read voltage between the two resistors.

**How do you put together a voltage divider?**

It's really pretty easy. Here is a schematic and explanation detailing how:

|  |  |
| --- | --- |
|  | Often resistor # 1 is a resistor with a value that changes, possibly a sensor or a potentiometer.  Resistor # 2 has whatever value is needed to create the ratio the user decides is acceptable for the voltage divider output.  The Voltage In and Ground portions are just there to establish which way the electrical current is heading, there can be any number of circuits before and after the voltage divider.  Here is the equation that represents how a voltage divider works:    If both resistors have the same value then Voltage Out is equal to ½ Voltage In. |

**Ok, how is this voltage divider information used?**

It depends on what you want to do with it really. There are two different purposes outlined above for the voltage divider, we will go over both.

If you wish to use the voltage divider as a sensor reading device first you need to know the maximum voltage allowed by the analog inputs you are using to read the signal. On an Arduino this is 5V. So, already we know the maximum value we need for Vout. The Vin is simply the amount of voltage already present on the circuit before it reaches the first resistor. You should be able to find the maximum voltage your sensor outputs by looking on the Datasheet, this is the maximum amount of voltage your sensor will let through given the voltage in of your circuit. Now we have exactly one variable left, the value of the second resistor. Solve for R2 and you will have all the components of your voltage divider figured out! We solve for R1's highest value because a smaller resistor will simply give us a smaller signal which will be readable by our analog inputs.

Powering an analog Reference is exactly the same as reading a sensor except you have to calculate for the Voltage Out value you want to use as the analog Reference.

Given three of these values you can always solve for the missing value using a little algebra, making it pretty easy to put together your own voltage divider. The S.I.K. has many voltage dividers in the example circuits. These include: Circuits # 7, 8, 9, 13 and 14. **SIK Worksheets v.1.0 Name:**

**Digital Date:**

All of the electrical signals that the Arduino works with are either Analog or Digital. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

**Digital**

An electronic signal transmitted as binary code that can be either the presence or absence of current, high and low voltages or short pulses at a particular frequency.

Humans perceive the world in analog, but robots, computers and circuits use Digital. A digital signal is a signal that has only two states. These states can vary depending on the signal, but simply defined the states are ON or OFF, never in between.

In the world of Arduino, Digital signals are used for everything with the exception of Analog Input. Depending on the voltage of the Arduino the ON or HIGH of the Digital signal will be equal to the system voltage, while the OFF or LOW signal will always equal 0V. This is a fancy way of saying that on a 5V Arduino the HIGH signals will be a little under 5V and on a 3.3V Arduino the HIGH signals will be a little under 3.3V.

To receive or send Digital signals the Arduino uses Digital pins # 0 - # 13. You may also setup your Analog In pins to act as Digital pins. To set up Analog In pins as Digital pins use the command:

*pinMode(pinNumber, value);*

where pinNumber is an Analog pin (A0 – A5) and value is either INPUT or OUTPUT. To setup Digital pins use the same command but reference a Digital pin for pinNumber instead of an Analog In pin. Digital pins default as input, so really you only need to set them to OUTPUT in pinMode. To read these pins use the command:

*digitalRead(pinNumber);*

where pinNumber is the Digital pin to which the Digital component is connected. The digitalRead command will return either a HIGH or a LOW signal. To send a Digital signal to a pin use the command:

*digitalWrite(pinNumber, value);*

where pinNumber is the number of the pin sending the signal and value is either HIGH or LOW.

The Arduino also has the capability to output a Digital signal that acts as an Analog signal, this signal is called Pulse Width Modulation (PWM). Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. To output a PWM signal use the command:

*analogWrite(pinNumber, value);*

where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

**Examples of Digital:**

Values: On/Off, Men's room/Women's room, pregnancy, consciousness, the list goes on....

Sensors/Interfaces: Buttons, Switches, Relays, CDs, etc....

**Things to remember about Digital:**

* Digital Input/Output uses the Digital pins, but Analog In pins can be used as Digital
* To receive a Digital signal use: *digitalRead(pinNumber);*
* To send a Digital signal use: *digitalWrite(pinNumber, value);*
* Digital Input and Output are always either HIGH or LOW

**SIK Worksheets v.1.0 Name:**

**Analog Date:**

All of the electrical signals that the Arduino works with are either Analog or Digital. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

**Analog**

A continuous stream of information with values between and including 0% and 100%.

Humans perceive the world in analog. Everything we see and hear is a continuous transmission of information to our senses. The temperatures we perceive are never 100% hot or 100% cold, they are constantly changing between our ranges of acceptable temperatures. (And if they are out of our range of acceptable temperatures then what are we doing there?) This continuous stream is what defines analog data. Digital information, the complementary concept to Analog, estimates analog data using only ones and zeros.

In the world of Arduino an Analog signal is simply a signal that can be HIGH (on), LOW (off) or anything in between these two states. This means an Analog signal has a voltage value that can be anything between 0V and 5V (unless you mess with the Analog Reference pin). Analog allows you to send output or receive input about devices that run at percentages as well as on and off. The Arduino does this by sampling the voltage signal sent to these pins and comparing it to a voltage reference signal (5V). Depending on the voltage of the Analog signal when compared to the Analog Reference signal the Arduino then assigns a numerical value to the signal somewhere between 0 (0%) and 1023 (100%). The digital system of the Arduino can then use this number in calculations and sketches.

To receive Analog Input the Arduino uses Analog pins # 0 - # 5. These pins are designed for use with components that output Analog information and can be used for Analog Input. There is no setup necessary, and to read them use the command:

*analogRead(pinNumber);*

where pinNumber is the Analog In pin to which the the Analog component is connected. The analogRead command will return a number including or between 0 and 1023.

The Arduino also has the capability to output a digital signal that acts as an Analog signal, this signal is called Pulse Width Modulation (PWM). Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. To output a PWM signal use the command:

*analogWrite(pinNumber, value);*

where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). On the Arduino UNO PWM pins are signified by a ~ sign. For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

**Examples of Analog:**

Values: Temperature, volume level, speed, time, light, tide level, spiciness, the list goes on....

Sensors: Temperature sensor, Photoresistor, Microphone, Turntable, Speedometer, etc....

**Things to remember about Analog:**

* Analog Input uses the Analog In pins, Analog Output uses the PWM pins
* To receive an Analog signal use: *analogRead(pinNumber);*
* To send a PWM signal use: *analogWrite(pinNumber, value);*
* Analog Input values range from 0 to 1023 (1024 values because it uses 10 bits, 210)
* PWM Output values range from 0 to 255 (256 values because it uses 8 bits, 28)

**SIK Worksheets v.1.0 Name:**

**Output Date:**

All of the electrical signals that the Arduino works with are either input or output. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

**Output Signals**

A signal exiting an electrical system, in this case a micro-controller.

Output to the Arduino pins is always Digital, however there are two different types of Digital Output; regular Digital Output and Pulse Width Modulation Output (PWM). Output is only possible with Digital pins # 0 - # 13. The Digital pins are preset as Output pins, so unless the pin was used as an Input in the same sketch, there is no reason to use the pinMode command to set the pin as an Output. Should a situation arise where it is necessary to reset a Digital pin to Output from Input use the command:

*pinMode(pinNumber, OUTPUT);*

where pinNumber is the Digital pin number set as Output. To send a Digital Output signal use the command:

*digitalWrite(pinNumber, value);*

where pinNumber is the Digital pin that is outputting the signal and value is the signal. When outputting a Digital signal value can be either HIGH (On) or LOW (Off).

Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. This means you can Output the Digital equivalent of an Analog signal using these pins. To Output a PWM signal use the command:

*analogWrite(pinNumber, value);*

where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

Output can be sent to many different devices, but it is up to the user to figure out which kind of Output signal is needed, hook up the hardware and then type the correct code to properly use these signals.

**Things to remember about Output:**

* Output is always Digital
* There are two kinds of Output: regular Digital or PWM (Pulse Width Modulation)
* To send an Output signal use *analogWrite(pinNumber, value);* (for analog) or *digitalWrite(pinNumber, value);* (for digital)
* Output pin mode is set using the pinMode command: *pinMode(pinNumber, OUTPUT);*
* Regular Digital Output is always either HIGH or LOW
* PWM Output varies from 0 to 255

**Examples of Output:**

Light Emitted Diodes (LED's), Piezoelectric Speakers, Servo Motors

**SIK Worksheets v.1.0 Name:**

**Input Date:**

All of the electrical signals that the Arduino works with are either input or output. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

**Input Signals**

A signal entering an electrical system, in this case a micro-controller. Input to the Arduino pins can come in one of two forms; Analog Input or Digital Input.

Analog Input enters your Arduino through the Analog In pins # 0 - # 5. These signals originate from analog sensors and interface devices. These analog sensors and devices use voltage levels to communicate their information instead of a simple yes (HIGH) or no (LOW). For this reason you cannot use a digital pin as an input pin for these devices. Analog Input pins are used only for receiving Analog signals. It is only possible to read the Analog Input pins so there is no command necessary in the setup( ) function to prepare these pins for input. To read the Analog Input pins use the command:

*analogRead(pinNumber);*

where pinNumber is the Analog Input pin number. This function will return an Analog Input reading between 0 and 1023. A reading of zero corresponds to 0 Volts and a reading of 1023 corresponds to 5 Volts. These voltage values are emitted by the analog sensors and interfaces. If you have an Analog Input that could exceed Vcc + .5V you may change the voltage that 1023 corresponds to by using the Aref pin. This pin sets the maximum voltage parameter your Analog Input pins can read. The Aref pin's preset value is 5V.

Digital Input can enter your Arduino through any of the Digital Pins # 0 - # 13. Digital Input signals are either HIGH (On, 5V) or LOW (Off, 0V). Because the Digital pins can be used either as input or output you will need to prepare the Arduino to use these pins as inputs in your

setup( )function. To do this type the command:

*pinMode(pinNumber, INPUT);*

inside the curly brackets of the setup( ) functionwhere pinNumber is the Digital pin number you wish to declare as an input. You can change the pinMode in the loop( )function if you need to switch a pin back and forth between input and output, but it is usually set in the setup( )function and left untouched in the loop( )function. To read the Digital pins set as inputs use the command:

*digitalRead(pinNumber);*

where pinNumber is the Digital Input pin number.

Input can come from many different devices, but each device's signal will be either Analog or Digital, it is up to the user to figure out which kind of input is needed, hook up the hardware and then type the correct code to properly use these signals.

**Things to remember about Input:**

* Input is either Analog or Digital, make sure to use the correct pins depending on type.
* To take an Input reading use *analogRead(pinNumber);* (for analog)
* Or *digitalRead(pinNumber);* (for digital)
* Digital Input needs a pinMode command such as *pinMode(pinNumber, INPUT);*
* Analog Input varies from 0 to 1023
* Digital Input is always either HIGH or LOW

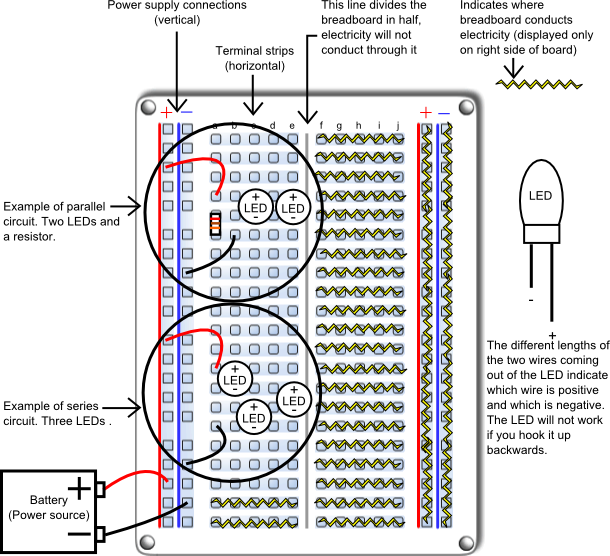
**Examples of Input:**

Push Buttons, Potentiometers, Photoresistors, Flex Sensors

**SIK Worksheets v.1.0 Name:**

**Breadboard**  **Date:**

One of the most important tools for electrical prototyping and invention is the **breadboard**. It's not a piece of bread that you stick electronics into, it's a piece of plastic with holes to place wires into and copper connecting the holes so electricity can get to all the pieces you are working with. But not all the holes are connected! Below is a diagram and explanation of how a **breadboard** works as well as examples of parallel and series circuits. Not sure what parallel and series circuits are? Don't worry! The important thing is learning how to use the **breadboard** so you can play around with some electronics.



The right side of this **breadboard** shows you which holes are connected and allow electricity to flow between them without anything else connecting them. This is made possible by strips of copper on the underside of the board. The **power** supply connections have a + and – indicating how to hook up your power source. The connections for the **power** supply run up and down. The **terminal** strips are labeled “a” through “j”, these connections run across the board, but are broken down the middle. This cuts the connection across the entire **terminal** area in half, giving you two unconnected sections to work with.

**SIK Worksheets v.1.0 Name:**

**Circuit 1, How the Circuits Work Date:**

**Circuit 1**

|  |  |  |
| --- | --- | --- |
| **Explanation:**  This circuit takes electricity from digital Pin # 9 on the Arduino. Pin # 9 on the Arduino has Pulse Width Modulation capability allowing the user to change the brightness of the LED when using analogWrite. The LED is connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire). The resistor dissipates current so the LED does not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground. |  | **Schematic:** |
| **Components:**  Arduino Digital Pin # 9: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.  LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Also lights up!  330 Ohm Resistor: A resistorresists the current flowing through the circuit. In this circuit the resistor reduces the current so the LED does not burn out.  Gnd: Ground |  | **Code:**  *int ledPin = 3;*  *void setup() {   pinMode(ledPin, OUTPUT);*  *}    void loop() {  digitalWrite(ledPin, HIGH); //LED on  delay(1000); // wait second  digitalWrite(ledPin, LOW); //LED off  delay(1000); // wait second }*  **or** for PWM output loop could read :  *int ledPin = 3;*  *void setup() {   pinMode(ledPin, OUTPUT);*  *}*  *void loop() {  analogWrite(ledPin, 255); // LED on  delay(1000); // wait second  analogWrite(ledPin, 0); // LED off  delay(1000); // wait second }* |

This first circuit is the simplest form of output in the kit. You can use the LED to teach both analog and digital output before moving on to more exciting outputs. There is an LED built into your Arduino board which corresponds to Digital Pin # 13.

**SIK Worksheets v.1.0 Name:**

**Circuit 8, How the Circuits Work Date:**

**Circuit 8**

|  |  |  |
| --- | --- | --- |
| **Explanation:**  This circuit is actually two different circuits. One circuit for the potentiometer and another for the LED. See How the Circuits Work, Circuit 1 for an explanation of the LED circuit. The potentiometer circuit gets electricity from the 5V on the Arduino. The electricity passes through the potentiometer and sends a signal to Analog Pin # 0 on the Arduino. The value of this signal changes depending on the setting of the dial on the potentiometer. This analog reading is then used in the code you load onto the Arduino and effects the power signal in the LED circuit. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground. |  | **Schematic:** |
| **Components:**  Arduino Digital Pin # 13: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.  Arduino Analog Pin # 0: Analog input to Arduino board.  330 Ohm Resistor: A resistorresists the current flowing through the circuit. In the LED circuit it reduces the current so the LED in the circuit does not burn out.  LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction.  Potentiometer: A voltage divider which outputs an analog value.  +5V: Five Volt power source.  Gnd: Ground |  | **Code:**  *int sensorPin = 0;*  *int ledPin = 13;*  *int sensorValue = 0;*  *void setup() {*  *pinMode(ledPin, OUTPUT);*  *}*  *void loop() {*  *//this line assigns whatever the //Analog Pin 0 reads to sensorValue*  *sensorValue = analogRead(sensorPin);*  *digitalWrite(ledPin, HIGH);*  *delay(sensorValue);*  *digitalWrite(ledPin, LOW);*  *delay(sensorValue);*  *}* |

This is another example of input, only this time it is Analog. Circuits 7 and 8 in the S.I.K. introduces you to the two kinds of input your board can receive: Digital and Analog. Not sure what a voltage divider is? Check the Voltage Divider page towards the back of this section.

**SIK Worksheets v.1.0 Name:**

**Circuit 7, How the Circuits Work Date:**

**Circuit 7**

|  |  |  |
| --- | --- | --- |
| **Explanation:**  This circuit is actually two different circuits. One circuit for the buttons and another for the LED. See ‘How the Circuits Work’, Circuit 1 for an explanation of the LED circuit. The button circuit gets electricity from the 5V on the Arduino. The electricity passes through a pull up resistor, causing the input on Arduino Pins # 2 and # 3 to read HIGH when the buttons are not being pushed. When a button is pushed it allows the current to flow to ground, causing a LOW reading on the input pin connected to it. This LOW reading is then used in the code you load onto the Arduino and effects the power signal in the LED circuit. |  | **Schematic:** |
| **Components:**  Arduino Digital Pin # 13: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.  Arduino Digital Pin # 2 and # 3: Digital input to Arduino board.  330 & 10K Ohm Resistors: Resistorsresist the current flowing through the circuit. In the LED circuit the 330 ohm resistor reduces the current so the LED in the circuit does not burn out. In the button circuits the 10K's ensure that the buttons will read HIGH when they are not pressed.  LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Lights up!  Button: A press button which is open (or disconnected) when not in use and closed (or connected) when pressed. This allows you to complete a circuit when you press a button.  +5V: Five Volt power source.  Gnd: Ground |  | **Code:**  *const int buttonPin = 2; const int ledPin = 13;*  *int buttonState = 0;*  *void setup() {  pinMode(ledPin, OUTPUT);*  *//this line below declares the button pin as input  pinMode(buttonPin, INPUT);  }*  *void loop(){  //this line assigns whatever the Digital Pin 2 reads to buttonState  buttonState = digitalRead(buttonPin);*  *if (buttonState == HIGH) {  digitalWrite(ledPin, HIGH);  }  else {  digitalWrite(ledPin, LOW);  } }* |

**Resistance**

Resistance is an important concept when you are creating circuits. Resistance is the difficulty a current encounters when it passes through a component. Everything that electricity passes through provides some measure of resistance, wires, motors, sensors, even the human body!

Measuring voltage, current and resistance are all done in different ways. To measure resistance you disconnect (turn off) your circuit and place both multimeter leads on either side of the portion of the circuit you wish to measure. For example: for measuring just a component you would place your leads on the power and ground leads of the component. To measure the resistance of multiple components you leave them connected and place the positive (red) multimeter lead closer to the disconnected power source and the negative (black) multimeter lead closer to the ground. Sometimes you will want to measure the resistance of input and output leads, but more often you will find yourself measuring resistance along the power to ground circuit. It is important to know how much resistance is present in components and circuits for many reasons. Too much resistance and the current will never travel through the whole circuit, too little and the current may fry some of your components! But most importantly you can use resistance to choose the path the current takes through your circuit.

|  |  |
| --- | --- |
| Hook up the circuit to the right using red LEDs. (Don't hook up the power yet.)  Measure the resistance of each of the possible paths the current can take from power (5v) to ground. There are three possible paths. You will have to measure each component separately and then add the resistance up for the total. You will can add the component's resistance together because the components are in series, if they were parallel it would require more math. Record the total resistance for each circuit below. (Hint: you won't be able to measure the LED)  Circuit 1: \_\_\_\_\_\_\_Ω Circuit 2: \_\_\_\_\_\_\_\_\_Ω Circuit 3: \_\_\_\_\_\_\_\_\_\_\_Ω  Now connect the power and, one at a time, press the two buttons. Which circuit makes the LED the dimmest? Circuit # \_\_\_\_\_\_  If you press both buttons which path does the current take? Circuit # \_\_\_\_\_\_  If the voltage is staying at 5v in this circuit no matter which paths are closed, there is a way to calculate the current given the resistance. Write the name of the law and the equation that solves for resistance below. Label all variables. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Now measure the resistance of a potentiometer when it is dialed all the way up and down. Record the highest and lowest values you get.  Highest:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ω  Lowest:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ω  Redraw the schematic above (use the back of the worksheet if necessary) but use a potentiometer to control the LED brightness instead of the buttons and various resistors. Remember that you must have at least 330Ω of total resistance, otherwise you'll burn out your LED! |  |

Since a circuit or component does not need a current running through it in order to measure the resistance you can take your multimeter and measure the resistance of anything you can think of. Wander around and measure the resistance of various objects. Start with a penny. Record the most interesting things that have resistance and the value of their resistance below. List at least three.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SIK Worksheets v.1.0 Name:**

**Voltage Drop Date:**

Voltage drop is an important concept when you are creating circuits. Voltage drop is the amount that the voltage drops when it passes through a component. The following exercises will show how to measure voltage drop in real life. This is essential when you are fixing your remote control car, electric guitar or even a cell phone.

Measuring voltage, current and resistance are all done in different ways. To measure voltage you connect your positive (red) multimeter lead to the side of the circuit that closer to your power source and the negative (black) multimeter lead to the side of the circuit that is closer to the ground. It is important to know how much voltage is going through a circuit for many reasons. The most important reasons being that too much voltage can damage your components and too little voltage may not allow electricity to flow all the way through to ground.

|  |  |
| --- | --- |
| Hook up the circuit to the right using red LEDs.  Close the circuit so only one LED is grounded with the 300Ω resistor. Insert the end of the resistor not plugged into the ground into a hole on the same row as the first LED's negative lead. The other LEDs don't light up, why is this?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Measure the voltage drop across just the LED and record. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_v  Measure the voltage drop across the LED and the resistor. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_v  Close the circuit so two LEDs light up.  Voltage drop across one LED = \_\_\_\_\_v Voltage drop across two LEDs = \_\_\_\_\_v  Measure the voltage drop across the whole circuit and record. \_\_\_\_\_\_\_\_\_\_\_\_\_v  Close the circuit so three LEDs light up.  Voltage drop across one LED = \_\_\_\_\_v Voltage drop across two LEDs = \_\_\_\_\_v  Voltage drop for three LEDs = \_\_\_\_\_v Voltage drop for whole circuit = \_\_\_\_\_v  What happened to the LEDs with the last question?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Now hook up the same circuit to the 3.3V power source without the resistor.  Why do you think you don't need the resistor?  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Measure the voltage drop across all the LEDs and record. \_\_\_\_\_\_\_\_\_\_\_\_v  Close the circuit so only two LEDs light up.  Voltage drop across one LED = \_\_\_\_\_v Voltage drop across two LEDs = \_\_\_\_\_v  Hook up the circuit above to the 5V power source but use the 3.3v as ground.  Wait a second! You can't use a power source as a ground! Or can you?  What is the voltage available and how many LEDs can you light up with it?  Voltage available = \_\_\_\_\_\_\_\_v # of LEDs you can light up = \_\_\_\_\_\_\_\_\_\_ |  |

Many people think of Gnd as the ONLY place to connect a 'negative' pin, but all you need is a voltage drop from the beginning of a circuit to the end. This difference in voltage is what draws the current in the correct direction.

**SIK Worksheets v.1.0 Name:**

**Transistors Date:**

**What is a transistor?**

Transistors are semiconductors used to amplify an electrical signal or switch an electrical signal on and off.

**Why is a transistor useful?**

Often you will need more power to run a component than your Arduino can provide. A transistor allows you to control the higher power signal by breaking or closing a circuit to ground. Combining this higher power allows you to amplify the electrical signal in your circuit.

**What is in a transistor?**

A transistor circuit has four parts; a signal power source (connects to transistor base), an affected power source (connects to transistor collector), voltage out (connects to transistor collector), and ground (connected to transistor emitter).

**How do you put together a transistor?**

It's really pretty easy. Here is a schematic and explanation detailing how:

|  |  |
| --- | --- |
|  | The transistor voltage in signal is the signal that is used to control the transistor's base.  Signal in is the power source for the signal out which is controlled by the transistor's action.  Signal out is the output of the signal originating from signal in, it is controlled by the collector.  The amount of electrical current allowed through the transistor and out of the emitter to ground is what closes the entire circuit, allowing electrical current to flow through signal out. |

**Ok, how is this transistor information used?**

It depends on what you want to do with it really. There are two different purposes outlined above for the transistor, we will go over both.

If you wish to use the transistor as a switch the signal in and voltage in signal are connected to the same power source with a switch between them. When the switch is moved to the closed position an electrical signal is provided to the transistor base creating forward bias and allowing the electrical signal to travel from the signal in to the transistor's collector to the emitter and finally to ground. When the circuit is completed in this way the signal out is provided with an electrical current from signal in.

The signal amplifier use of the transistor works the same way only Signal In and Voltage In are not connected. This disconnection allows the user to send differing values to the base of the transistor. The closer the voltage in value is to the saturation voltage of the transistor the more electrical current that is allowed through the emitter to ground. By changing the amount of electrical current allowed through to ground you change the signal value of signal out. For examples of transistor uses see S.I.K. circuits # 3 and # 11.

**SIK Worksheets v.1.0 Name:**

**Voltage Dividers Date:**

**What is a voltage divider?**

Voltage dividers are a way to produce a voltage that is a fraction of the original voltage.

**Why is a voltage divider useful?**

One of the ways a voltage divider is useful is when you want to take readings from a circuit that has a voltage beyond the limits of your input pins. By creating a voltage divider you can be sure that you are getting an accurate reading of a voltage from a circuit. Voltage dividers are also used to provide an analog Reference signal.

**What is in a voltage divider?**

A voltage divider has three parts; 2 resistors and a way to read voltage between the 2 resistors.

**How do you put together a voltage divider?**

It's really pretty easy. Here is a schematic and explanation detailing how:

|  |  |
| --- | --- |
|  | Often resistor # 1 is a resistor with a value that changes, possibly a sensor or a potentiometer.  Resistor # 2 has whatever value is needed to create the ratio the user decides is acceptable for the voltage divider output.  The Voltage In and Ground portions are just there to establish which way the electrical current is heading, there can be any number of circuits before and after the voltage divider.  Here is the equation that represents how a voltage divider works:  If both resistors have the same value then Voltage Out is equal to ½ Voltage In. |

**Ok, how is this voltage divider information used?**

It depends on what you want to do with it really. There are two different purposes.

If you wish to use the voltage divider as a sensor reading device first you need to know the maximum voltage allowed by the analog inputs you are using to read the signal. On an Arduino this is 5V. So, already we know the maximum value we need for Vout. The Vin is simply the amount of voltage already present on the circuit before it reaches the first resistor. You should be able to find the maximum voltage your sensor outputs by looking on the Datasheet, this is the maximum amount of voltage your sensor will let through given the voltage in of your circuit. Now we have exactly one variable left, the value of the second resistor. Solve for R2 and you will have all the components of your voltage divider figured out! We solve for R1's highest value because a smaller resistor will simply give us a smaller signal which will be readable by our analog inputs.

Powering an analog Reference is exactly the same as reading a sensor except you have to calculate for the Voltage Out value you want to use as the analog Reference.

Given three of these values you can always solve for the missing value using a little algebra, making it pretty easy to put together your own voltage divider. The S.I.K. has many voltage dividers in the example circuits. These include: Circuits # 7, 8, 9, 13 and 14.

to use the input capabilities of the Arduino. Notice the difference in *setup( )*. You are still using a Digital Pin but you are using it as input rather than output.

Fire sensor:



**Product Description**

We have gained a remarkable position in the market that is involved in offering Fire Sensor.

indicator light: green one for switch, red one for power.Built in a potentiometer for sensitivity control.On-board signal output indication, output effective signal is high, at the same time the indicator light up, output signal can directly connect with microcontroller IO.Can detect fire or wavelength in 760 ~ 1100 nm nano within the scope of the light source.Detection angle about 60 degrees, the flame spectrum especially sensitive.The flame of the most sensitive sensors flame, the regular light is also a response, generally used for fire alarm purposes.

**2. INTRODUCTION TO GSM**

A GSM modem is a specialized type of modem which accepts a SIM card, and operates over a subscription to a mobile operator, just like a mobile phone. From the mobile operator perspective, a GSM modem looks just like a mobile phone. When a GSM modem is connected to a computer, this allows the computer to use the GSM modem to communicate over the mobile network. While these GSM modems are most frequently used to provide mobile internet connectivity, many of them can also be used for sending and receiving SMS and MMS messages. A GSM modem can be a dedicated modem device with a serial, USB or Bluetooth connection, or it can be a mobile phone that provides GSM modem capabilities.

A GSM modem exposes an interface that allows applications such as Now SMS to send and receive messages over the modem interface. The mobile operator charges for this message sending and receiving as if it was performed directly on a mobile phone. To perform these tasks, a GSM modem must support an “extended AT command set” for sending/receiving SMS messages.



Fig2.0:GSM MODEM

GSM modems can be a quick and efficient way to get started with SMS, because a special subscription to an SMS service provider is not required. In most parts of the world, GSM modems are a cost effective solution for receiving SMS messages, because the sender is paying for the message delivery.

### 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.

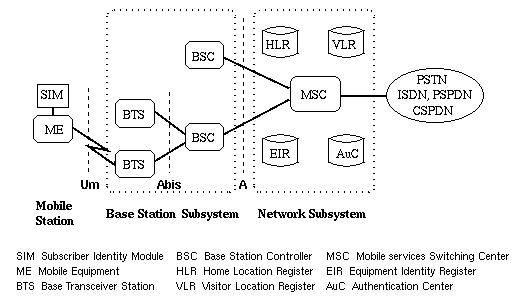


Fig 2.1 General architecture of a GSM network

**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.

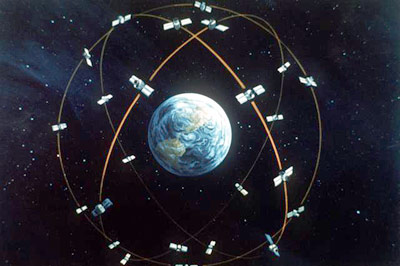
****

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**

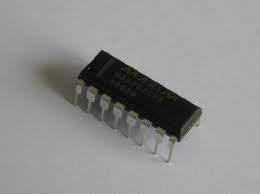
**07 08 09 10 11 12 13 14 4 5 6**

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

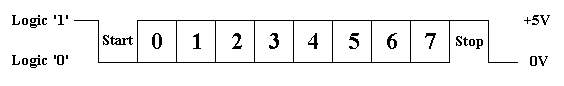
The LCD plays a major role in the entire operation as it has the ability to display the certain data that the user has entitled.

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.

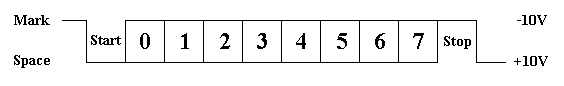
The data sent using this method, is said to be framed. That is the data is framed between a Start and Stop Bit.

**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.

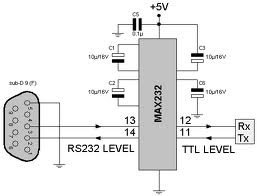
**Serial communication**

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.

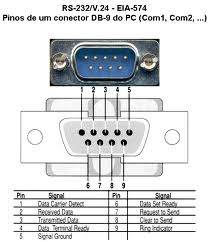
Serial data communication uses two methods, asynchronous and synchronous. The synchronous method transfers data at a time while the asynchronous transfers a single byte at a time. There are some special IC chips made by many manufacturers for data communications. These chips are commonly referred to 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.

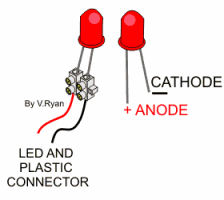


IBM PC/ compatible computers based on x86(8086, 80286, 386, 486 and Pentium) microprocessors normally have two COM ports. Both COM ports have RS232 type connectors. Many PCs use one each of the DB-25 and DB-9 RS232 connectors. The COM ports are designated as COM1 and COM2. We can connect the serial port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

## RELAYS SPDT

[](http://electronics.howstuffworks.com/electronics-parts-pictures.htm)

## Overview OF Relays

A relay is an electrically operated switch used to isolate one electrical circuit from another. In its simplest form, a relay consists of a coil used as an electromagnet to open and close switch contacts. Since the two circuits are isolated from one another, a lower voltage circuit can be used to trip a relay, which will control a separate circuit that requires a higher voltage or amperage. Relays can be found in early telephone exchange equipment, in industrial control circuits, in car audio systems, in automobiles, on water pumps, in high-power audio amplifiers and as protection devices.

## Relay Switch Contacts

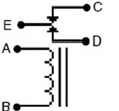
The switch contacts on a relay can be "normally open" (NO) or "normally closed" (NC)--that is, when the coil is at rest and not energized (no current flowing through it), the switch contacts are given the designation of being NO or NC. In an open circuit, no current flows, such as a wall light switch in your home in a position that the light is off. In a closed circuit, metal switch contacts touch each other to complete a circuit, and current flows, similar to turning a light switch to the "on" position. In the accompanying schematic diagram, points A and B connect to the coil. Points C and D connect to the



switch. When you apply a voltage across the coil at points A and B, you create an electromagnetic field, which attracts a lever in the switch, causing it to make or break contact in the circuit at points C and D (depending if the design is NO or NC). The switch contacts remain in this state until you remove the voltage to the coil. Relays come in different switch configurations. The switches may have more than one "pole," or switch contact. The diagram shows a "single pole single throw" configuration, referred to as SPST. This is similar to a wall light switch in your home. With a single "throw" of the switch, you close the circuit.

## The Single Pole Double Throw Relay

A single pole double throw (SPDT) relay configuration switches one common pole to two other poles, flipping between them. As shown in the schematic diagram, the common point E completes a circuit with C when the relay coil is at rest, that is, no voltage is applied to it.

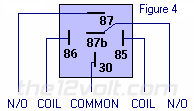


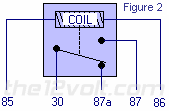
This circuit is "closed." A gap between the contacts of point E and D creates an "open" circuit. When you apply power to the coil, a metal level is pulled down, closing the circuit between points E and D and opening the circuit between E and C. A single pole double throw relay can be used to alternate which circuit a voltage or signal will be sent to.

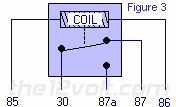
**SPDT Relay**:

(**S**ingle **P**ole **D**ouble **T**hrow **Relay**) an electromagnetic switch, consist of a coil (terminals 85 & 86), 1 common terminal (30), 1 normally closed terminal (87a), and one normally open terminal (87) (Figure 1).

When the coil of an SPDT relay (Figure 1) is at rest (not energized), the common terminal (30) and the normally closed terminal (87a) have continuity. When the coil is energized, the common terminal (30) and the normally open terminal (87) have continuity.   
  
The diagram below center (Figure 2) shows an SPDT relay at rest, with the coil not energized. The diagram below right (Figure 3) shows the relay with the coil energized. As you can see, the coil is an electromagnet that causes the arm that is always connected to the common (30) to pivot when energized whereby contact is broken from the normally closed terminal (87a) and made with the normally open terminal (87).   
  
When energizing the coil of a relay, polarity of the coil does not matter unless there is a diode across the coil. If a diode is not present, you may attach positive voltage to either terminal of the coil and negative voltage to the other, otherwise you must connect positive to the side of the coil that the cathode side (side with stripe) of the diode is connected and negative to side of the coil that the anode side of the diode is connected.







**Why do I want to use a relay and do I really need to?**

Anytime you want to switch a device which draws more current than is provided by an output of a switch or component you'll need to use a relay. The coil of an **SPDT** or an **SPST** relay that we most commonly use draws very little current (less than 200 milliamps) and the amount of current that you can pass through a relay's common, normally closed, and normally open contacts will handle up to 30 or 40 amps. This allows you to switch devices such as headlights, parking lights, horns, etc., with low amperage outputs such as those found on keyless entry and alarm systems, and other components. In some cases you may need to switch multiple things at the same time using one output. A single output connected to multiple relays will allow you to open continuity and/or close continuity simultaneously on multiple wires.   
  
There are far too many applications to list that require the use of a relay, but we do show many of the most popular applications in the pages that follow and many more in our Relay Diagrams - Quick Reference application. If you are still unclear about what a relay does or if you should use one after you browse through the rest of this section, please post a question in the12volt's install bay. (We recommend Tyco (formerly Bosch) or Potter & Brumfield relays for all of the SPDT and SPST relay applications shown on this site.)



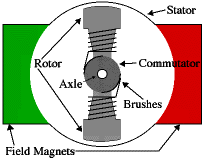
**DC MOTOR**

****

DC motors are configured in many types and sizes, including brush less, servo, and gear motor types. A motor consists of a rotor and a permanent magnetic field stator. The magnetic field is maintained using either permanent magnets or electromagnetic windings. DC motors are most commonly used in variable speed and torque.  
 Motion and controls cover a wide range of components that in some way are used to generate and/or control motion. Areas within this category include bearings and bushings, clutches and brakes, controls and drives, drive components, encoders and resolves, Integrated motion control, limit switches, linear actuators, linear and rotary motion components, linear position sensing, [motors](http://dc-motors.globalspec.com/Industrial-Directory/motors) (both AC and DC motors), orientation position sensing, pneumatics and pneumatic components, positioning stages, slides and guides, power transmission (mechanical), seals, slip rings, solenoids, springs.   
  
 Motors are the devices that provide the actual speed and torque in a drive system.  This family includes AC motor types (single and multiphase motors, universal, servo motors, induction, synchronous, and gear motor) and DC motors (brush less, servo motor, and gear motor) as well as linear, stepper and air motors, and motor contactors and starters.

In any electric motor, operation is based on simple electromagnetism. A current-carrying conductor generates a magnetic field; when this is then placed in an external magnetic field, it will experience a force proportional to the current in the conductor, and to the strength of the external magnetic field. As you are well aware of from playing with magnets as a kid, opposite (North and South) polarities attract, while like polarities (North and North, South and South) repel. The internal configuration of a DC motor is designed to harness the magnetic interaction between a current-carrying conductor and an external magnetic field to generate rotational motion.

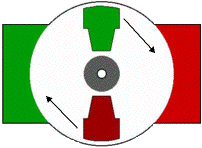
Let's start by looking at a simple 2-pole DC electric motor (here red represents a magnet or winding with a "North" polarization, while green represents a magnet or winding with a "South" polarization).



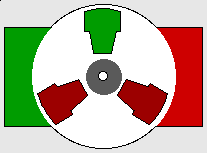
Every DC motor has six basic parts -- axle, rotor (a.k.a., armature), stator, commutator, field magnet(s), and brushes. In most common DC motors (and all that Beamers will see), the external magnetic field is produced by high-strength permanent magnets1. The stator is the stationary part of the motor -- this includes the motor casing, as well as two or more permanent magnet pole pieces. The rotor (together with the axle and attached commutator) rotates with respect to the stator. The rotor consists of windings (generally on a core), the windings being electrically connected to the commutator. The above diagram shows a common motor layout -- with the rotor inside the stator (field) magnets.

The geometry of the brushes, commutator contacts, and rotor windings are such that when power is applied, the polarities of the energized winding and the stator magnet(s) are misaligned, and the rotor will rotate until it is almost aligned with the stator's field magnets. As the rotor reaches alignment, the brushes move to the next commutator contacts, and energize the next winding. Given our example two-pole motor, the rotation reverses the direction of current through the rotor winding, leading to a "flip" of the rotor's magnetic field, and driving it to continue rotating.

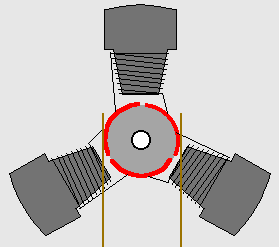
In real life, though, DC motors will always have more than two poles (three is a very common number). In particular, this avoids "dead spots" in the commutator. You can imagine how with our example two-pole motor, if the rotor is exactly at the middle of its rotation (perfectly aligned with the field magnets), it will get "stuck" there. Meanwhile, with a two-pole motor, there is a moment where the commutator shorts out the power supply (i.e., both brushes touch both commutator contacts simultaneously). This would be bad for the power supply, waste energy, and damage motor components as well. Yet another disadvantage of such a simple motor is that it would exhibit a high amount of torque” ripple" (the amount of torque it could produce is cyclic with the position of the rotor).



So since most small DC motors are of a three-pole design, let's tinker with the workings of one via an interactive animation (JavaScript required):



You'll notice a few things from this -- namely, one pole is fully energized at a time (but two others are "partially" energized). As each brush transitions from one commutator contact to the next, one coil's field will rapidly collapse, as the next coil's field will rapidly charge up (this occurs within a few microsecond). We'll see more about the effects of this later, but in the meantime you can see that this is a direct result of the coil windings' series wiring:



There's probably no better way to see how an average dc motor is put together, than by just opening one up. Unfortunately this is tedious work, as well as requiring the destruction of a perfectly good motor.

**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.

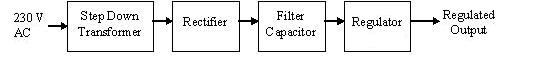
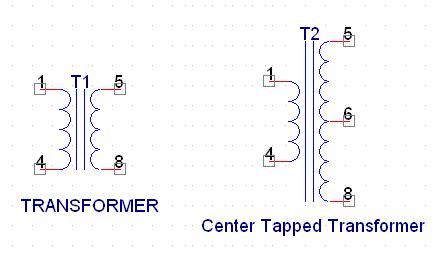


Figure 1 shows the basic block diagram of a fixed regulated power supply. Let us go through each block.

**TRANSFORMER**



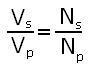
A transformer consists of two coils also called as “WINDINGS” namely PRIMARY & SECONDARY.

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.

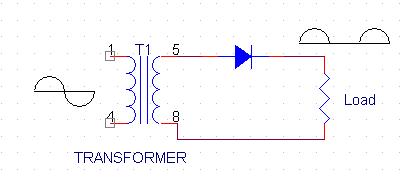


**Rectifier**

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.

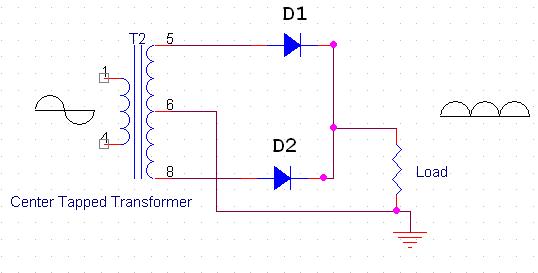
Rectifier can be classified as follows:

**1)      Half Wave rectifier.**



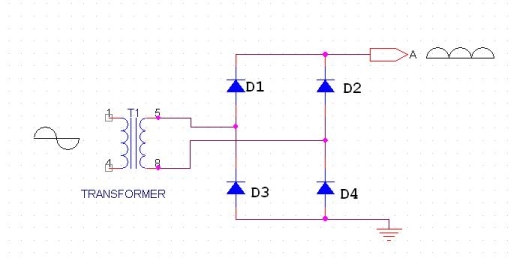
This is the simplest type of rectifier as you can see in the diagram a half wave rectifier consists of only one diode. When an AC signal is applied to it during the positive half cycle the diode is forward biased & current flows through it. But during the negative half cycle diode is reverse biased & 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.**



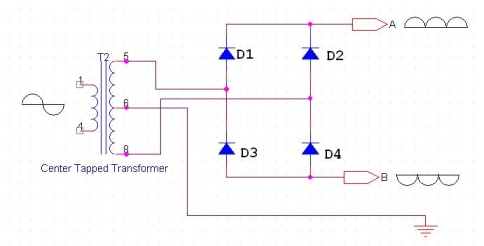
Half wave rectifier is quite simple but it is very inefficient, for greater efficiency we would like to use both the half cycles of the AC signal. This can be achieved by using a center tapped transformer i.e. we would have to double the size of secondary winding & provide connection to the center. So 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.

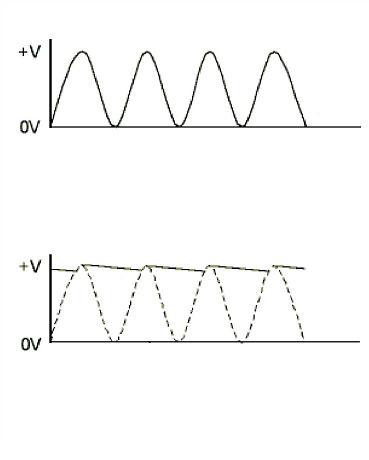


If we use a center tapped transformer for a bridge rectifier we can get both positive & negative half cycles which can thus be used for generating fixed positive & fixed negative voltages.

**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

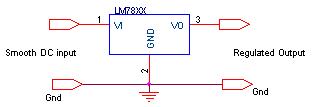
1)      Linear Voltage Regulator

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

2)      Switching Regulators.

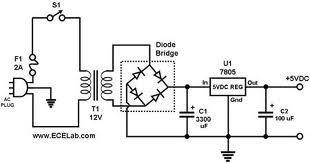
      They regulate the output voltage by switching the Current ON/OFF very rapidly. Since their output is either ON or OFF it dissipates very low power thus achieving higher efficiency as compared to linear voltage regulators. But they are more complex & generate high noise due to their switching action. 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 |

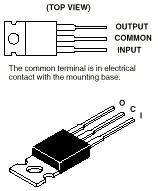
A variable regulated power supply, also called a variable bench power supply, is one where you can continuously adjust the output voltage to your requirements. Varying the output of the power supply is the recommended way to test a project after having double checked parts placement against circuit drawings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





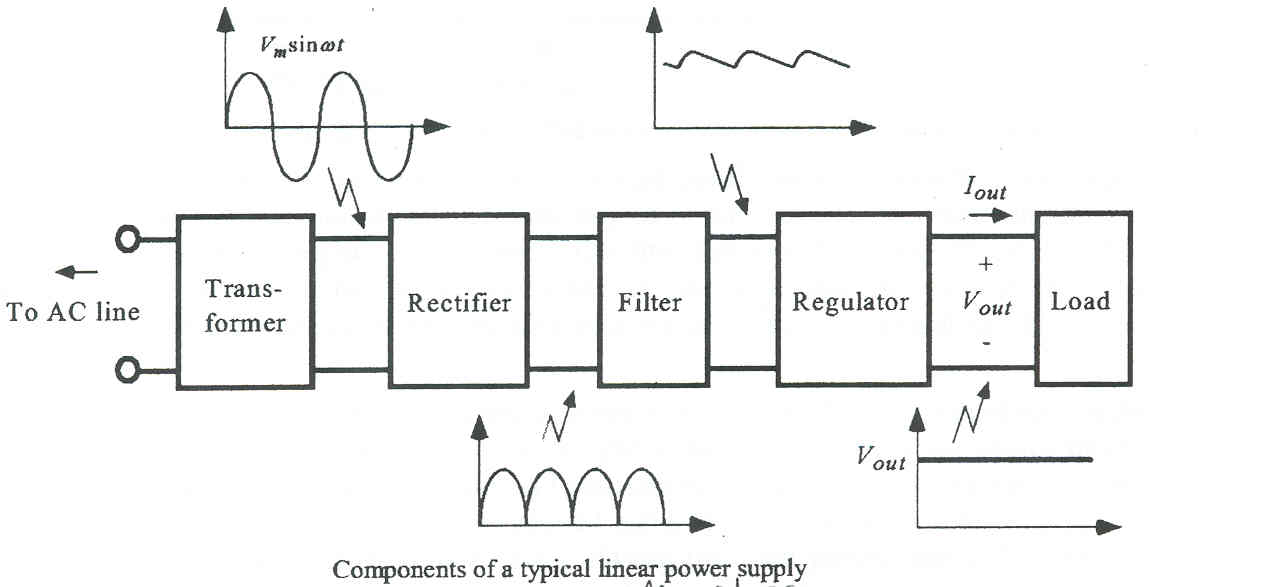
We are using General purpose PCB. To mount or place the components on the PCB. After that

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.

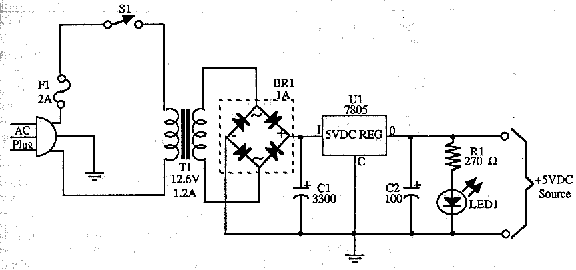
**7.2 CIRCUIT FEATURES:**

* 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

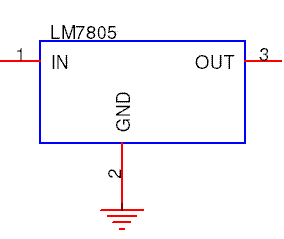
**7.3 BLOCK DIAGRAM:**



**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 CODE:

#include<LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 10, 11, 12, 13);

int BUZZ = A3; //Connect BUZZ To Pin #7 ////////buzzer

int RLED = A0; //Connect LED To Pin #A0 ///////led

int GLED = A1; //Connect LED To Pin #A1 ///////led

int relay = A2; //Connect relay To Pin #6 ///////relay

char data = 0;

void setup()

{

lcd.begin(16, 2);

Serial.begin(9600);

pinMode(BUZZ, OUTPUT);

pinMode(RLED, OUTPUT);

pinMode(GLED, OUTPUT);

pinMode(relay, OUTPUT);

digitalWrite(BUZZ,LOW);

digitalWrite(GLED,LOW);

digitalWrite(relay,LOW);

digitalWrite(RLED,LOW);

lcd.clear();

lcd.setCursor(0,0);lcd.print("Fire Detection");

lcd.setCursor(0,1);lcd.print("and Alerting");

delay(5000);lcd.clear();

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

lcd.setCursor(0,1);lcd.print("Arduino- GSM");

delay(5000);lcd.clear();

digitalWrite(GLED,LOW);delay(100);

digitalWrite(BUZZ,LOW);delay(100);

digitalWrite(GLED,HIGH);

lcd.setCursor(0,0);

lcd.print(" NO FIRE ");

lcd.setCursor(0,1);

lcd.print(" ALAERT ");

digitalWrite(BUZZ,LOW);delay(100);

digitalWrite(RLED,LOW);delay(100);

digitalWrite(GLED,HIGH);delay(100);

digitalWrite(relay,LOW);delay(100);

}

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

void loop()

{

st:

if (Serial.available() > 0)

{

data = Serial.read();

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

if(data == 'S')

{

lcd.setCursor(0,0);

lcd.print(" NO FIRE ");

lcd.setCursor(0,1);

lcd.print(" ALAERT ");

digitalWrite(BUZZ,LOW);delay(100);

digitalWrite(RLED,LOW);delay(100);

digitalWrite(GLED,HIGH);delay(100);

digitalWrite(relay,LOW);delay(100);

}

if(data == 'F')

{

lcd.setCursor(0,0);

lcd.print(" Fire ");

lcd.setCursor(0,1);

lcd.print(" Detected ");

digitalWrite(BUZZ,HIGH);delay(100);

digitalWrite(RLED,HIGH);delay(100);

digitalWrite(GLED,LOW);delay(100);

digitalWrite(relay,HIGH);delay(100);

lcd.clear();lcd.print("Sending SMS ");

delay(500);

Send();

delay(500);

digitalWrite(BUZZ,LOW);digitalWrite(RLED,HIGH);

digitalWrite(relay,LOW);delay(100);

lcd.clear();lcd.print("CALLING------1 ");

Serial.print("ATD+91");

Serial.print("7003664172");

Serial.print(";\r\n");delay(12000);

Serial.println("ATH");

delay(2000);lcd.clear();

lcd.clear();lcd.print("CALLING------2 ");

Serial.print("ATD+91");

Serial.print("9007392003");

Serial.print(";\r\n");delay(12000);

Serial.println("ATH");

delay(2000);lcd.clear();

lcd.setCursor(0,0);

lcd.print(" NO FIRE ");

lcd.setCursor(0,1);

lcd.print(" ALAERT ");

digitalWrite(BUZZ,LOW);delay(100);

digitalWrite(RLED,LOW);delay(100);

digitalWrite(GLED,HIGH);delay(100);

digitalWrite(relay,LOW);delay(100);

goto st;

}

}

}

void init\_sms1()

{Serial.println("AT+CMGF=1");delay(400);Serial.println("AT+CMGS=\"7003664172\""); delay(500);}

void init\_sms2()

{Serial.println("AT+CMGF=1");delay(400);Serial.println("AT+CMGS=\"9007392003\""); delay(500);}

void Send\_sms()

{

Serial.println("Fire Detected");delay(1000);

Serial.println("Plz Rescue ");delay(1000);

Serial.println("https://www.google.com/maps/place/22.583419,88.374658");delay(1000);

Serial.write(26);delay(3000);lcd.clear();

lcd.print("Message Sent ");

delay(1000); lcd.clear();

Serial.print("AT\r\n");delay(500);

Serial.print("AT\r\n");delay(500);

Serial.println("AT+CMGF=1");delay(500);

lcd.clear();

}

void Send()

{

init\_sms1();delay(1000);lcd.clear();lcd.print("Sending SMS ");Send\_sms();

init\_sms2();delay(1000);lcd.clear();lcd.print("Sending SMS ");Send\_sms();

}