**SMART TRAFFIC SYSTEM FOR EMERGENCY VEHICLES**

**ABSTRACT** In today’s world, traffic jams during rush hours is one of the major concerns. During rush hours, emergency vehicles like Ambulances, fire truck get stuck in jams. Due to this, these emergency vehicles are not able to reach their destinations on time, resulting into a loss of human lives. We are willing to develop a system which is used to provide clearance to any emergency vehicle when it struck in traffic jam. Here we clear the path of the emergency vehicle hence it can reach the destination in time. Emergency vehicle stuck in a heavy traffic condition it will send an information to the next vehicle through light medium from headlight to indicator of next vehicle. The information received by the next vehicle and it transmit to the another vehicle next to it. The process will continue till the information reaches to the first vehicle and then further information will be transfer to the traffic signal system through a road stud. The traffic signal light will turn on from red to green.

Traffic congestion problem is a phenomenon which contributed huge impact to the transportation system in country. This causes many problems especially when there are emergency cases at traffic light intersections which are always busy with many vehicles. A traffic light controller system is designed in order to solve these problems. This system was designed to be operated when it received signal from emergency vehicles based on radio frequency (RF) transmission and used the microcontroller to change the sequence back to the normal sequence before the emergency mode was triggered. This system will reduce accidents which often happen at the traffic light intersections because of other vehicle had to huddle for given a special route to emergency vehicle. As the result, this project successful analyzing and implementing the wireless communication; the radio frequency (RF) transmission in the traffic light control system for emergency vehicles.

**INTRODUCTION**

One of the many problems that the world faces with increased population and rapid growth in the number of vehicles is traffic congestion. In countries such as India, the rate of road expansion is just one-third the vehicular growth rate. Statistics show that the current annual growth of vehicles is around 11% while the annual road extension remains to be only around 4%. The effects of increased traffic congestion are many. Our project focuses on the severe impact caused by traffic congestion on the emergency vehicle transportation system. In places such as India and Thailand where the road width and length prove to be impossible to create a separate lane for emergency vehicles, it is difficult for ambulances to navigate through the traffic. These systems however require the presence of traffic policemen during the hours of peak traffic. Also, above methods do not cater to the necessities of emergency vehicles such as ambulances with lives at stake.

Traffic control systems are used to control more than two passage paths of vehicles or wherever the pedestrians passage used to cross a road. Also used, when two paths cross each other in the case of a four way lane. The purpose of a traffic control system is to control the vehicle flow through a lane and to prevent accidents or a road blockage. In our country, sequential logic traffic system is used at present in which the control lights used for present traffic system are namely, red, yellow and green for stop, get ready and go respectively in which each light operates on after another for a given period of time. The delay parameters on which the traffic load depends are time, day, season, weather and unpredictable situations such as accidents or constructional activities. The congestion problem can be solved by construction of new roads. The only drawback is that the surroundings also becomes more congested. There exists a need to change the traffic system rather than creating new infrastructure twice

**LITERATURE SURVEY**

Kouvelas et al. [3] implemented an webster method for fixed traffic signal control derivation at isolated junctions and used for real-time operations. It is also proved that this approach is suitable for saturated traffic conditions through simulation results. This method avoids the need for pre-specified fixed signal traffic plan and it gives more accurate results than the method discussed in [1].

The scheme proposed by Zhang et al. [4] applies lagrange coordinates, discrete model and a continuous model and an correlation exists, by which the approximation using the increment M vanishes. This leads to the development of Kerner’s three phase traffic theory which explains the empirical features of traffic breakdown to resolve the congested real traffic patterns.

Bauza et al. [5] presented a Cooperative Traffic Congestion detection (CoTEC) method for cooperative vehicular traffic system. The performance evaluation of this method is done using a unique open source simulation platform called iTETRIS. The results show that the method provides better and accurate results for different traffic scenarios with congestion problems.

Automatic traffic control and surveillance are the important factors for road usage and management. Timers for each stage are the simplest way to control the traffic. Another way is the usage of electronic sensors that can be used to find out vehicles and to produce signal as discussed by Chandrasekhar et al. [7].

Advanced Driver Assistance Systems (ADAS) implemented by Zhao et al. [6] describes the demand for high precision navigation in urban environments using low cost sensors by employing a sliding window smoothing estimator on tightly coupled differential global positioning system. The paper suggest a traffic system using masking algorithm to control the traffic density. For capturing still images of traffic on roads, web camera is used in each stage of traffic. After which, image matching is done using an reference frame.

**BLOCK DIAGRAM**

**Traffic Signal System**

RF Receiver

Decoder

**AVR**

**MICRO-CONTROLLER**

Led Light for Signal

Power Supply

**EMERGENCY VEHICLES**

Key Button

Encoder

RF Receiver

**HARDWARE REQUIREMENTS &** **DESCRIPTION**

**HARDWARE REQUIREMENTS**

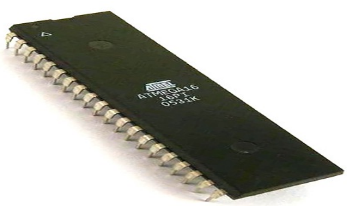
* AVRATMEGA Microcontroller
* RF Receiver
* Decoder
* LED
* Power Supply

**ATMEGA16 MICROCONTROLLER**

AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed to one-time programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time. AVR microcontrollers find many applications as embedded systems. ATmega16 is an 8-bit high performance microcontroller from the Atmel's Mega AVR family. Atmega16 is a 40 pin microcontroller based on enhanced RISC (Reduced Instruction Set Computing) architecture with 131 powerful instructions. It has a 16 KB programmable flash memory, static RAM of 1 KB and EEPROM of 512 Bytes. he endurance cycle of flash memory and EEPROM is 10,000 and 100,000, respectively.

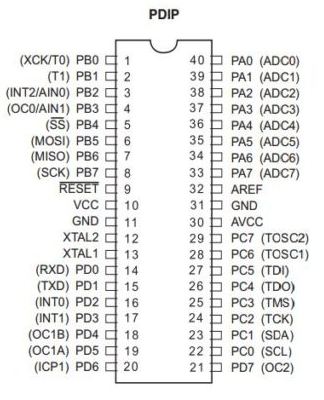
Most of the instructions execute in one machine cycle. It can work on a maximum frequency of 16MHz. ATmega16 pin diagram should clarify things a bit. This is the advanced version of the 8051 microcontrollers which has the features beat the 8051 microcontroller features. It’s a computer inbuilt with CPU, RAM, ROM, EEPROM, Timers, Counters, ADC and last four 8 bit ports like port A, port B, port C, port D. Each port has 8 input and output pins for extra performance. In the below section, we can observe the features of this microcontroller.

Atmel Corporation manufactured the ATmega16 microcontroller which comes under Atmel’s Advanced Virtual RISC family. It has an advanced RISC (Reduced Instruction Set Computing) system and a high-performance microcontroller. This is the advanced version of the 8051 microcontrollers which has the features beat the 8051 microcontroller features. It’s a computer inbuilt with CPU, RAM, ROM, EEPROM, Timers, Counters, ADC and last four 8 bit ports like port A, port B, port C, port D. Each port has 8 input and output pins for extra performance. In the below section, we can observe the features of this microcontroller.



**ATmega16 pin diagram**

This microcontroller has 40 pins and each pin has its importance. In these 40 pins, I/O pins are 32. And these are categorized into 4 ports. Each port having 8 I/O pins.

****

**ATmega16 Pin description**

|  |  |
| --- | --- |
| PIN1 | I/O , T0 ( Timer0 External Counter Input) ,XCK : USART External Clock I/O |
| PIN2 | I/O, T1 (Timer1 External Counter Input) |
| PIN3 | I/O, AIN0: Analog Comparator Positive Input , INT2: External Interrupt 2 Input |
| PIN4 | I/O, AIN1: Analog Comparator Negative Input, OC0 : Timer0 Output Compare Match Output |
| PIN9 | Reset Pin, Active Low Reset |
| PIN10 | VCC=+5V |
| PIN11 | GND |
| PIN12 | XTAL2 |
| PIN13 | XTAL1 |
| PIN14 | (RXD) ,I/O PIN 0,USART Serial Communication Interface |
| PIN15 | (TXD) ,I/O Pin 1,USART Serial Communication Interface |
| PIN16 | (INT0),I/O Pin 2, External Interrupt INT0 |
| PIN17 | (INT1),I/O Pin 3, External Interrupt INT1 |
| PIN18 | (OC1B),I/O Pin 4, PWM Channel Outputs |
| PIN19 | (OC1A),I/O Pin 5, PWM Channel Outputs |
| PIN20 | (ICP), I/O Pin 6, Timer/Counter1 Input Capture Pin |
| PIN21 | (OC2),I/O Pin 7,Timer/Counter2 Output Compare Match Output |
| PIN22 | (SCL),I/O Pin 0,TWI Interface |
| PIN23 | (SDA),I/O Pin 1,TWI Interface |
| PIN24-27 | JTAG INTERFACE |
| PIN28 : | (TOSC1),I/O Pin 6,Timer Oscillator Pin 1 |
| PIN29 : | (TOSC2),I/O Pin 7,Timer Oscillator Pin 2 |
| PIN30 : | AVCC (for ADC) |
| PIN31 : | GND (for ADC) |
| PIN33 – PIN40 | PAx: I/O, ADCx (Where x is 7 – 0) |

4 PORT-A 8 pins (pin 33-40)

1 PORT-B 8 pins (pin 1-8)

3 PORT-C 8 pins (pin 22-29)

2 PORT-D 8 pins (pin 14-21)

PORT-A: Here, PIN 33 to 40 are coming to PORT – A. This port A acts as an analog input to A/D converter. Port A can be used as 8 bit bidirectional I/O port. It has an internal pull-up resistor.

PORT – B: It has the pins from 1 to 8. This port B is used for I/O bidirectional pins.

PORT – C: This port C having eight I/O bidirectional pins.

PORT – D: Port D pins can be used as input or output pin. The extra peripherals like PWM channels, timer/counter, USART are connected to this port.

RESET – Pin 9 is for Reset pin.

Pin 10 – This pin is used for power supply purposes. By this pin, a power supply of 5V can be connected to the microcontroller.

Pin 12 & Pin 13 – High clock pulses can be generated by a crystal oscillator. And this crystal oscillator is connected to these pins. This microcontroller works at the 1MHz frequency.

**ATmega16 Programming**

There are multiple ways to program the ATmega16 and AVR microcontrollers. Here are the ways to do the ATmega16 programming. The following methods are helpful for how to burn the code into an ATmega16 microcontroller. They are:

Installing of USBASP version 2.0 programmer drivers on the computers.It can be done with the Atmel studio installer package.

Designing and updating Sketch into Atmega16.Finally, can be completed by ATmega16 with one LED and Oscillator circuit.

**Power**: Every microcontroller required power to operate and ATMega16 gives three power pins, one is power input and the remaining two are ground pins, those are internally connected with each other. Those pins in the microcontroller are:

* VCC – Pin10, GND – Pin11, Pin31

**Oscillator**: ATMega16 has internal 8 MHz changeable oscillator but to extend the clock speed an external oscillator will be used at the oscillator pins of the microcontroller which are given below:

* XTAL2 – GPIO12
* XTAL1 – GPIO13

**Digital Input/Output**: Input and output is the requirement of each device and micro-controller need these functions to communicate with external devices. In ATMega16 there are four ports (A, B, C & D) which comes with a total of 32 input/output pins. All these pins can be used for any input/output control function with external peripherals.

**Interrupt**: It is one of the most used functions during variating the high voltages but the basic purpose of this function is to get the attention of CPU. In this microcontroller, there are total three interrupt function which can be used by external modules or by an external button to get the attention of CPU. These pins in ATmega16 are listed below:

* INT0 – GPIO16
* INT1 – GPIO17
* INT2 – GPIO3

**USART**: External devices need to communicate with the microcontroller for multiple kinds of operations. ATMega16 gives a serial communication method for communication and programming. The UART pins are available in the microcontroller to operate with devices serially. It uses two communication pins and some internal programming to communicate but those pins can also be used for programming by using a third pin known as clock pins. The clock pins can also be used for simple data communication instead of programming. All those pins in the microcontroller are listed below:

* TX (output) – GPIO15
* RX (input) – GPIO14
* XCK (clock I/O) – GPIO1

**SPI**: SPI protocol is one of the best serial communication protocols for multiple modules. It can be used in the case of multiple peripherals to communicate efficiently with the microcontroller. The communication wires consist of three wires two for data and one for the clock, but it also has a third wire which is used for device selection. The device selection pin is knowing as Select Slave pin and it is predefined in the microcontroller but any output pin can be made as a select slave pin by programming. SPI protocol not only use for communication, but it can also be used to program the microcontroller. All SPI pins of ATMega16 are:

* SS’ – GPIO5
* MOSI – GPIO6
* MISO – GPIO7
* SCK – GPIO8

**I2C**: Some sensors and servos come with another serial communication protocol called I2C. To communicate with those peripherals ATMega16 also gives I2C pins interface. One pin is used for data communication and one for a clock. Both pins are listed below:

* SDA – GPIO23
* SCL – GPIO22

**JTAG**: These pins are present in most of the board for testing purposes. The reason for designing the JTAG was to test the device and PCB after the manufacturer has completed the design. JTAG is connected to the internal test port but it can also be used for programming the microcontroller and even the bootloader. JTAG pins in the microcontroller are:

* TDI (test data in) – GPIO27
* TDO (test data out) – GPIO26
* TMS (test mode selects) – GPIO25
* TCK (test clock) – GPIO24

**Timer:** The microcontroller has three internal timers/counter. They can perform by using the internal oscillator or they can also be operated with oscillators used by the microcontroller, but they also could perform the counter by using their own external oscillator. Two timers are 8-bit and one of 16 pins, only two timers support the external pulse input pins to operate. All timers and oscillator pins for timers are given below:

* T0 – GPIO1
* T1 – GPIO2
* TOSC1 – GPIO28
* TOSC2 – GPIO29

**Analog Comparator:** To compare the analog signal a comparator is being used in the microcontroller. The comparator takes the two inputs of the same signal and the input will be in inverting and non-inverting form. After comparing the analog signal internally their output can be used internally or it can be used to perform any other function on the output pins but all of them will be handled by programming. Analog comparator pins in ATMega16 are listed below:

* AN0 – GPIO3
* AN1 – GPIO4

**Capture/Compare/PWM**: PWM becomes a basic function for most of the devices to control their voltages. In ATMega16 there are four PWM pins that use Prescaler to generate the desired output signal. Those pins are:

* OC0 – GPIO4
* OC1A – GPIO19
* OC1B – GPIO18
* OC2 – GPIO21

**Input Capture**: ATMega16 can calculate the external impulse pulse duty cycle and frequency. This calculator can be used to perform the further operation. There is only one pin in the microcontroller which could do that, which is listed below:

* ICP – GPIO20

**Analog to Digital Converter**: ADC is the requirement of every microcontroller due to the output of multiple modules in analog signal and ATMega16 gives the 8 A/D channels. In the microcontroller, all ADC can be used for analog converter individually but the first analog converter will require external power to act at its AVCC pins.

**AREF**: Analog signal maximum value is decided by the supply voltages which sometimes affect the output because of different voltage levels by the analog output device. To overcome this issue an analog Reference pin will be used, where analog output device voltages will be used to measure the output according to the given voltages. Aref pin in the microcontroller is listed below:

* AREF – GPIO32

**RESET**: To reset the microcontroller ATMega16 an external an internal reset is used. An internal reset can be used through programming but to use the external, a low pulse signal will be required.

* Reset – GPIO9

**Applications**

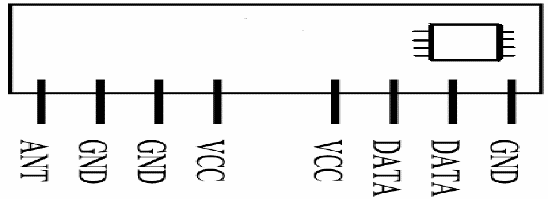
* The microcontroller found in commercial products.
* It is compatible with multi-sensor devices because most of the pins perform only a single operation.
* Smart industrial machines also use the ATMega16 due to its cheapness and functionality.
* To calculate the frequency and duty cycle of the external device, the ATMega16 is used.

**RF RECEIVER**

**433 MHz RF Receiver**:

It is an ideal for short-range remote control applications where cost is a primary concern. The receiver module requires no external RF components except for the antenna. It generates virtually no emissions, making FCC and ETSI approvals easy. The super-regenerative design exhibits exceptional sensitivity at a very low cost.

**PIN DIAGRAM**:



**PIN DESCRIPTION:**

ANT:

 Antenna input.

GND:

Receiver Ground. Connect to ground plane.

VCC (5V):

VCC pins are electrically connected and provide operating voltage for the Receiver. VCC can be applied to either or both. VCC should be bypassed with a .1μF ceramic capacitor. Noise on the power supply will degrade receiver sensitivity.

DATA:

Digital data output. This output is capable of driving one TTL or CMOS load. It is a CMOS compatible output.

**Operation:**

**Super-Regenerative AM Detection:**

 The RF Receiver module uses a super-regenerative AM detector to demodulate the incoming AM carrier. A super regenerative detector is a gain stage with positive feedback greater than unity so that it oscillates. An RC-time constant is included in the gain stage so that when the gain stage oscillates, the gain will be lowered over time proportional to the RC time constant until the oscillation eventually dies. When the oscillation dies, the current draw of the gain stage decreases, charging the RC circuit, increasing the gain, and ultimately the oscillation starts again. In this way, the oscillation of the gain stage is turned on and off at a rate set by the RC time constant. This rate is chosen to be super-audible but much lower than the main oscillation rate. Detection is accomplished by measuring the emitter current of the gain stage. Any RF input signal at the frequency of the main oscillation will aid the main oscillation in restarting. If the amplitude of the RF input increases, the main oscillation will stay on for a longer period of time, and the emitter current will be higher. Therefore, we can detect the original base-band signal by simply low-pass filtering the emitter current. The average emitter current is not very linear as a function of the RF input level. It exhibits a 1/ln response because of the exponentially rising nature of oscillator start-up. The steep slope of a logarithm near zero results in high sensitivity to small input signals.

**Data Slicer**

          The data slicer converts the base-band analog signal from the super-regenerative detector to a CMOS/TTL compatible output. Because the data slicer is AC coupled to the audio output, there is a minimum data rate. AC coupling also limits the minimum and maximum pulse width. Typically, data is encoded on the transmit side using pulse-width modulation (PWM) or non-return-to-zero (NRZ).

The most common source for NRZ data is from a UART embedded in a micro-controller. Applications that use NRZ data encoding typically involve microcontrollers. The most common source for PWM data is from a remote control IC such as the HC-12E.

Data is sent as a constant rate square-wave. The duty cycle of that square wave will generally be either 33% (a zero) or 66% (a one). The data slicer on the STR-433 is optimized for use with PWM encoded data, though it will work with NRZ data if certain encoding rules are followed.

**Power Supply:**

The STR-433 is designed to operate from a 5V power supply. It is crucial that this power supply be very quiet. The power supply should be bypassed using a 0.1uF low-ESR ceramic capacitor and a 4.7uF tantalum capacitor. These capacitors should be placed as close to the power pins as possible. The STR- 433 is designed for continuous duty operation. From the time power is applied, it can take up to 750mSec for the data output to become valid.

**Antenna Input:**

It will support most antenna types, including printed antennas integrated directly onto the PCB and simple single core wire of about 17cm. The performance of the different antennas varies. Any time a trace is longer than 1/8th the wavelength of the frequency it is carrying, it should be a 50 ohm microstrip.

**Features:**

* Low Cost
* 5V operation
* 3.5mA current drain
* No External Parts are required
* Receiver Frequency: 433.92 MHZ
* Typical sensitivity: -105dBm
* IF Frequency: 1MHz

**Applications:**

* Car security system
* Sensor reporting
* Automation system
* Remote Keyless Entry (RKE)
* Remote Lighting Controls
* On-Site Paging
* Asset Tracking
* Wireless Alarm and Security Systems
* Long Range RFID
* Automated Resource Management

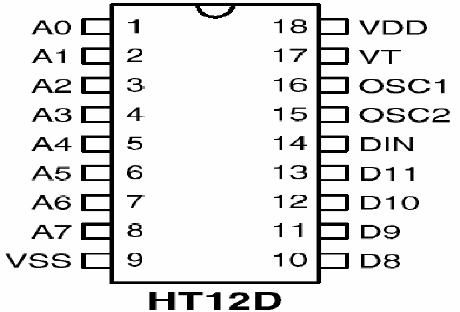
**DECODER**

A **decoder** is a device which does the reverse of an encoder, undoing the encoding so that the original information can be retrieved. The same method used to encode is usually just reversed in order to decode. In digital electronics this would mean that a decoder is a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs. Enable inputs must be on for the decoder to function, otherwise its outputs assume a single "disabled" output code word. Decoding is necessary in applications such as data multiplexing, 7 segment display and memory address decoding.

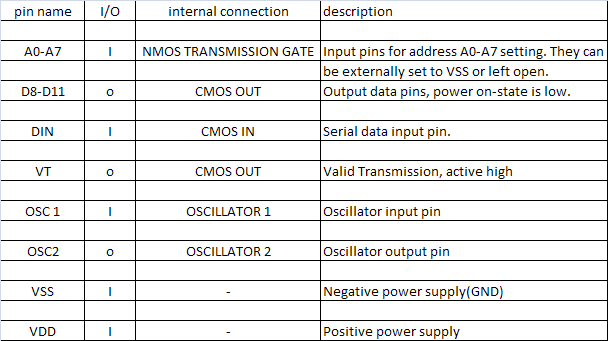
**HT12D:**

The HT12D is a decoder IC made especially to pair with the HT12E encoder. It is a CMOS IC made for remote control system application. The decoder is capable of decoding 8 bits of address (A0-A7) and 4 bits of data (AD8-AD11) information. For proper operation, a pair of encoder/decoder with the same number of addresses and data format should be chosen. The decoders receive serial addresses and data from programmed encoders that are transmitted by a carrier using an RF or an IR transmission medium. They compare the serial input data three times continuously with their local addresses. If no error or unmatched codes are found, the input data codes are decoded and then transferred to the output pins. The VT pin also goes high to indicate a valid transmission. The decoders are capable of decoding information that consists of N bits of address and 12\_N bits of data. Of this series, the HT12D is arranged to provide 8 address bits and 4 data bits, and HT12F is used to decode 12 bits of address information.

**Pin Diagram**

****

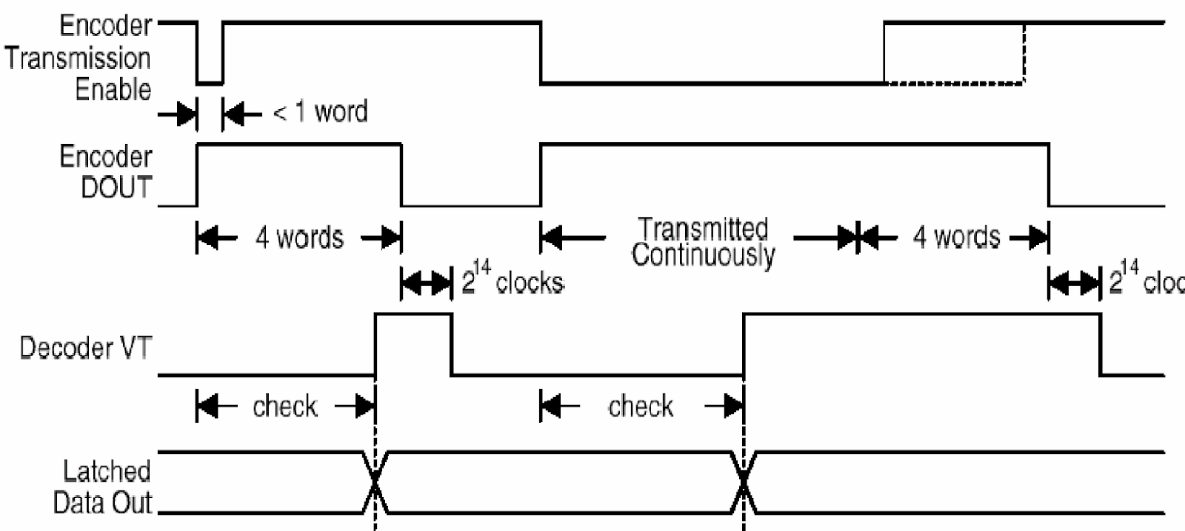
**Pin Description:**



**Decoder Operation**

HT12D receives digital serial data from its DIN(pin14). A signal in the DIN activates the oscillator which then decodes the incoming address and data.

**Decoder Timing**



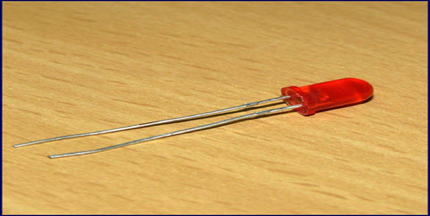
After decoding, it checks the serial input data three times continuously with its local addresses. If no error or unmatched codes are found, the input data codes are decoded and then transferred to the data output pins. This pin remains high for 214=16384 decoder clocks after the encoder’s DOUT pin goes low. Since the decoder operates at 150 KHz, it takes 150000\*16384=0.1 seconds for the VT pin to go low. This pin also goes low if the address code is incorrect or no signal is received. The 4 data pins are latched to their respective pins, meaning that the previous data remains on the pins unless a new data arrives to replace the existing one.

**Applications**

1. Burglar alarm system
2. Smoke and fire alarm system
3. Garage door controllers
4. Car door controllers
5. Car alarm system, Security system
6. Cordless telephones, Other remote control systems

**LED**

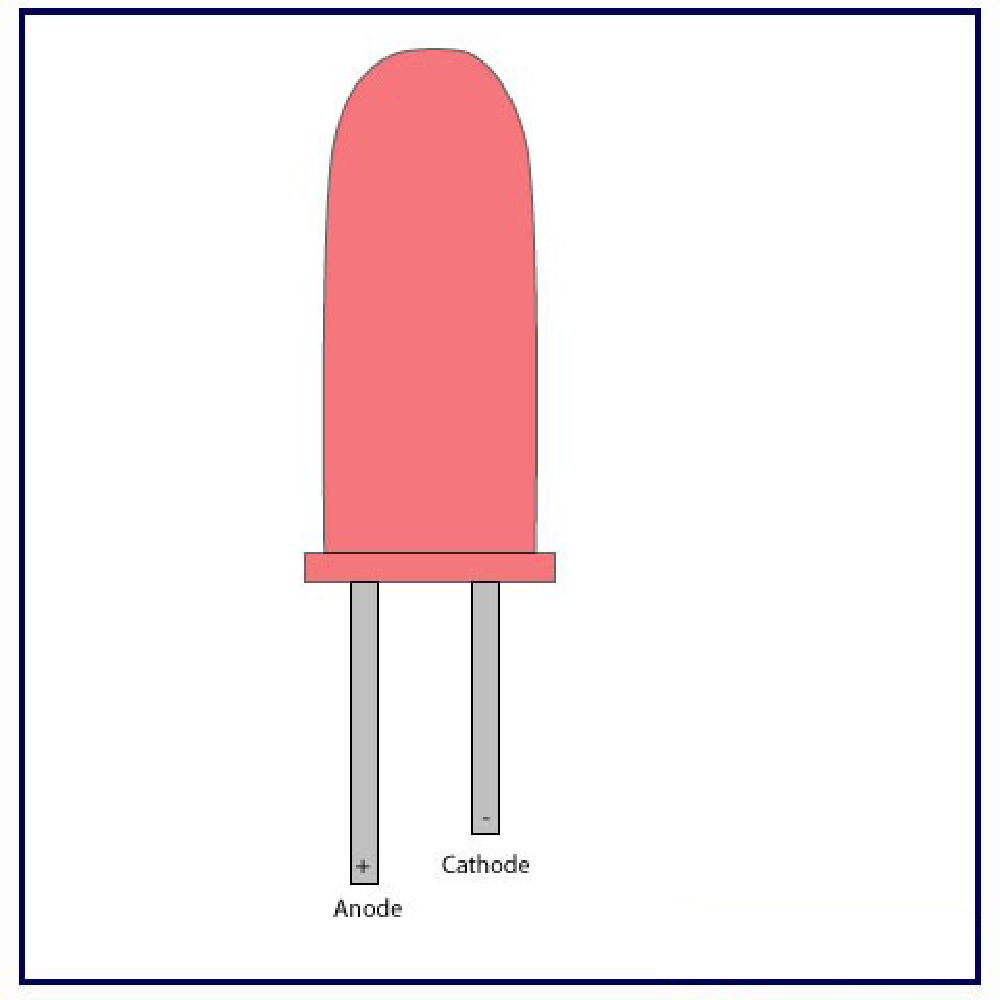
**Light emitting diodes** (**LEDs**) are semiconductor light sources. The light emitted from **LED**s varies from visible to infrared and ultraviolet regions. They operate on low voltage and power. LEDs are one of the most common electronic components and are mostly used as indicators in circuits. They are also used for luminance and optoelectronic applications.



Based on semiconductor diode, **LED**s emit photons when electrons recombine with holes on forward biasing. The two terminals of LEDs are anode (+) and cathode (-) and can be identified by their size. The longer leg is the positive terminal or anode and shorter one is negative terminal.

The forward voltage of **LED** (1.7V-2.2V) is lower than the voltage supplied (5V) to drive it in a circuit. Using an LED as such would burn it because a high current would destroy its p-n gate. Therefore a current limiting resistor is used in series with LED. Without this resistor, either low input voltage (equal to forward voltage) or PWM (pulse width modulation) is used to drive the LED. Get details about internal structure of a LED.

**Pin Diagram:**



A light-emitting diode (LED) is a semiconductor light source that emits light when current flows through it. Electrons in the semiconductor recombine with electron holes, releasing energy in the form of photons (Energy packets). The color of the light (corresponding to the energy of the photons) is determined by the energy required for electrons to cross the band gap of the semiconductor. White light is obtained by using multiple semiconductors or a layer of light-emitting phosphor on the semiconductor device.

Appearing as practical electronic components in 1962, the earliest LEDs emitted low-intensity infrared (IR) light. Infrared LEDs are used in remote-control circuits, such as those used with a wide variety of consumer electronics. The first visible-light LEDs were of low intensity and limited to red. Early LEDs were often used as indicator lamps, replacing small incandescent bulbs, and in seven-segment displays. Later developments produced LEDs available in visible, ultraviolet (UV), and infrared wavelengths, with high, low, or intermediate light output, for instance white LEDs suitable for room and outdoor area lighting. LEDs have also given rise to new types of displays and sensors, while their high switching rates are useful in advanced communications technology with applications as diverse as aviation lighting, fairy lights, automotive headlamps, advertising, general lighting, traffic signals, camera flashes, lighted wallpaper, horticultural grow lights, and medical devices.

LEDs have many advantages over incandescent light sources, including lower power consumption, longer lifetime, improved physical robustness, smaller size, and faster switching. In exchange for these generally favorable attributes, disadvantages of LEDs include electrical limitations to low voltage and generally to DC (not AC) power, inability to provide steady illumination from a pulsing DC or an AC electrical supply source, and lesser maximum operating temperature and storage temperature. In contrast to LEDs, incandescent lamps can be made to intrinsically run at virtually any supply voltage, can utilize either AC or DC current interchangeably, and will provide steady illumination when powered by AC or pulsing DC even at a frequency as low as 50 Hz. LEDs usually need electronic support components to function, while an incandescent bulb can and usually does operate directly from an unregulated DC or AC power source.

**RGB systems**

Mixing red, green, and blue sources to produce white light needs electronic circuits to control the blending of the colors. Since LEDs have slightly different emission patterns, the colour balance may change depending on the angle of view, even if the RGB sources are in a single package, so RGB diodes are seldom used to produce white lighting. Nonetheless, this method has many applications because of the flexibility of mixing different colours, and in principle, this mechanism also has higher quantum efficiency in producing white light.

  There are several types of multicolour white LEDs: di-, tri-, and tetra chromatic white LEDs. Several key factors that play among these different methods include color stability, color rendering capability, and luminous efficacy. Often, higher efficiency means lower color rendering, presenting a trade-off between the luminous efficacy and color rendering. For example, the dichromatic white LEDs have the best luminous efficacy (120 lm/W), but the lowest color rendering capability. Although tetra chromatic white LEDs have excellent color rendering capability, they often have poor luminous efficacy. Trichromatic white LEDs are in between, having both good luminous efficacy (>70 lm/W) and fair color rendering capability.

  One of the challenges is the development of more efficient green LEDs. The theoretical maximum for green LEDs is 683 lumens per watt but as of 2010 few green LEDs exceed even 100 lumens per watt. The blue and red LEDs approach their theoretical limits. Multicolor LEDs offer a means to form light of different colors. Most perceivable colors can be formed by mixing different amounts of three primary colors. This allows precise dynamic color control. Their emission power decays exponentially with rising temperature, resulting in a substantial change in color stability. Such problems inhibit industrial use. Multicolor LEDs without phosphors cannot provide good color rendering because each LED is a narrowband source. LEDs without phosphor, while a poorer solution for general lighting, are the best solution for displays, either backlight of LCD, or direct LED based pixels.

  Dimming a multicolour LED source to match the characteristics of incandescent lamps is difficult because manufacturing variations, age, and temperature change the actual color value output. To emulate the appearance of dimming incandescent lamps may require a feedback system with color sensor to actively monitor and control the color.

**Filament**

An LED filament consists of multiple LED chips connected in series on a common longitudinal substrate that forms a thin rod reminiscent of a traditional incandescent filament. These are being used as a low-cost decorative alternative for traditional light bulbs that are being phased out in many countries. The filaments use a rather high voltage, allowing them to work efficiently with mains voltages. Often a simple rectifier and capacitive current limiting are employed to create a low-cost replacement for a traditional light bulb without the complexity of the low voltage, high current converter that single die LEDs need. Usually, they are packaged in bulb similar to the lamps they were designed to replace, and filled with inert gas at slightly lower than ambient pressure to remove heat efficiently and prevent corrosion.

**Advantages**

Efficiency: LEDs emit more lumens per watt than incandescent light bulbs.[133] The efficiency of LED lighting fixtures is not affected by shape and size, unlike fluorescent light bulbs or tubes.

Color: LEDs can emit light of an intended color without using any color filters as traditional lighting methods need. This is more efficient and can lower initial costs.

Size: LEDs can be very small (smaller than 2 mm) and are easily attached to printed circuit boards.

Switch on time: LEDs light up extremely quickly. A typical red indicator LED achieves full brightness in under a microsecond. LEDs used in communications devices can have even faster response times.

Cycling: LEDs are ideal for uses subject to frequent on-off cycling, unlike incandescent and fluorescent lamps that fail faster when cycled often, or high-intensity discharge lamps (HID lamps) that require a long time before restarting.

Dimming: LEDs can very easily be dimmed either by pulse-width modulation or lowering the forward current. This pulse-width modulation is why LED lights, particularly headlights on cars, when viewed on camera or by some people, seem to flash or flicker. This is a type of stroboscopic effect.

Cool light: In contrast to most light sources, LEDs radiate very little heat in the form of IR that can cause damage to sensitive objects or fabrics. Wasted energy is dispersed as heat through the base of the LED.

Slow failure: LEDs mainly fail by dimming over time, rather than the abrupt failure of incandescent bulbs.

Lifetime: LEDs can have a relatively long useful life. One report estimates 35,000 to 50,000 hours of useful life, though time to complete failure may be shorter or longer. Fluorescent tubes typically are rated at about 10,000 to 25,000 hours, depending partly on the conditions of use, and incandescent light bulbs at 1,000 to 2,000 hours. Several DOE demonstrations have shown that reduced maintenance costs from this extended lifetime, rather than energy savings, is the primary factor in determining the payback period for an LED product.

Shock resistance: LEDs, being solid-state components, are difficult to damage with external shock, unlike fluorescent and incandescent bulbs, which are fragile.

Focus: The solid package of the LED can be designed to focus its light. Incandescent and fluorescent sources often require an external reflector to collect light and direct it in a usable manner. For larger LED packages total internal reflection (TIR) lenses are often used to the same effect. When large quantities of light are needed, many light sources are usually deployed, which are difficult to focus or collimate on the same target.

**Disadvantages**

Temperature dependence: LED performance largely depends on the ambient temperature of the operating environment – or thermal management properties. Overdriving an LED in high ambient temperatures may result in overheating the LED package, eventually leading to device failure. An adequate heat sink is needed to maintain long life. This is especially important in automotive, medical, and military uses where devices must operate over a wide range of temperatures, and require low failure rates.

Voltage sensitivity: LEDs must be supplied with a voltage above their threshold voltage and a current below their rating. Current and lifetime change greatly with a small change in applied voltage. They thus require a current-regulated supply (usually just a series resistor for indicator LEDs).

Color rendition: Most cool-white LEDs have spectra that differ significantly from a black body radiator like the sun or an incandescent light. The spike at 460 nm and dip at 500 nm can make the color of objects appear differently under cool-white LED illumination than sunlight or incandescent sources, due to metamerism, red surfaces being rendered particularly poorly by typical phosphor-based cool-white LEDs. The same is true with green surfaces. The quality of color rendition of an LED is measured by the Color Rendering Index (CRI).

Area light source: Single LEDs do not approximate a point source of light giving a spherical light distribution, but rather a lambertian distribution. So, LEDs are difficult to apply to uses needing a spherical light field. Different fields of light can be manipulated by the application of different optics or "lenses". LEDs cannot provide divergence below a few degrees.

Light pollution: Because white LEDs emit more short wavelength light than sources such as high-pressure sodium vapor lamps, the increased blue and green sensitivity of scotopic vision means that white LEDs used in outdoor lighting cause substantially more sky glow.

Efficiency droop: The efficiency of LEDs decreases as the electric current increases. Heating also increases with higher currents, which compromises LED lifetime. These effects put practical limits on the current through an LED in high power applications.

Impact on wildlife: LEDs are much more attractive to insects than sodium-vapor lights, so much so that there has been speculative concern about the possibility of disruption to food webs. LED lighting near beaches, particularly intense blue and white colors, can disorient turtle hatchlings and make them wander inland instead. The use of "turtle-safe lighting" LEDs that emit only at narrow portions of the visible spectrum is encouraged by conservancy groups in order to reduce harm.

Use in winter conditions: Since they do not give off much heat in comparison to incandescent lights, LED lights used for traffic control can have snow obscuring them, leading to accidents.

Thermal runaway: Parallel strings of LEDs will not share current evenly due to the manufacturing tolerances in their forward voltage. Running two or more strings from a single current source may result in LED failure as the devices warm up. If forward voltage binning is not possible, a circuit is required to ensure even distribution of current between parallel strands.

**Applications**

·        Visual signals where light goes more or less directly from the source to the human eye, to convey a message or meaning

·        Illumination where light is reflected from objects to give visual response of these objects

·        Measuring and interacting with processes involving no human vision

·        Narrow band light sensors where LEDs operate in a reverse-bias mode and respond to incident light, instead of emitting light

·        Indoor cultivation, including cannabis.

**Uses**

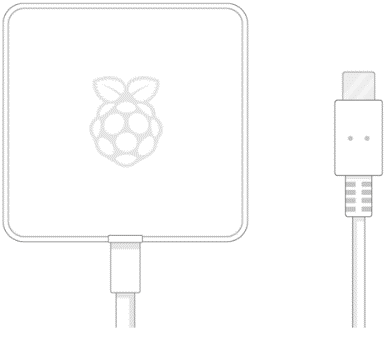
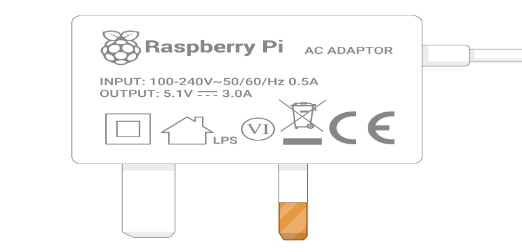
·        Power sources

·        Electrical polarity

·        Safety and health

**A POWER SUPPLY**

Power supply is a reference to a source of electrical power. A device or system that supplies electrical or other types of energy to an output load or group of loads is called a power supply unit or PSU. The term is most commonly applied to electrical energy supplies, less often to mechanical ones, and rarely to others. Power supplies for electronic devices can be broadly divided into linear and switching power supplies. The linear supply is a relatively simple design that becomes increasingly bulky and heavy for high current devices; voltage regulation in a linear supply can result in low efficiency. A switched-mode supply of the same rating as a linear supply will be smaller, is usually more efficient, but will be more complex.

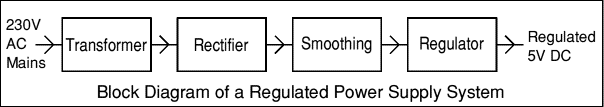
You need a power supply that provides:

* At least 3.0 amps for raspberry pi 4
* At least 2.5 amps for raspberry pi 3

**Linear Power supply**

An AC powered linear power supply usually uses a transformer to convert the voltage from the wall outlet (mains) to a different, usually a lower voltage. If it is used to produce DC, a rectifier is used. A capacitor is used to smooth the pulsating current from the rectifier. Some small periodic deviations from smooth direct current will remain, which is known as ripple. These pulsations occur at a frequency related to the AC power frequency (for example, a multiple of 50 or 60 Hz).

The voltage produced by an unregulated power supply will vary depending on the load and on variations in the AC supply voltage. For critical electronics applications a linear regulator will be used to stabilize and adjust the voltage. This regulator will also greatly reduce the ripple and noise in the output direct current. Linear regulators often provide current limiting, protecting the power supply and attached circuit from over current. Adjustable linear power supplies are common laboratory and service shop test equipment, allowing the output voltage to be set over a wide range. For example, a bench power supply used by circuit designers may be adjustable up to 30 volts and up to 5 amperes output. Some can be driven by an external signal, for example, for applications requiring a pulsed output.



**SOFTWARE REQUIREMENTS AND DESCRIPTION**

**ARDUINO SOFTWARE (IDE)**

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino and Genuino hardware to upload programs and communicate with them.

**Writing Sketches**

Programs written using Arduino Software (IDE) are called sketches. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom righthand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

NB: Versions of the Arduino Software (IDE) prior to 1.0 saved sketches with the extension .pde. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the .ino extension on save.

|  |  |
| --- | --- |
| https://www.arduino.cc/en/uploads/Guide/play.png | *Verify*  Checks your code for errors compiling it. |
| https://www.arduino.cc/en/uploads/Guide/export.png | *Upload*  Compiles your code and uploads it to the configured board. See [uploading](https://www.arduino.cc/en/Guide/Environment#uploading) below for details.  Note: If you are using an external programmer with your board, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer" |
| https://www.arduino.cc/en/uploads/Guide/new.png | *New*  Creates a new sketch. |
| https://www.arduino.cc/en/uploads/Guide/open.png | *Open*  Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.  Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the File | Sketchbookmenu instead. |
| https://www.arduino.cc/en/uploads/Guide/save.png | *Save*  Saves your sketch. |
| https://www.arduino.cc/en/uploads/Guide/serial_monitor.png | *Serial Monitor*  Opens the [serial monitor](https://www.arduino.cc/en/Guide/Environment#serialmonitor). |

Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

**File**

* **New**  
  Creates a new instance of the editor, with the bare minimum structure of a sketch already in place.
* **Open**   
  Allows to load a sketch file browsing through the computer drives and folders.
* **Open Recent**Provides a short list of the most recent sketches, ready to be opened.
* **Sketchbook**  
  Shows the current sketches within the sketchbook folder structure; clicking on any name opens the corresponding sketch in a new editor instance.
* **Examples**  
  Any example provided by the Arduino Software (IDE) or library shows up in this menu item. All the examples are structured in a tree that allows easy access by topic or library.
* **Close**   
  Closes the instance of the Arduino Software from which it is clicked.
* **Save**   
  Saves the sketch with the current name. If the file hasn't been named before, a name will be provided in a "Save as.." window.
* **Save as...**Allows to save the current sketch with a different name.
* **Page Setup**It shows the Page Setup window for printing.
* **Print**   
  Sends the current sketch to the printer according to the settings defined in Page Setup.
* **Preferences**   
  Opens the Preferences window where some settings of the IDE may be customized, as the language of the IDE interface.
* **Quit**   
  Closes all IDE windows. The same sketches open when Quit was chosen will be automatically reopened the next time you start the IDE.

**Edit**

* **Undo/Redo**   
  Goes back of one or more steps you did while editing; when you go back, you may go forward with Redo.
* **Cut**   
  Removes the selected text from the editor and places it into the clipboard.
* **Copy**   
  Duplicates the selected text in the editor and places it into the clipboard.
* **Copy for Forum**Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax coloring.
* **Copy as HTML**  
  Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.
* **Paste**  
  Puts the contents of the clipboard at the cursor position, in the editor.
* **Select All**  
  Selects and highlights the whole content of the editor.
* **Comment/Uncomment**  
  Puts or removes the // comment marker at the beginning of each selected line.
* **Increase/Decrease Indent**  
  Adds or subtracts a space at the beginning of each selected line, moving the text one space on the right or eliminating a space at the beginning.
* **Find**  
  Opens the Find and Replace window where you can specify text to search inside the current sketch according to several options.
* **Find Next**Highlights the next occurrence - if any - of the string specified as the search item in the Find window, relative to the cursor position.
* **Find Previous**  
  Highlights the previous occurrence - if any - of the string specified as the search item in the Find window relative to the cursor position.

**Sketch**

* **Verify/Compile**Checks your sketch for errors compiling it; it will report memory usage for code and variables in the console area.
* **Upload**  
  Compiles and loads the binary file onto the configured board through the configured Port.
* **Upload Using Programmer**  
  This will overwrite the bootloader on the board; you will need to use Tools > Burn Bootloader to restore it and be able to Upload to USB serial port again. However, it allows you to use the full capacity of the Flash memory for your sketch. Please note that this command will NOT burn the fuses. To do so a *Tools -> Burn Bootloader* command must be executed.
* **Export Compiled Binary**Saves a .hex file that may be kept as archive or sent to the board using other tools.
* **Show Sketch Folder**  
  Opens the current sketch folder.
* **Include Library**  
  Adds a library to your sketch by inserting #include statements at the start of your code. For more details, see[libraries](https://www.arduino.cc/en/Guide/Environment#libraries) below. Additionally, from this menu item you can access the Library Manager and import new libraries from .zip files.
* **Add File...**  
  Adds a source file to the sketch (it will be copied from its current location). The new file appears in a new tab in the sketch window. Files can be removed from the sketch using the tab menu accessible clicking on the small triangle icon below the serial monitor one on the right side o the toolbar.

**Tools**

* **Auto Format**This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.
* **Archive Sketch**  
  Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
* **Fix Encoding & Reload**  
  Fixes possible discrepancies between the editor char map encoding and other operating systems char maps.
* **Serial Monitor**  
  Opens the serial monitor window and initiates the exchange of data with any connected board on the currently selected Port. This usually resets the board, if the board supports Reset over serial port opening.
* **Board**  
  Select the board that you're using. See below for [descriptions of the various boards](https://www.arduino.cc/en/Guide/Environment#boards).
* **Port**  
  This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.
* **Programmer**  
  For selecting a harware programmer when programming a board or chip and not using the onboard USB-serial connection. Normally you won't need this, but if you're [burning a bootloader](https://www.arduino.cc/en/Tutorial/Bootloader) to a new microcontroller, you will use this.
* **Burn Bootloader**  
  The items in this menu allow you to burn a [bootloader](https://www.arduino.cc/en/Hacking/Bootloader) onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino or Genuino board but is useful if you purchase a new ATmega microcontroller (which normally come without a bootloader). Ensure that you've selected the correct board from the Boards menu before burning the bootloader on the target board. This command also set the right fuses.

**Help**

Here you find easy access to a number of documents that come with the Arduino Software (IDE). You have access to Getting Started, Reference, this guide to the IDE and other documents locally, without an internet connection. The documents are a local copy of the online ones and may link back to our online website.

* **Find in Reference**  
  This is the only interactive function of the Help menu: it directly selects the relevant page in the local copy of the Reference for the function or command under the cursor.

**Sketchbook**

The Arduino Software (IDE) uses the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the File > Sketchbook menu or from the Open button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the Preferences dialog.

Beginning with version 1.0, files are saved with a .ino file extension. Previous versions use the .pde extension. You may still open .pde named files in version 1.0 and later, the software will automatically rename the extension to .ino.

**Tabs, Multiple Files, and Compilation**

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no visible extension), C files (.c extension), C++ files (.cpp), or header files (.h).

**Uploading**

Before uploading your sketch, you need to select the correct items from the Tools > Board and Tools > Port menus. The[boards](https://www.arduino.cc/en/Guide/Environment#boards) are described below. On the Mac, the serial port is probably something like /dev/tty.usbmodem241 (for an Uno or Mega2560 or Leonardo) or /dev/tty.usbserial-1B1 (for a Duemilanove or earlier USB board), or/dev/tty.USA19QW1b1P1.1 (for a serial board connected with a Keyspan USB-to-Serial adapter). On Windows, it's probably COM1 or COM2 (for a serial board) or COM4, COM5, COM7, or higher (for a USB board) - to find out, you look for USB serial device in the ports section of the Windows Device Manager. On Linux, it should be /dev/ttyACMx ,/dev/ttyUSBx or similar. Once you've selected the correct serial port and board, press the upload button in the toolbar or select the Upload item from the File menu. Current Arduino boards will reset automatically and begin the upload. With older boards (pre-Diecimila) that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the RX and TX LEDs blink as the sketch is uploaded. The Arduino Software (IDE) will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino bootloader, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

**Libraries**

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the Sketch > Import Library menu. This will insert one or more #include statements at the top of the sketch and compile the library with your sketch. Because libraries are uploaded to the board with your sketch, they increase the amount of space it takes up. If a sketch no longer needs a library, simply delete its #includestatements from the top of your code.

There is a [list of libraries](https://www.arduino.cc/en/Reference/Libraries) in the reference. Some libraries are included with the Arduino software. Others can be downloaded from a variety of sources or through the Library Manager. Starting with version 1.0.5 of the IDE, you do can import a library from a zip file and use it in an open sketch. See these [instructions for installing a third-party library](https://www.arduino.cc/en/Guide/Libraries).

**Third-Party Hardware**

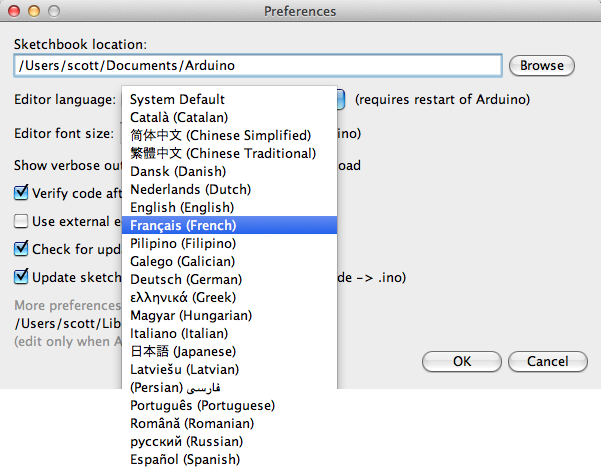
Support for third-party hardware can be added to the hardware directory of your sketchbook directory. Platforms installed there may include board definitions (which appear in the board menu), core libraries, bootloaders, and programmer definitions. To install, create the hardware directory, then unzip the third-party platform into its own sub-directory. (Don't use "arduino" as the sub-directory name or you'll override the built-in Arduino platform.) To uninstall, simply delete its directory.

**Serial Monitor**

Displays serial data being sent from the Arduino or Genuino board (USB or serial board). To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down that matches the rate passed to Serial.begin in your sketch. Note that on Windows, Mac or Linux, the Arduino or Genuino board will reset (rerun your sketch execution to the beginning) when you connect with the serial monitor.

**Preferences**

Some preferences can be set in the preferences dialog (found under the Arduino menu on the Mac, or File on Windows and Linux). The rest can be found in the preferences file, whose location is shown in the preference dialog.

**Language Support**

Since version 1.0.1 , the Arduino Software (IDE) has been translated into 30+ different languages. By default, the IDE loads in the language selected by your operating system. (Note: on Windows and possibly Linux, this is determined by the locale setting which controls currency and date formats, not by the language the operating system is displayed in.)

If you would like to change the language manually, start the Arduino Software (IDE) and open the Preferences window. Next to the Editor Language there is a dropdown menu of currently supported languages. Select your preferred language from the menu, and restart the software to use the selected language. If your operating system language is not supported, the Arduino Software (IDE) will default to English.

You can return the software to its default setting of selecting its language based on your operating system by selectingSystem Default from the Editor Language drop-down. This setting will take effect when you restart the Arduino Software (IDE). Similarly, after changing your operating system's settings, you must restart the Arduino Software (IDE) to update it to the new default language.

Boards

The board selection has two effects: it sets the parameters (e.g. CPU speed and baud rate) used when compiling and uploading sketches; and sets and the file and fuse settings used by the burn bootloader command. Some of the board definitions differ only in the latter, so even if you've been uploading successfully with a particular selection you'll want to check it before burning the bootloader. You can find a comparison table between the various boards [here](https://www.arduino.cc/en/Products/Compare).

Arduino Software (IDE) includes the built in support for the boards in the following list, all based on the AVR Core. The[Boards Manager](https://www.arduino.cc/en/Guide/Cores) included in the standard installation allows to add support for the growing number of new boards based on different cores like Arduino Due, Arduino Zero, Edison, Galileo and so on.

**EMBEDDED C**

An embedded system is a [computer system](http://en.wikipedia.org/wiki/Computer_system) designed to perform one or a few dedicated functions often with [real-time computing](http://en.wikipedia.org/wiki/Real-time_computing) constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a [personal computer](http://en.wikipedia.org/wiki/Personal_computer) (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.

Embedded systems are controlled by one or more main processing cores that are typically either [microcontrollers](http://en.wikipedia.org/wiki/Microcontroller) or [digital signal processors](http://en.wikipedia.org/wiki/Digital_signal_processor) (DSP).The key characteristic, however, is being dedicated to handle a particular task, which may require very powerful processorsSince the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from [economies of scale](http://en.wikipedia.org/wiki/Economies_of_scale).

Peripherals

Embedded Systems talk with the outside world via [peripherals](http://en.wikipedia.org/wiki/Peripheral), such as:

* Serial Communication Interfaces (SCI): [RS-232](http://en.wikipedia.org/wiki/RS-232), [RS-422](http://en.wikipedia.org/wiki/RS-422), [RS-485](http://en.wikipedia.org/wiki/RS-485) etc.
* Synchronous Serial Communication Interface: [I2C](http://en.wikipedia.org/wiki/I2C), [SPI](http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus), SSC and ESSI (Enhanced Synchronous Serial Interface)
* [Universal Serial Bus](http://en.wikipedia.org/wiki/Universal_Serial_Bus) (USB)
* Analog to Digital/Digital to Analog ([ADC](http://en.wikipedia.org/wiki/Analog-to-digital_converter)/[DAC](http://en.wikipedia.org/wiki/Digital-to-analog_converter))

**Advantages**

·         Reliability

·         Simple control loop

·         Interrupt controlled system

**CONCLUSION AND FUTURE WORK**

The prototype for real time image processing to automate traffic signal system based on density estimation and emergency vehicle detection such as ambulance is done. The benefits of this new method include non-use of sensors or RFID tags which reduces the stress of traffic sergeants. MATLAB simulation reduces the production costs and helps to achieve high speed and accuracy. Further with improvements, this work can be extended to detect road accidents and to identify violations that occurs in the spiral car movements. The accuracy of this work can be improvised further by doing thermal image processing. Thermal image processing is effective even during extreme weather conditions such as, mist or fog. Secondly, cloud computing can be done for the road data analysis.

**REFERENCES**

[1] Akoum A, Daya B and Chauvet P 2010A New algorithmic approach for detection and identification of vehicle plate numbers J. Software Engineering & Applications. 3 99-108.

[2] Cosmo H, Munuo and Michael Kisangiri 2014Vehicle number plates detection and recognition using improved algorithms: a review with tanzanian case study International Journal of Engineering and Computer Science 3 5828-32.

[3] Kouvelas A,Aboudolas K, Papageorgiou M and Kosmatopoulos E B 2011 A hybrid strategy for real time traffic signal control of urban road networks IEEE Transaction Intelligent Transport System12 884-94.

[4] Zhang P, X.Wu C and Wong S C 2012A semi-discrete model and its approach to a solution for a wide moving jam in traffic flowPhys. A.Statist. Mech. Appl. 3 456-63.

[5] Bauza R and Gozalvez J 2013Traffic congestion detection in large- scale scenarios using vehicle to vehicle communications J.Netw. Comput. Appl.36 1295-1307.

[6] Zhao S, Chen Y and Farell J 2016 Ahigh-precision vehicle navigation in urban environments using an MEM’s IMU and single-frequency GPS receiver IEEE Transaction Intelligent Transport System 17 2854-67.

[7] Chandrasekhar M, Sai Krishna C, Chakradhar B, Phaneendra Kumar P and Sasanka C,2013 Traffic control using digital image processing International Journal of Advanced Electrical and Electronics Engineering, (IJAEEE)2.

[8] Jyoti Sharma and Savita Sivani 2014 LAN based traffic light control system with emergency service identification and density based control International Journal of Engineering Science Invention 3 01-03.

[9] Kastrinaki V, Zervakis M and Kalaitzakis K 2003 A survey of video processing techniques for traffic applications Image and Vision Computing 21 359-81.

[10] Jinglei Z, Zhengguang L and Univ T 2007A vision-based road surveillance system using improved background subtraction and region growing approach Eigth ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing 819-22.

**BLOCK DIAGRAM**

**TRAFFIC SIGNAL SYSTEM**

RF Receiver

Decoder

Microcontroller

Led Light for Signal

Power Supply

**EMERGENCY VEHICLES**

Key Button

Encoder

RF Receiver