**Title**

**ABSTRACT**

**CHAPTER 1**

**INTRODUCTION**

**BLOCK DIAGRAM**

**CHAPTER 2**

**2.1 EMBEDDED SYSTEM**

An embedded system can be defined as a computing device that does a specific focused job. Appliances such as the air-conditioner, VCD player, DVD player, printer, fax machine, mobile phone etc. are examples of embedded systems. Each of these appliances will have a processor and special hardware to meet the specific requirement of the application along with the embedded software that is executed by the processor for meeting that specific requirement. The embedded software is also called “firm ware”. The desktop/laptop computer is a general purpose computer. You can use it for a variety of applications such as playing games, *word* processing, accounting, software development and so on. In contrast, the software in the embedded systems is always fixed listed below:

· Embedded systems do a very specific task, they cannot be programmed to do different things. . Embedded systems have very limited resources, particularly the memory. Generally, they do not have secondary storage devices such as the CDROM or the floppy disk. Embedded systems have to work against some deadlines. A specific job has to be completed within a specific time. In some embedded systems, called real-time systems, the deadlines are stringent. Missing a deadline may cause a catastrophe-loss of life or damage to property. Embedded systems are constrained for power. As many embedded systems operate through a battery, the power consumption has to be very low.

· Some embedded systems have to operate in extreme environmental conditions such as very high temperatures and humidity.

**Application Areas**

Nearly 99 per cent of the processors manufactured end up in embedded systems. The embedded system market is one of the highest growth areas as these systems are used in very market segment- consumer electronics, office automation, industrial automation, biomedical engineering, wireless communication, data communication, telecommunications, transportation, military and so on.

**Consumer appliances**:

At home we use a number of embedded systems which include digital camera, digital diary, DVD player, electronic toys, microwave oven, remote controls for TV and air-conditioner, VCO player, video game consoles, video recorders etc. Today’s high-tech car has about 20 embedded systems for transmission control, engine spark control, air-conditioning, navigation etc. Even wristwatches are now becoming embedded systems. The palmtops are powerful embedded systems using which we can carry out many general-purpose tasks such as playing games and word processing.

**Office Automation:**

The office automation products using embedded systems are copying machine, fax machine, key telephone, modem, printer, scanner etc.

**Industrial Automation**:

Today a lot of industries use embedded systems for process control. These include pharmaceutical, cement, sugar, oil exploration, nuclear energy, electricity generation and transmission. The embedded systems for industrial use are designed to carry out specific tasks such as monitoring the temperature, pressure, humidity, voltage, current etc., and then take appropriate action based on the monitored levels to control other devices or to send information to a centralized monitoring station. In hazardous industrial environment, where human presence has to be avoided, robots are used, which are programmed to do specific jobs. The robots are now becoming very powerful and carry out many interesting and complicated tasks such as hardware assembly.

**Medical Electronics**:

Almost every medical equipment in the hospital is an embedded system. These equipments include diagnostic aids such as ECG, EEG, blood pressure measuring devices, X-ray scanners; equipment used in blood analysis, radiation, colonoscopy, endoscopy etc. Developments in medical electronics have paved way for more accurate diagnosis of diseases.

**Computer Networking**:

Computer networking products such as bridges, routers, Integrated Services Digital Networks (ISDN), Asynchronous Transfer Mode (ATM), X.25 and frame relay switches are embedded systems which implement the necessary data communication protocols. For example, a router interconnects two networks. The two networks may be running different protocol stacks. The router’s function is to obtain the data packets from incoming pores, analyze the packets and send them towards the destination after doing necessary protocol conversion. Most networking equipments, other than the end systems (desktop computers) we use to access the networks, are embedded systems.

**Telecommunications**:

In the field of telecommunications, the embedded systems can be categorized as subscriber terminals and network equipment. The subscriber terminals such as key telephones, ISDN phones, terminal adapters, web cameras are embedded systems. The network equipment includes multiplexers, multiple access systems, Packet Assemblers Dissemblers (PADs), sate11ite modems etc. IP phone, IP gateway, IP gatekeeper etc. are the latest embedded systems that provide very low-cost voice communication over the Internet.

**Wireless Technologies**:

Advances in mobile communications are paving way for many interesting applications using embedded systems. The mobile phone is one of the marvels of the last decade of the 20’h century. It is a very powerful embedded system that provides voice communication while we are on the move. The Personal Digital Assistants and the palmtops can now be used to access multimedia service over the Internet. Mobile communication infrastructure such as base station controllers, mobile switching centers are also powerful embedded systems.

**Insemination:**

Testing and measurement are the fundamental requirements in all scientific and engineering activities. The measuring equipment we use in laboratories to measure parameters such as weight, temperature, pressure, humidity, voltage, current etc. are all embedded systems. Test equipment such as oscilloscope, spectrum analyzer, logic analyzer, protocol analyzer, radio communication test set etc. are embedded systems built around powerful processors. Thank to miniaturization, the test and measuring equipment are now becoming portable facilitating easy testing and measurement in the field by field-personnel.

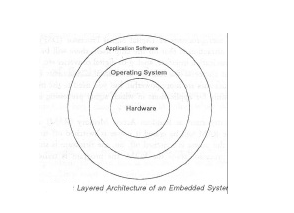
**Security:**

Security of persons and information has always been a major issue. We need to protect our homes and offices; and also the information we transmit and store. Developing embedded systems for security applications is one of the most lucrative businesses nowadays. Security devices at homes, offices, airports etc. for authentication and verification are embedded systems. Encryption devices are nearly 99 per cent of the processors that are manufactured end up in~ embedded systems. Embedded systems find applications in every industrial segment- consumer electronics, transportation, avionics, biomedical engineering, manufacturing, process control and industrial automation, data communication, telecommunication, defense, security etc. Used to encrypt the data/voice being transmitted on communication links such as telephone lines. Biometric systems using fingerprint and face recognition are now being extensively used for user authentication in banking applications as well as for access control in high security buildings.

**Finance:**

Financial dealing through cash and cheques are now slowly paving way for transactions using smart cards and ATM (Automatic Teller Machine, also expanded as Any Time Money) machines. Smart card, of the size of a credit card, has a small micro-controller and memory; and it interacts with the smart card reader! ATM machine and acts as an electronic wallet. Smart card technology has the capability of ushering in a cashless society. Well, the list goes on. It is no exaggeration to say that eyes wherever you go, you can see, or at least feel, the work of an embedded system.

**Overview of Embedded System Architecture**

Every embedded system consists of custom-built hardware built around a Central Processing Unit (CPU). This hardware also contains memory chips onto which the software is loaded. The software residing on the memory chip is also called the ‘firmware’. The embedded system architecture can be represented as a layered architecture as shown in Fig. The operating system runs above the hardware, and the application software runs above the operating system. The same architecture is applicable to any computer including a desktop computer. However, there are significant differences. It is not compulsory to have an operating system in every embedded system. For small appliances such as remote control units, air conditioners, toys etc., there is no need *for* an operating system and you can write only the software specific to that application. For applications involving complex processing, it is advisable to have an operating system. In such a case, you need to integrate the application software with the operating system and then transfer the entire software on to the memory chip. Once the software is transferred to the memory chip, the software will continue to run *for* a long time you don’t need to reload new software.

Now, let us see the details of the various building blocks of the hardware of an embedded system. As shown in Fig. the building blocks are;

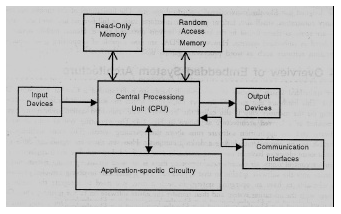
· Central Processing Unit (CPU)

· Memory (Read-only Memory and Random Access Memory)

· Input Devices

· Output devices

· Communication interfaces

· Application-specific circuitry

**Central Processing Unit (CPU):**

The Central Processing Unit (processor, in short) can be any of the following: microcontroller, microprocessor or Digital Signal Processor (DSP). A micro-controller is a low-cost processor. Its main attraction is that on the chip itself, there will be many other components such as memory, serial communication interface, analog-to digital converter etc. So, for small applications, a micro-controller is the best choice as the number of external components required will be very less. On the other hand, microprocessors are more powerful, but you need to use many external components with them. D5P is used mainly for applications in which signal processing is involved such as audio and video processing.

**Memory:**

The memory is categorized as Random Access 11emory (RAM) and Read Only Memory (ROM). The contents of the RAM will be erased if power is switched off to the chip, whereas ROM retains the contents even if the power is switched off. So, the firmware is stored in the ROM. When power is switched on, the processor reads the ROM; the program is program is executed.

**Input Devices**:

Unlike the desktops, the input devices to an embedded system have very limited capability. There will be no keyboard or a mouse, and hence interacting with the embedded system is no easy task. Many embedded systems will have a small keypad-you press one key to give a specific command. A keypad may be used to input only the digits. Many embedded systems used in process control do not have any input device *for* user interaction; they take inputs *from* sensors or transducers 1’fnd produce electrical signals that are in turn fed to other systems.

**Output Devices**:

The output devices of the embedded systems also have very limited capability. Some embedded systems will have a *few* Light Emitting Diodes (LEDs) *to* indicate the health status of the system modules, or *for* visual indication of alarms. A small Liquid Crystal Display (LCD) may also be used to display *some* important parameters.

**Communication Interfaces**:

The embedded systems may need to, interact with other embedded systems at they may have to transmit data to a desktop. To facilitate this, the embedded systems are provided with one or a *few* communication interfaces such as RS232, RS422, RS485, Universal Serial Bus (USB), IEEE 1394, Ethernet etc.

**Application-Specific Circuitry**:

Sensors, transducers, special processing and control circuitry may be required fat an embedded system, depending on its application. This circuitry interacts with the processor to carry out the necessary work. The entire hardware has to be given power supply either through the 230 volts main supply or through a battery. The hardware has to design in such a way that the power consumption is minimized.

**CHAPTER 3**

**HARDWARE SPECIFICATIONS**

**POWER SUPPLY**

The power supply section is the section which provide +5V for the components to work. IC LM7805 is used for providing a constant power of +5V.

The ac voltage, typically 220V, is connected to a transformer, which steps down that ac voltage down to the level of the desired dc output. A diode rectifier then provides a full-wave rectified voltage that is initially filtered by a simple capacitor filter to produce a dc voltage. This resulting dc voltage usually has some ripple or ac voltage variation.

A regulator circuit removes the ripples and also retains the same dc value even if the input dc voltage varies, or the load connected to the output dc voltage changes. This voltage regulation is usually obtained using one of the popular voltage regulator IC units.

****

Fig: Block Diagram of Power Supply

**3.1.1 TRANSFORMER**

Transformers convert AC electricity from one voltage to another with little loss of power. Transformers work only with AC and this is one of the reasons why mains electricity is AC.

Step-up transformers increase voltage, step-down transformers reduce voltage. Most power supplies use a step-down transformer to reduce the dangerously high mains voltage (230V in India) to a safer low voltage.

The input coil is called the primary and the output coil is called the secondary. There is no electrical connection between the two coils; instead they are linked by an alternating magnetic field created in the soft-iron core of the transformer. Transformers waste very little power so the power out is (almost) equal to the power in. Note that as voltage is stepped down current is stepped up.

The transformer will step down the power supply voltage (0-230V) to (0- 6V) level. Then the secondary of the potential transformer will be connected to the bridge rectifier, which is constructed with the help of PN junction diodes. The advantages of using bridge rectifier are it will give peak voltage output as DC.

**3.1.2 RECTIFIER**

There are several ways of connecting diodes to make a rectifier to convert AC to DC. The bridge rectifier is the most important and it produces full-wave varying DC. A full-wave rectifier can also be made from just two diodes if a centre-tap transformer is used, but this method is rarely used now that diodes are cheaper. A single diode can be used as a rectifier but it only uses the positive (+) parts of the AC wave to produce half-wave varying DC

**3.1.3 BRIDGE RECTIFIER**

When four diodes are connected as shown in figure, the circuit is called as bridge rectifier. The input to the circuit is applied to the diagonally opposite corners of the network, and the output is taken from the remaining two corners. Let us assume that the transformer is working properly and there is a positive potential, at point A and a negative potential at point B. the positive potential at point A will forward bias D3 and reverse bias D4.



Fig: Bridge Rectifier

The negative potential at point B will forward bias D1 and reverse D2. At this time D3 and D1 are forward biased and will allow current flow to pass through them; D4 and D2 are reverse biased and will block current flow.

One advantage of a bridge rectifier over a conventional full-wave rectifier is that with a given transformer the bridge rectifier produces a voltage output that is nearly twice that of the conventional full-wave circuit.

i. The main advantage of this bridge circuit is that it does not require a special centre tapped transformer, thereby reducing its size and cost.

ii. The single secondary winding is connected to one side of the diode bridge network and the load to the other side as shown below.

iii. The result is still a pulsating direct current but with double the frequency.

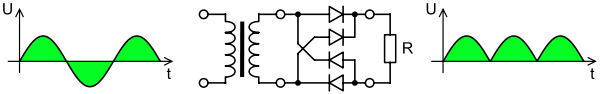


Fig: Output Waveform of DC

**3.1.4 SMOOTHING**

Smoothing is performed by a large value electrolytic capacitor connected across the DC supply to act as a reservoir, supplying current to the output when the varying DC voltage from the rectifier is falling. The capacitor charges quickly near the peak of the varying DC, and then discharges as it supplies current to the output.

**3.1.5 VOLTAGE REGULATORS**

Voltage regulators comprise a class of widely used ICs. Regulator IC units contain the circuitry for reference source, comparator amplifier, control device, and overload protection all in a single IC. IC units provide regulation of either a fixed positive voltage, a fixed negative voltage, or an adjustably set voltage. The regulators can be selected for operation with load currents from hundreds of milli amperes to tens of amperes, corresponding to power ratings from milli watts to tens of watts.

A fixed three-terminal voltage regulator has an unregulated dc input voltage, Vi, applied to one input terminal, a regulated dc output voltage, Vo, from a second terminal, with the third terminal connected to ground.

The series 78 regulators provide fixed positive regulated voltages from 5 to 24 volts. Similarly, the series 79 regulators provide fixed negative regulated voltages from 5 to 24 volts. Voltage regulator ICs are available with fixed (typically 5, 12 and 15V) or variable output voltages. They are also rated by the maximum current they can pass. Negative voltage regulators are available, mainly for use in dual supplies. Most regulators include some automatic protection from excessive current ('overload protection') and overheating ('thermal protection').

Many of the fixed voltage regulator ICs has 3 leads and look like power transistors, such as the 7805 +5V 1Amp regulator. They include a hole for attaching a heat sink if necessary.

****

Fig: Regulator

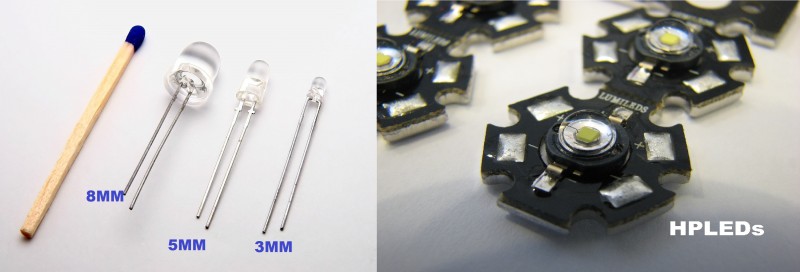
****

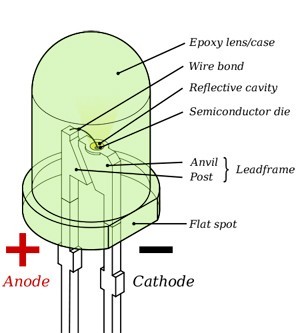
Fig: Circuit Diagram of Power Supply

**LIGHT EMITTING DIODE(LED)**

LED is abbreviation of Light Emitting Diode. It’s nothing, but just a combination of semiconductors which emits light when current pass through it . Over the years, semiconductor technology has advanced to bigger heights, Light Emitting Devices have also been a part of this revolution and as a result, Now we have LED’s which give better illumination with low power consumption.

**Types of LED**– There are many types of LEDs available in the market.. As you can see on above pic there is different LEDs according to our requirement and there has been many other are too available depending upon different parameters . And LEDs are choose according to parameters like space required by it, size, intensity, colors, etc. Typical LEDs are in size of 3mm, 5mm and 8mm. Nowadays HPLEDs(high power LEDs) are running in market which emits higher luminous intensity. High power LED’s has very high heat dissipation so LED’s need to mounted along with a cooling system known as heat sink.





**Operating parameters & circuit symbol**– Above figures show basic elements inside the LED and circuit symbol which helps in interfacing LED with 8051. Typical current ratings ranges from around 1 mA to above 20 mA and voltage is at about colors.

* 1.9 to 2.1 V for red, orange and yellow,
* 3.0 to 3.4 V for green and blue,
* 2.9 to 4.2 V for violet, pink, purple and white.
* 5 V and 12 V LEDs are incorporate a suitable [series](http://en.wikipedia.org/wiki/Series_and_parallel_circuits#Resistors) [resistor](http://en.wikipedia.org/wiki/Resistor) for direct connection to a 5 V or 12 V supply.

**Applications-**LED is everywhere because it’s an indicating component used in many areas. Just look around, if u can’t find even single LED, you are not on earth.

**MICROCONTROLLER**

A Microcontroller (or MCU) is a [computer](file:///E:\wiki\Computer)-on-a-[chip](file:///E:\wiki\Integrated_circuit) used to control [electronic](file:///E:\wiki\Electronics) [devices](file:///E:\wiki\Devices). It is a type of [microprocessor](file:///E:\wiki\Microprocessor) emphasizing self-sufficiency and cost-effectiveness, in contrast to a general-purpose microprocessor (the kind used in a [PC](file:///E:\wiki\Personal_computer)). A typical microcontroller contains all the [memory](file:///E:\wiki\Memory) and [interfaces](file:///E:\wiki\Interface_(computer_science)) needed for a simple application, whereas a general purpose microprocessor requires additional chips to provide these functions.

A microcontroller is a single [integrated circuit](file:///E:\wiki\Integrated_circuit) with the following key features:

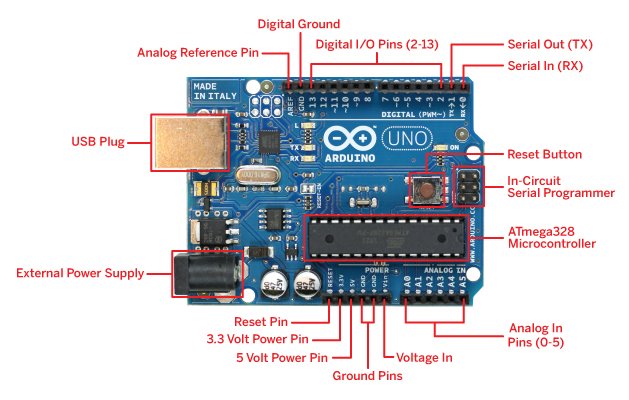
1. [central processing unit](file:///E:\wiki\Central_processing_unit) - ranging from small and simple 8-[bit](file:///E:\wiki\Bit) processors to sophisticated 32- or 64-bit processors
2. [input/output](file:///E:\wiki\Input\output) [interfaces](file:///E:\wiki\Network_interface) such as [serial ports](file:///E:\wiki\Serial_port)
3. [RAM](file:///E:\wiki\RAM) for data storage
4. [ROM](file:///E:\wiki\Read-only_Memory), [EEPROM](file:///E:\wiki\EEPROM) or [Flash memory](file:///E:\wiki\Flash_memory) for [program](file:///E:\wiki\Computer_program) storage
5. [clock generator](file:///E:\wiki\Clock_generator) - often an oscillator for a quartz timing crystal, resonator or [RC](file:///E:\wiki\RC_circuit) circuit

Microcontrollers are inside many kinds of [electronic equipment](file:///E:\wiki\Electronic_equipment) (see [embedded system](file:///E:\wiki\Embedded_system)). They are the vast majority of all processor chips sold. Over 50% are "simple" controllers, and another 20% are more specialized [digital signal processors (DSPs)](file:///E:\wiki\Digital_signal_processor) (ref?). A typical home in a [developed country](file:///E:\wiki\Developed_country) is likely to have only one or two general-purpose microprocessors but somewhere between one and two dozen microcontrollers. A typical mid range vehicle has as many as 50 or more microcontrollers. They can also be found in almost any [electrical](file:///E:\wiki\Electrical) [device](file:///E:\wiki\Tool): [washing machines](file:///E:\wiki\Washing_machine), [microwave ovens](file:///E:\wiki\Microwave_oven), [telephones](file:///E:\wiki\Telephone) etc.

**Arduino Uno Board**

Arduino/Genuino Uno is a microcontroller board based on the ATmega328P ([datasheet](http://www.atmel.com/Images/doc8161.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 worring 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.



Arduino Board

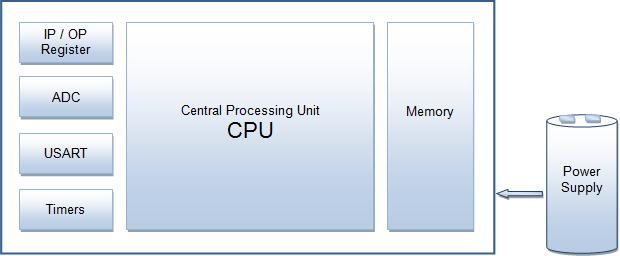
**ATMEGA 328P FEATURES**

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

**DESCRIPTION**

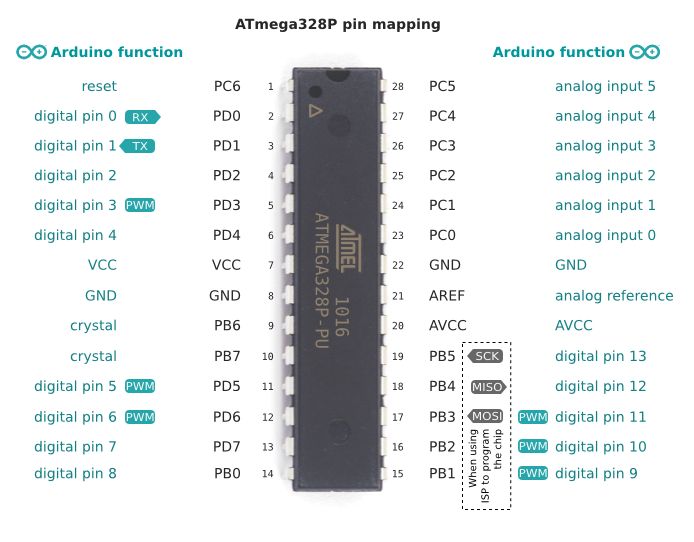
The high-performance Atmel picoPower 8-bit AVR RISC-based microcontroller combines 32KB ISP flash memory with read-while-write capabilities, 1024B EEPROM, 2KB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible timer/counters with compare modes, internal and external interrupts, serial programmable USART, a byte-oriented 2-wire serial interface, SPI serial port, a 6-channel 10-bit A/D converter (8-channels in TQFP and QFN/MLF packages), programmable watchdog timer with internal oscillator, and five software selectable power saving modes. The device operates between 1.8-5.5 volts.

By executing powerful instructions in a single clock cycle, the device achieves throughputs approaching 1 MIPS per MHz, balancing power consumption and processing speed.



Block Diagram

**PIN DIAGRAM**



Pin Diagram

**PIN DESCRIPTION**

**VCC**

Digital supply voltage.

**GND**

Ground.

**Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2**

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tristated when a reset condition becomes active, even if the clock is not running. Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7...6 is used as TOSC2...1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set. The various special features of Port B are elaborated in ”Alternate Functions of Port B” on page 82 and ”System Clock and Clock Options”.

**Port C (PC5:0)**

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5...0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tristated when a reset condition becomes active, even if the clock is not running.

**PC6/RESET**

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in Table 29-11 on page 305. Shorter pulses are not guaranteed to generate a Reset.

**Port D (PD7:0)**

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tristated when a reset condition becomes active, even if the clock is not running.

**AVCC**

AVCC is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter.

Note that PC6...4 use digital supply voltage, VCC.

**AREF**

AREF is the analog reference pin for the A/D Converter.

ADC7:6 (TQFP and QFN/MLF Package Only)

In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

Each of the 14 digital pins on the Uno can be used as an input or output, using [pinMode()](https://www.arduino.cc/en/Reference/PinMode),[digitalWrite()](https://www.arduino.cc/en/Reference/DigitalWrite), and [digitalRead()](https://www.arduino.cc/en/Reference/DigitalRead) functions. They operate at 5 volts. Each pin can provide or receive 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50k ohm. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller.

**In addition, some pins have specialized functions:**

Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.

PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.

LED: 13. There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. There are a couple of other pins on the board:

AREF. Reference voltage for the analog inputs. Used with analogReference().

Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

**PROGRAMMING**

The Arduino/Genuino Uno can be programmed with the ([Arduino Software](https://www.arduino.cc/en/Main/Software) (IDE)). Select "Arduino/Genuino Uno from the Tools > Board menu (according to the microcontroller on your board). For details, see the [reference](https://www.arduino.cc/en/Reference/HomePage) and [tutorials](https://www.arduino.cc/en/Tutorial/HomePage).

The ATmega328 on the Arduino/Genuino Uno comes preprogrammed with a [bootloader](https://www.arduino.cc/en/Hacking/Bootloader?from=Tutorial.Bootloader) that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol ([reference](http://www.atmel.com/Images/doc2525.pdf), [C header files](http://www.atmel.com/dyn/resources/prod_documents/avr061.zip)).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header using [Arduino ISP](https://www.arduino.cc/en/Main/ArduinoISP) or similar; see [these instructions](https://www.arduino.cc/en/Hacking/Programmer) for details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available in the Arduino repository. The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

* On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then rese ing the 8U2.
* On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use [Atmel's FLIP software](http://www.atmel.com/products/microcontrollers/default.aspx)(Windows) or the [DFU programmer](http://dfu-programmer.github.io/) (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See [this user-contributed tutorial](http://forum.arduino.cc/index.php/topic,111.0.html) for more information.

**Differences with other boards**

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

**Communication**

Arduino/Genuino Uno has a number of facilities for communicating with a computer, another Arduino/Genuino board, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The 16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, [on Windows, a .inf file is required](https://www.arduino.cc/en/Guide/Windows#toc4). The Arduino Software (IDE) includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A [SoftwareSerial library](https://www.arduino.cc/en/Reference/SoftwareSerial) allows serial communication on any of the Uno's digital pins.

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino Software (IDE) includes a Wire library to simplify use of the I2C bus; see the [documentation](https://www.arduino.cc/en/Reference/Wire) for details. For SPI communication, use the [SPI library](https://www.arduino.cc/en/Reference/SPI).

**Automatic (Software) Reset**

Rather than requiring a physical press of the reset button before an upload, the Arduino/Genuino Uno board is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino Software (IDE) uses this capability to allow you to upload code by simply pressing the upload button in the interface toolbar. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Uno board contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line.

**BUZZER**

A [buzzer](http://www.microbuzzer.com/) or beeper is a signaling device, usually electronic, typically used in automobiles, house hold appliances such as a microwave oven, or game shows.

It most commonly consists of a number of switches or sensors connected to a control unit that determines if and which button was pushed or a preset time has lapsed, and usually illuminates a light on the appropriate button or control panel, and sounds a warning in the form of a continuous or intermittent buzzing or beeping sound. Initially this device was based on an electromechanical system which was identical to an electric bell without the metal gong (which makes the ringing noise). Often these units were anchored to a wall or ceiling and used the ceiling or wall as a sounding board. Another implementation with some AC-connected devices was to implement a circuit to make the AC current into a noise loud enough to drive a loudspeaker and hook this circuit up to a cheap 8-ohm speaker. Nowadays, it is more popular to use a ceramic-based piezoelectric sounder like a Sonalert which makes a high-pitched tone. Usually these were hooked up to “driver” circuits which varied the pitch of the sound or pulsed the sound on and off.

In game shows it is also known as a “lockout system,” because when one person signals (“buzzes in”), all others are locked out from signalling. Several game shows have large buzzer buttons which are identified as “plungers”.



Fig. Buzzer

**USES**

* [Annunciator panels](http://en.wikipedia.org/wiki/Annunciator_panel)
* Electronic [metronomes](http://en.wikipedia.org/wiki/Metronome)
* [Game shows](http://en.wikipedia.org/wiki/Game_show)
* [Microwave ovens](http://en.wikipedia.org/wiki/Microwave_oven) and other [household appliances](http://en.wikipedia.org/wiki/Major_appliance)
* [Sporting](http://en.wikipedia.org/wiki/Sport) events such as [basketball](http://en.wikipedia.org/wiki/Basketball) games
* Electrical [alarms](http://en.wikipedia.org/wiki/Alarms)

**LIQUID CRYSTAL DISPLAY(LCD)**

LCD (Liquid Crystal Display) screen is an electronic display module and find a wide range of applications. A 16x2 LCD display is very basic module and is very commonly used in various devices and circuits. These modules are preferred over [seven segments](http://www.engineersgarage.com/content/seven-segment-display) and other multi segment [LED](http://www.engineersgarage.com/content/led)s. The reasons being: LCDs are economical; easily programmable; have no limitation of displaying special & even [custom characters](http://www.engineersgarage.com/microcontroller/8051projects/create-custom-characters-LCD-AT89C51) (unlike in seven segments), [animations](http://www.engineersgarage.com/microcontroller/8051projects/display-custom-animations-LCD-AT89C51) and so on.

A **16x2 LCD** means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix. This LCD has two registers, namely, Command and Data.

The command register stores the command instructions given to the LCD. A command is an instruction given to LCD to do a predefined task like initializing it, clearing its screen, setting the cursor position, controlling display etc. The data register stores the data to be displayed on the LCD. The data is the ASCII value of the character to be displayed on the LCD.

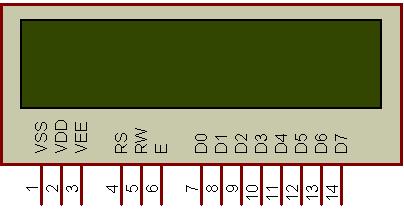


Fig. 16x2 LCD

**Introduction**

The most commonly used Character based LCDs are based on Hitachi's HD44780 controller or other which are compatible with HD44580.

**Pin Description**

Most LCDs with 1 controller has 14 Pins and LCDs with 2 controller has 16 Pins (two pins are extra in both for back-light LED connections). Pin description is shown in the table below.

**Pin Configuration table for a 16X2 LCD character display:-**

|  |  |  |
| --- | --- | --- |
| **Pin Number** | **Symbol** | **Function** |
| **1** | Vss | Ground Terminal |
| **2** | Vcc | Positive Supply |
| **3** | Vdd | Contrast adjustment |
| **4** | RS | Register Select; 0→Instruction Register, 1→Data Register |
| **5** | R/W | Read/write Signal; 1→Read, 0→ Write |
| **6** | E | Enable; Falling edge |
| **7** | DB0 | Bi-directional data bus, data transfer is performed once, thru DB0 to DB7, in the case of interface data length is 8-bits; and twice, through DB4 to DB7 in the case of interface data length is 4-bits. Upper four bits first then lower four bits. |
| **8** | DB1 |
| **9** | DB2 |
| **10** | DB3 |
| **11** | DB4 |
| **12** | DB5 |
| **13** | DB6 |
| **14** | DB7 |
| **15** | LED-(K) | Back light LED cathode terminal |
| **16** | LED+(A) | Back Light LED anode terminal |

Table Pin Description Of LCD

**Data/Signals/Execution of LCD**

Coming to data, signals and execution.

LCD accepts two types of signals, one is data, and another is control. These signals are recognized by the LCD module from status of the RS pin. Now data can be read also from the LCD display, by pulling the R/W pin high. As soon as the E pin is pulsed, LCD display reads data at the falling edge of the pulse and executes it, same for the case of transmission.

                LCD display takes a time of 39-43µS to place a character or execute a command. Except for clearing display and to seek cursor to home position it takes 1.53ms to 1.64ms. Any attempt to send any data before this interval may lead to failure to read data or execution of the current data in some devices. Some devices compensate the speed by storing the incoming data to some temporary registers.

**Instruction Register (IR) and Data Register (DR)**

There are two 8-bit registers in HD44780 controller Instruction and Data register. Instruction register corresponds to the register where you send commands to LCD e.g LCD shift command, LCD clear, LCD address etc. and Data register is used for storing data which is to be displayed on LCD. when send the enable signal of the LCD is asserted, the data on the pins is latched in to the data register and data is then moved automatically to the DDRAM and hence is displayed on the LCD. Data Register is not only used for sending data to DDRAM but also for CGRAM, the address where you want to send the data, is decided by the instruction you send to LCD. We will discuss more on LCD instruction set further in this tutorial.

**Commands and Instruction set**

Only the instruction register (IR) and the data register (DR) of the LCD can be controlled by the MCU. Before starting the internal operation of the LCD, control information is temporarily stored into these registers to allow interfacing with various MCUs, which operate at different speeds, or various peripheral control devices. The internal operation of the LCD is determined by signals sent from the MCU. These signals, which include register selection signal (RS), read/write signal (R/W), and the data bus (DB0 to DB7), make up the LCD instructions (Table 3). There are four categories of instructions that:

* Designate LCD functions, such as display format, data length, etc.
* Set internal RAM addresses
* Perform data transfer with internal RAM
* Perform miscellaneous functions

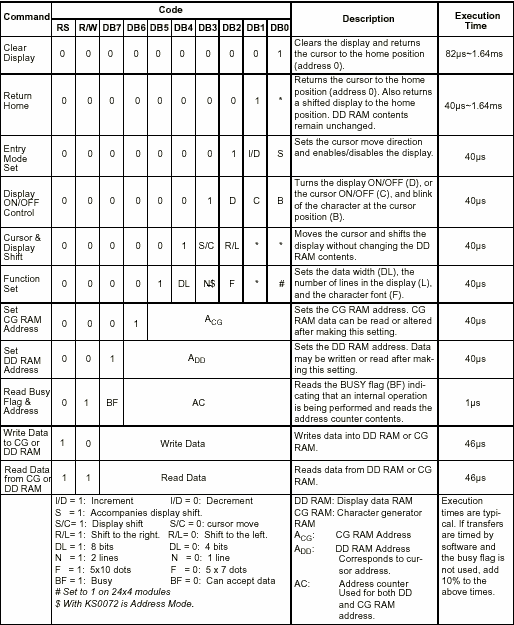


Table Showing various LCD Command Description

Although looking at the table you can make your own commands and test them. Below is a brief list of useful commands which are used frequently while working on the LCD.

**List Of Command**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Instruction** | **Hex** | **Decimal** |
| 1 | Function Set: 8-bit, 1 Line, 5x7 Dots | 0x30 | 48 |
| 2 | Function Set: 8-bit, 2 Line, 5x7 Dots | 0x38 | 56 |
| 3 | Function Set: 4-bit, 1 Line, 5x7 Dots | 0x20 | 32 |
| 4 | Function Set: 4-bit, 2 Line, 5x7 Dots | 0x28 | 40 |
| 5 | Entry Mode | 0x06 | 6 |
| 6 | Display off Cursor off (clearing display without clearing DDRAM content) | 0x08 | 8 |
| 7 | Display on Cursor on | 0x0E | 14 |
| 8 | Display on Cursor off | 0x0C | 12 |
| 9 | Display on Cursor blinking | 0x0F | 15 |
| 10 | Shift entire display left | 0x18 | 24 |
| 12 | Shift entire display right | 0x1C | 30 |
| 13 | Move cursor left by one character | 0x10 | 16 |
| 14 | Move cursor right by one character | 0x14 | 20 |
| 15 | Clear Display (also clear DDRAM content) | 0x01 | 1 |
| 16 | Set DDRAM address or coursor position on display | 0x80+add\* | 128+add\* |
| 17 | Set CGRAM address or set pointer to CGRAM location | 0x40+add\*\* | 64+add\*\* |

Table : Frequently Used Commands And Instructions For Lcd

\* DDRAM address given in LCD basics section see Figure 2,3,4  
\*\* CGRAM address from 0x00 to 0x3F, 0x00 to 0x07 for char1 and so on.

**Liquid crystal displays interfacing with Controller**

The LCD standard requires 3 control lines and 8 I/O lines for the data bus.

• **8 data pins D7:D0**

Bi-directional data/command pins.  
Alphanumeric characters are sent in ASCII format.

• **RS:  Register Select**

RS = 0 -> Command Register is selected  
RS = 1 -> Data Register is selected

• **R/W: Read or Write**

0 -> Write,  1 -> Read

• **E: Enable (Latch data)**

Used to latch the data present on the data pins.  
A high-to-low edge is needed to latch the data.

**Chapter**

**SENSORS**

**CHAPTER 4**

**SOFTWARE SPECIFICATION**

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.
* *Saveas...* Allows to save the current sketch with a different name.
* *PageSetup* 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 Sketch 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).

To write your own library.

**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.

For details on creating packages for third-party hardware, see the [Arduino IDE 1.5 3rd party Hardware specification](https://github.com/arduino/Arduino/wiki/Arduino-IDE-1.5-3rd-party-Hardware-specification).

**SERIAL MONITOR**

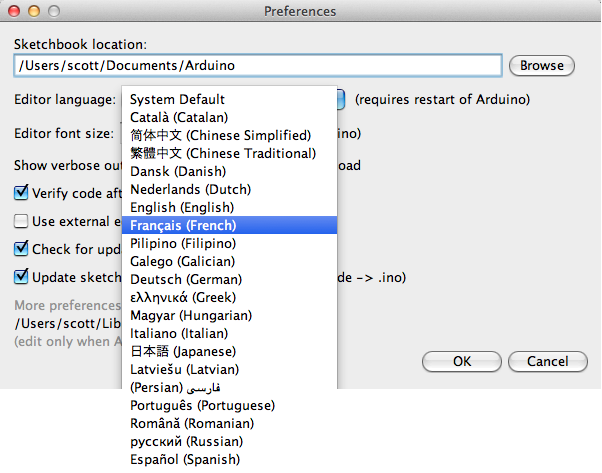
Displays serial data being sent from the Arduino or Genuino board (USB or serial board). To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down that matches the rate passed to Serial.begin in your sketch. Note that on Windows, Mac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

You can also talk to the board from Processing, Flash, MaxMSP, etc (see the [interfacing page](http://www.arduino.cc/playground/Main/Interfacing) for details).

**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.

* *Arduino Yùn*   
  An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
* *Arduino/Genuino Uno*   
  An ATmega328 running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
* *Arduino Diecimila or Duemilanove w/ ATmega168*   
  An ATmega168 running at 16 MHz with auto-reset.
* *Arduino Nano w/ ATmega328*   
  An ATmega328 running at 16 MHz with auto-reset. Has eight analog inputs.
* *Arduino/Genuino Mega 2560*   
  An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
* *Arduino Mega*   
  An ATmega1280 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
* *Arduino Mega ADK*   
  An ATmega2560 running at 16 MHz with auto-reset, 16 Analog In, 54 Digital I/O and 15 PWM.
* *Arduino Leonardo*   
  An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
* *Arduino/Genuino Micro*   
  An ATmega32u4 running at 16 MHz with auto-reset, 12 Analog In, 20 Digital I/O and 7 PWM.
* *Arduino Esplora*   
  An ATmega32u4 running at 16 MHz with auto-reset.
* *Arduino Mini w/ ATmega328*  
  An ATmega328 running at 16 MHz with auto-reset, 8 Analog In, 14 Digital I/O and 6 PWM.
* *Arduino Ethernet*   
  Equivalent to Arduino UNO with an Ethernet shield: An ATmega328 running at 16 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
* *Arduino Fio*   
  An ATmega328 running at 8 MHz with auto-reset. Equivalent to Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ATmega328, 6 Analog In, 14 Digital I/O and 6 PWM.
* *Arduino BT w/ ATmega328*   
  ATmega328 running at 16 MHz. The bootloader burned (4 KB) includes codes to initialize the on-board bluetooth module, 6 Analog In, 14 Digital I/O and 6 PWM..
* *LilyPad Arduino USB*   
  An ATmega32u4 running at 8 MHz with auto-reset, 4 Analog In, 9 Digital I/O and 4 PWM.
* *LilyPad Arduino*   
  An ATmega168 or ATmega132 running at 8 MHz with auto-reset, 6 Analog In, 14 Digital I/O and 6 PWM.
* *Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328*   
  An ATmega328 running at 16 MHz with auto-reset. Equivalent to Arduino Duemilanove or Nano w/ ATmega328; 6 Analog In, 14 Digital I/O and 6 PWM.
* *Arduino NG or older w/ ATmega168*   
  An ATmega168 running at 16 MHz *without* auto-reset. Compilation and upload is equivalent to Arduino Diecimila or Duemilanove w/ ATmega168, but the bootloader burned has a slower timeout (and blinks the pin 13 LED three times on reset); 6 Analog In, 14 Digital I/O and 6 PWM.
* *Arduino Robot Control*   
  An ATmega328 running at 16 MHz with auto-reset.
* *Arduino Robot MOTOR*   
  An ATmega328 running at 16 MHz with auto-reset.
* *Arduino Gemma*   
  An ATtiny85 running at 8 MHz with auto-reset, 1 Analog In, 3 Digital I/O and 2 PWM.

### THE COMPILATION PROCESS

The arduino code is actually just plain old c without all the header part (the includes and all). when you press the 'compile' button, the IDE saves the current file as arduino.c in the 'lib/build' directory then it calls a makefile contained in the 'lib' directory.

This makefile copies arduino.c as prog.c into 'lib/tmp' adding 'wiringlite.inc' as the beginning of it. this operation makes the arduino/wiring code into a proper c file (called prog.c).

After this, it copies all the files in the 'core' directory into 'lib/tmp'. these files are the implementation of the various arduino/wiring commands adding to these files adds commands to the language

The core files are supported by pascal stang's procyon avr-lib that is contained in the 'lib/avrlib' directory. At this point the code contained in lib/tmp is ready to be compiled with the c compiler contained in 'tools'. If the make operation is succesfull then you'll have prog.hex ready to be downloaded into the processor.

**CHAPTER 5**

**Working**

**CHAPTER 5**

**CIRCUIT DIAGRAM**

**Results**

**Add schematic here**

**CODE**

CONCLUSIONS

REFERENCES