Chapter 1
INTRODUCTION

The aim of home automation is to control home devices from a central control point. In this paper, we present the design and implementation of a low cost but yet flexible and secure internet based home automation system. The communication between the devices is wireless. The protocol between the units in the design is enhanced to be suitable for most of the appliances. The system is designed to be low cost and flexible with the increasing variety of devices to be controlled. In this paper presents the way to provide Ethernet internet connectivity to Arduino based embedded systems. This system uses Arduino to store the main application source code, web pages and TCP/IP stack which is a vital element of the system software. An Ethernet controller chip, ENC28J60 is used to handle the Ethernet communications and it is interfaced with the Rabbit Processor using SPI protocol. Configurations like IP address and other details are set using RS232 interface. The site can be viewed on any system with Internet/LAN connection by configuring the specific IP address and by giving User Login ID, password. There are several I/O pins available at the Rabbit Processor which is used to interface with sensors, LCD displays, Motors and relays for monitoring and controlling AC appliances. Nowadays, Internet has spread worldwide and most of the internet connections use Ethernet as media for data transfer. In industries or in home appliances, most of the time we need to monitor and control different parameters using microcontrollers, but the microcontroller doesn’t have the port of internet connection; So that we used the Rabbit processor which have Ethernet port exist on it. The popularity of home automation has been increasing greatly in recent years due to much higher affordability and simplicity through Smartphone and tablet connectivity. The concept of the "Internet of Things" has tied in closely with the popularization of home automation. Ethernet provides inexpensive, relatively high speed network access to individual users and low delay that can support many applications. This implementation is an attempt to connect an embedded device to an Ethernet. Using Ethernet based system we can control various home appliances from anywhere across the world.

1.1 Ethernet:

Ethernet is a family of computer networking technologies for Local Area Networks (LANs) and Metropolitan Area Networks (MANs). It was commercially introduced in 1980 and first standardized in 1983 as IEEE 802.3 and has since been refined to support higher bit rates and longer link distances. The primary alternative for contemporary LANs is not a wired standard, but instead a wireless LAN standardized as IEEE 802.11 and also known as Wi-Fi. Over the course of the last 20 years, Ethernet has become the dominant LAN (local area network) technology used throughout the world. As the computerized workplace has become more networks based, with the rise in E-mail and the Internet, Ethernet has become even more prevalent. Almost all companies nowadays use Ethernet to connect their computers to each other. Ethernet has been a relatively inexpensive, reasonably fast, and very popular LAN technology for several decades. Two individuals at Xerox PARC -- Bob Metcalfe and D.R. Boggs-- developed Ethernet beginning in 1972 and specifications based on this work appeared in IEEE802.3 in 1980.

IEEE 802.3, an Ethernet LAN typically uses coaxial cable or special grades of twisted pair wires. Ethernet is also used in wireless LANs. Ethernet uses the CSMA/CD access method to handle simultaneous demands.

1.2 IEEE Standards:

IEEE developed a set of network standards. They include:

- IEEE 802.1: Standards related to network management.
- IEEE 802.2: General standard for the data link layer in the OSI Reference Model. The IEEE divides this layer into two sub-layers -- the logical link control (LLC) layer and the media access control (MAC) layer. The MAC layer varies for different network types and is defined by standards IEEE 802.3 through IEEE 802.5.
- IEEE 802.3: Defines the MAC layer for bus networks that use CSMA/CD. This is the basis of the Ethernet standard.
- IEEE 802.4: Defines the MAC layer for bus networks that use a token-passing mechanism (token bus networks).
- IEEE 802.5: Defines the MAC layer for token-ring networks.
- IEEE 802.6: Standard for Metropolitan Area Networks (MANs).

1.3 Ethernet Protocol:

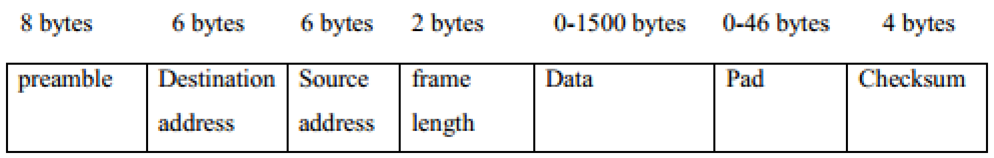
The Ethernet protocol is made up of a number of components, such as the structure of Ethernet frames, the Physical Layer and its MAC operation. The fundamental structure of the

Ethernet Protocol is its Frame Structure.

Frame Structure:

Information is sent around an Ethernet network in discreet messages known as frames.

The frame structure is quite simple, consisting of the following fields shown in fig 1.1:



8 bytes	6 bytes	6 bytes	2 bytes	0-1500 bytes	0-46 bytes	4 bytes
preamble	Destination address	Source address	frame length	Data	Pad	Checksum

Figure 1.1: Frame Structure of Ethernet

1. The Preamble - This consists of seven bytes, all of the form ("101 01010"). This allows the receiver's clock to be synchronized with the sender's.
2. The Start Frame Delimiter - This is a single byte ("10101011") which is used to indicate the start of a frame.
3. The Destination Address - This is the address of the intended recipient of the frame. The addresses in 802.3 use globally unique hardwired 48 bit addresses.
4. The Source Address - This is the address of the source, in the same form as above.
5. The Length - This is the length of the data in the Ethernet frame, which can be anything from 0 to 1500 bytes.
6. Data - This is the information being sent by the frame.
7. Pad - 802.3 frames must be at least 64 bytes long, so if the data is shorter than 46 bytes, the pad field must compensate. The reason for the minimum length lies with the collision detection mechanism. In CSMA/CD the sender must wait at least two times the maximum propagation delay before it knows that no collision has occurred. If a station sends a very short message, then it might release the ether without knowing that the frame has been corrupted. 802.3 sets an upper limit on the propagation delay and the minimum frame size is set at the amount of data which can be sent in twice this figure.
8. Checksum - This is used for error detection and recovery

1.4 Topology:

Topology is the shape of a local-area network (LAN) or other communications system. In other words, a topology describes pictorially the configuration or arrangement of a network, including its nodes and connecting lines. Topologies are either physical or

logical. Ethernet uses
topology to transfer the data. There are four principal topologies used in LANs.

a. Bus topology

b. Ring topology

c. Star topology

d. Tree topology

a. Bus topology:

All devices are connected to a central cable, called the bus or backbone. Bus networks are
relatively inexpensive and easy to install for small networks. Ethernet systems use a bus topology.

b. Ring topology:

All devices are connected to one another in the shape of a closed loop, so that each device
is connected directly to two other devices, one on either side of it. Ring topologies are relatively
expensive and difficult to install, but they offer high bandwidth and can span large distances.

c. Star topology:

All devices are connected to a central hub. Star networks are relatively easy to install and manage, but bottlenecks can occur because all data must pass through the hub.

d. Tree topology:

A tree topology combines characteristics of linear bus and star topologies. It consists of groups of star-configured workstations connected to a linear bus backbone cable. These topologies can also be mixed. For example, a bus-star network consists of a high bandwidth bus,
called the backbone, which connects collections of slower-bandwidth star segments.

Chapter 2

HARDWARE REQUIREMENTS

2.1 Block Diagram of IoT Based Home Automation using Ethernet:

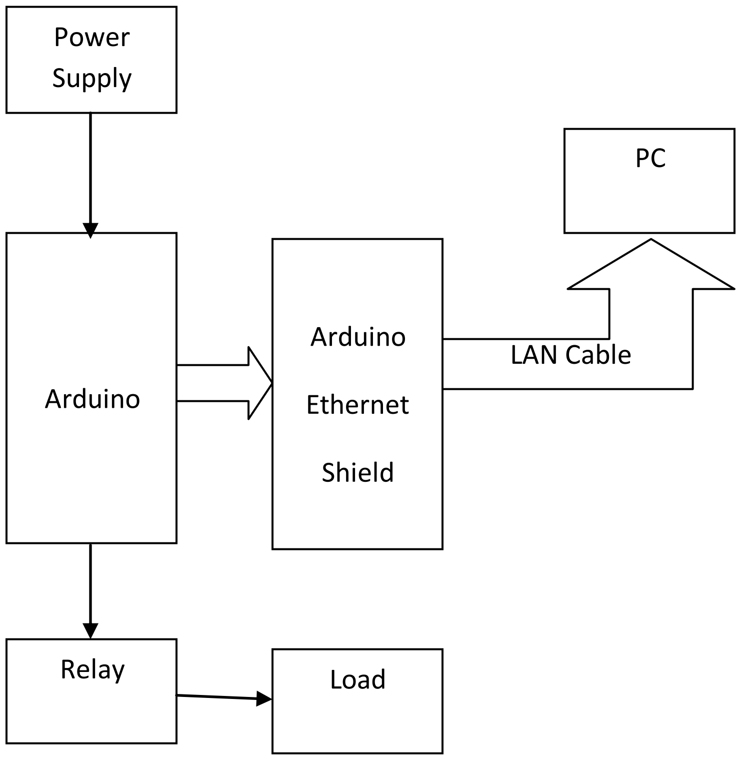


Fig: 2.1 Block diagram

2.2 HARDWARE REQUIREMENTS:

2.2.1: ARDUINO

The Arduino microcontroller is an easy to use yet powerful single board computer that has gained considerable traction in the hobby and professional market. The Arduino is open-source, which means hardware is reasonably priced and development software is free. This guide is for students in ME 2011, or students anywhere who are confronting the Arduino for the first time. For advanced Arduino users, prowl the web; there are lots of resources.

This is what the Arduino board looks like.

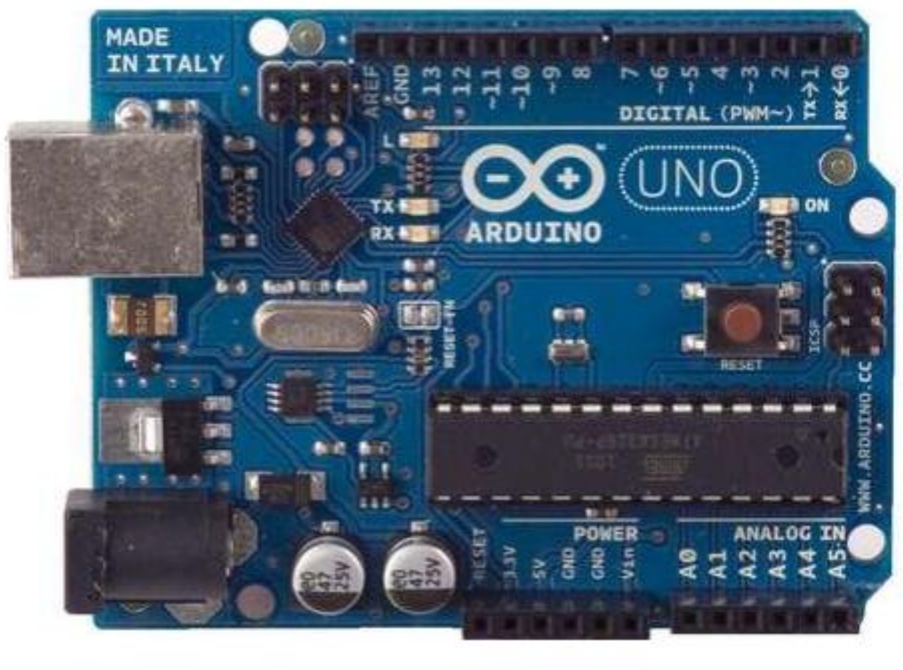


Fig 2.2.1.1 ARDUINO board

The Arduino programming language is a simplified version of C/C++. If you know C, programming the Arduino will be familiar. If you do not know C, no need to worry as only a few commands are needed to perform useful functions.

An important feature of the Arduino is that you can create a control program on the host PC, download it to the Arduino and it will run automatically. Remove the USB cable connection to the PC, and the program will still run from the top each time you push the reset button. Remove the battery and put the Arduino board in a closet for six months. When you reconnect the battery, the last program you stored will run. This means that you connect the board to the host PC to develop and debug your program, but once that is done, you no longer need the PC to run the program.

Arduino Hardware

The power of the Arduino is not its ability to crunch code, but rather its ability to interact with the outside world through its input-output (I/O) pins. The Arduino has 14 digital I/O pins labeled 0 to 13 that can be used to turn motors and lights on and off and read the state of switches. Each digital pin can sink or source about 40 mA of current. This is more than adequate for interfacing to most devices, but does mean that interface circuits are needed to control devices other than simple LED's. In other words, you cannot run a motor directly using the current available from an Arduino pin, but rather must have the pin drive an interface circuit that in turn drives the motor. A later section of this document shows how to interface to a small motor. To interact with the outside world, the program sets digital pins to a high or low value using C code instructions, which corresponds to +5 V or 0 V at the pin. The pin is connected to external interface electronics and then to the device being switched on and off. The sequence of events is shown in this figure.

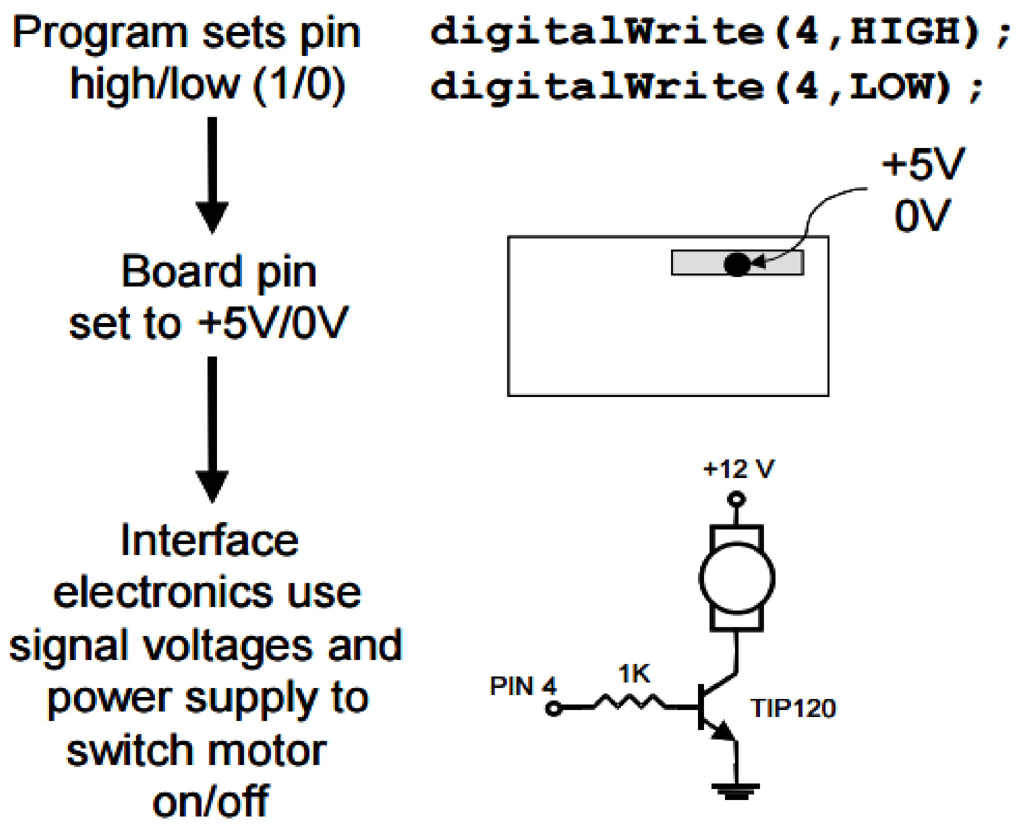

Program sets pin high/low (1/0)
digitalWrite(4,HIGH);
digitalWrite(4,LOW);
+5V
0V
Board pin set to +5V/0V
+12 V
Interface electronics use signal voltages and power supply to switch motor on/off
PIN 4
1K
TIP120

Fig 2.2.1.2 Algorithm for writing into ARDUINO

To determine the state of switches and other sensors, the Arduino is able to read the voltage value applied to its pins as a binary number. The interface circuitry translates the sensor signal into a 0 or +5 V signal applied to the digital I/O pin. Through a program command, the Ardiomp interrogates the state of the pin. If the pin is at 0 V, the program will read it as a 0 or LOW. If it is at +5 V, the program will read it as a 1 or HIGH. If more than +5 V is applied, you may blow out your board, so be careful. The sequence of events to read a pin is shown in this figure.

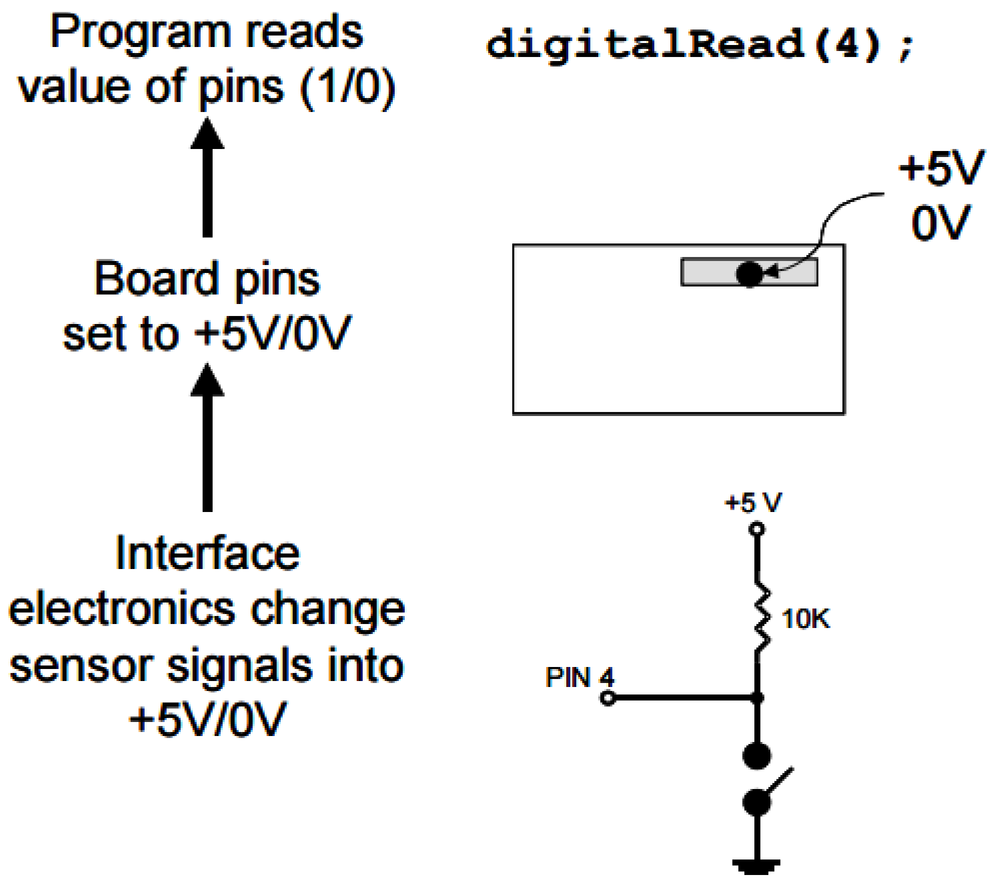

Program reads value of pins (1/0)
digitalRead(4);
Board pins set to +5V/0V
+5V
0V
Interface electronics change sensor signals into +5V/0V
+5 V
10K
PIN 4

Fig 2.2.1.3 Algorithm for reading from ARDUINO

Interacting with the world has two sides. First, the designer must create electronic interface circuits that allow motors and other devices to be controlled by a low (1-10 mA) current signal that switches between 0 and 5 V, and other circuits that convert sensor readings into a switched 0 or 5 V signal. Second, the designer must write a program using the set of Arduino commands that set and read the I/O pins. Examples of both can be found in the Arduino resources section of the ME2011 web site.

2.2.2. Atmega328p features:

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture

  – 131 Powerful Instructions

  – Most Single Clock Cycle Execution

  – 32 x 8 General Purpose Working Registers

  – Fully Static Operation

  – Up to 20 MIPS Throughput at 20 MHz

  – On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments

  – 4/8/16/32K Bytes of In-System Self-Programmable Flash progam memory (ATmega48PA/88PA/168PA/328P)

  – 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)

  – 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)

  – Write/Erase Cycles: 10,000 Flash/100,000 EEPROM

  – Data retention: 20 years at 85°C/100 years at 25°C(1)

  – Optional Boot Code Section with Independent Lock Bits In-System Programming by On-chip Boot Program True Read-While-Write Operation

  – Programming Lock for Software Security
- Peripheral Features

  – Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode

  – One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode

  – Real Time Counter with Separate Oscillator

  – Six PWM Channels – 8-channel 10-bit ADC in TQFP and QFN/MLF package Temperature Measurement – 6-channel 10-bit ADC in PDIP Package Temperature Measurement

  – Programmable Serial USART

  – Master/Slave SPI Serial Interface

  – Byte-oriented 2-wire Serial Interface (Philips I2 C compatible)

– Programmable Watchdog Timer with Separate On-chip Oscillator

– On-chip Analog Comparator

– Interrupt and Wake-up on Pin Change

- Special Microcontroller Features

  – Power-on Reset and Programmable Brown-out Detection

  – Internal Calibrated Oscillator

  – External and Internal Interrupt Sources

  – Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby

- I/O and Packages

  – 23 Programmable I/O Lines

  – 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF

- Operating Voltage:

  – 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P

- Temperature Range:

  – -40°C to 85°C

- Speed Grade:

  – 0 - 20 MHz @ 1.8 - 5.5V

- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:

  – Active Mode: 0.2 mA

  – Power-down Mode: 0.1 µA

  – Power-save Mode: 0.75 µA (Including 32 kHz RTC)

2.3: PIN CONFIGURATIONS

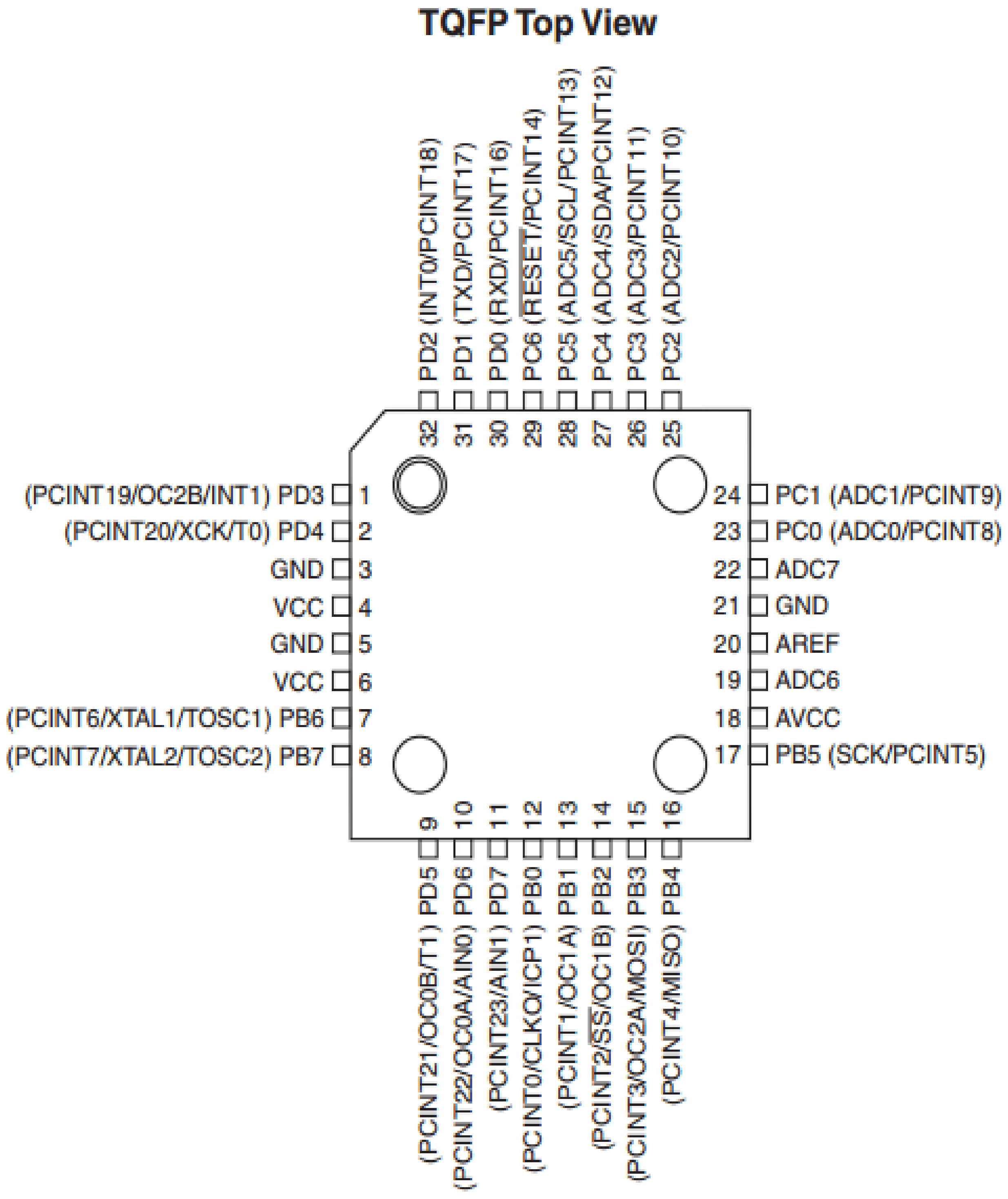


Fig 2.3.1: Pin Configuration of TQFP

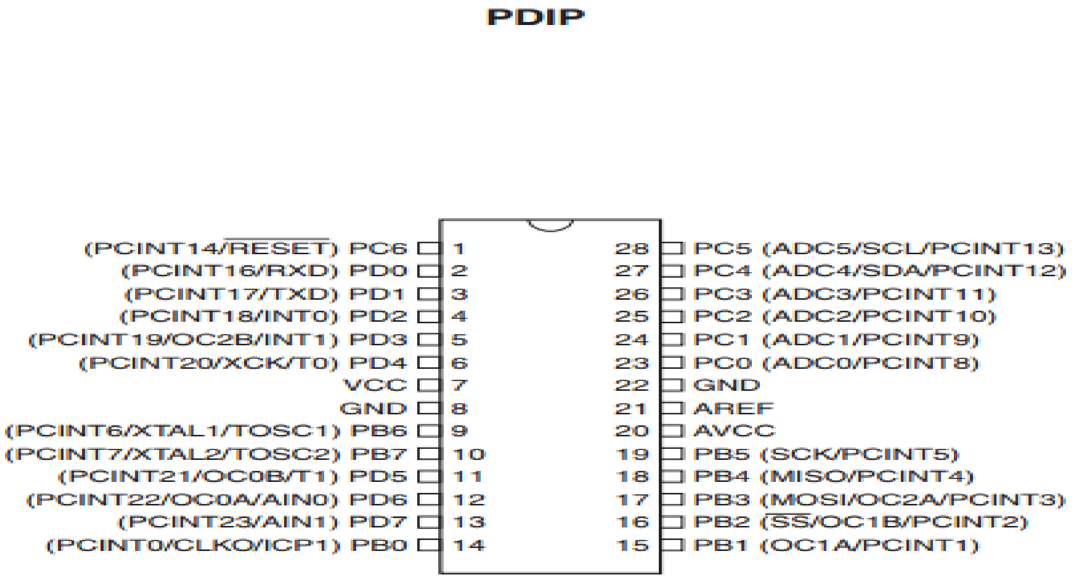


Fig 2.3.2: PDIP pin configuration

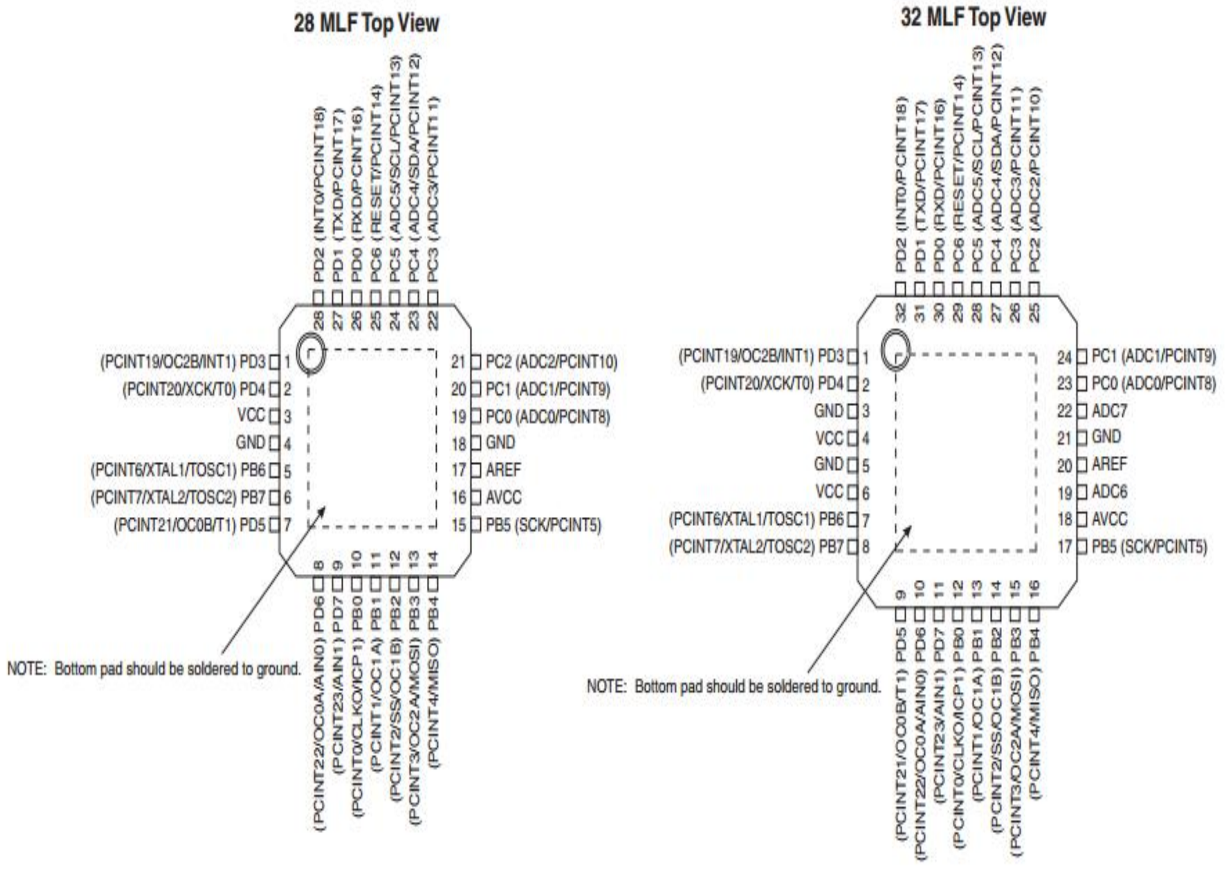


Fig:2.3.3: Pin configurations of MLF

Pin Descriptions

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 tri-stated 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 76 and ”System Clock and Clock Options” on page 26.

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 tri-stated 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 28-3 on page 308. Shorter pulses are not guaranteed to generate a Reset. The various special features of Port C are elaborated in ”Alternate Functions of Port C” on page 79.

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 tri-stated when a reset condition becomes active, even if the clock is not running. The various special features of Port D are elaborated in ”Alternate Functions of Port D” on page 82.

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.

OVERVIEW

The ATmega48PA/88PA/168PA/328P is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega48PA/88PA/168PA/328P achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

2.4: BLOCK DIAGRAM OF ATMEGA 328P:

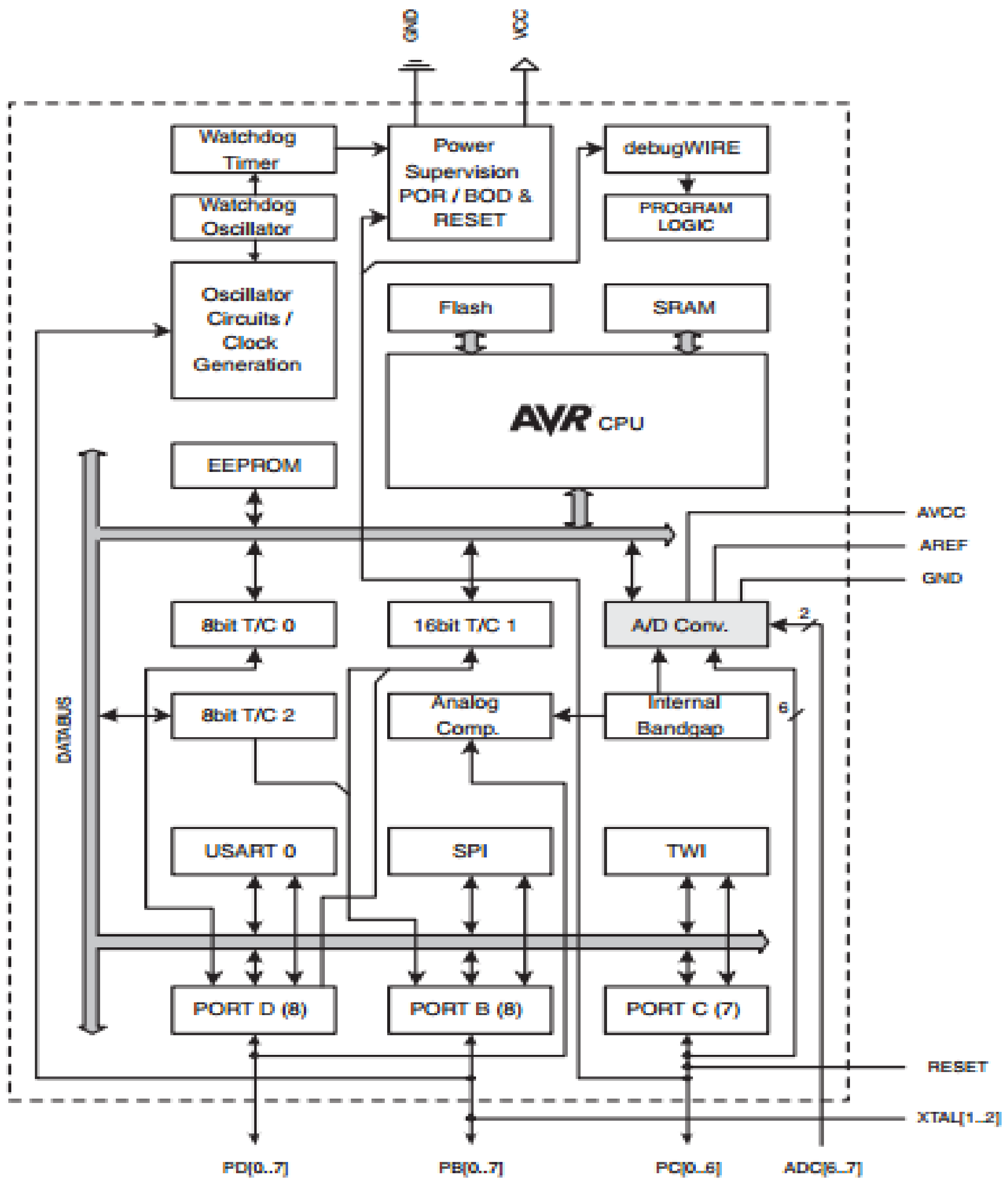


Fig 2.4.1: ATMEGA 328P's Block Diagram

The ATmega48PA/88PA/168PA/328P provides the following features: 4/8/16/32K bytes of InSystem Programmable Flash with Read-While-Write capabilities, 256/512/512/1K bytes

EEPROM, 512/1K/1K/2K bytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire Serial Interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable Watchdog Timer with internal Oscillator, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire Serial Interface, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega48PA/88PA/168PA/328P is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications. The ATmega48PA/88PA/168PA/328P AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

2.4.1 Comparison between ATmega48PA, ATmega88PA, ATmega168PA and ATmega328P

The ATmega48PA, ATmega88PA, ATmega168PA and ATmega328P differ only in memory sizes, boot loader support, and interrupt vector sizes. Table 2-1 summarizes the different memory and interrupt vector sizes for the three devices.

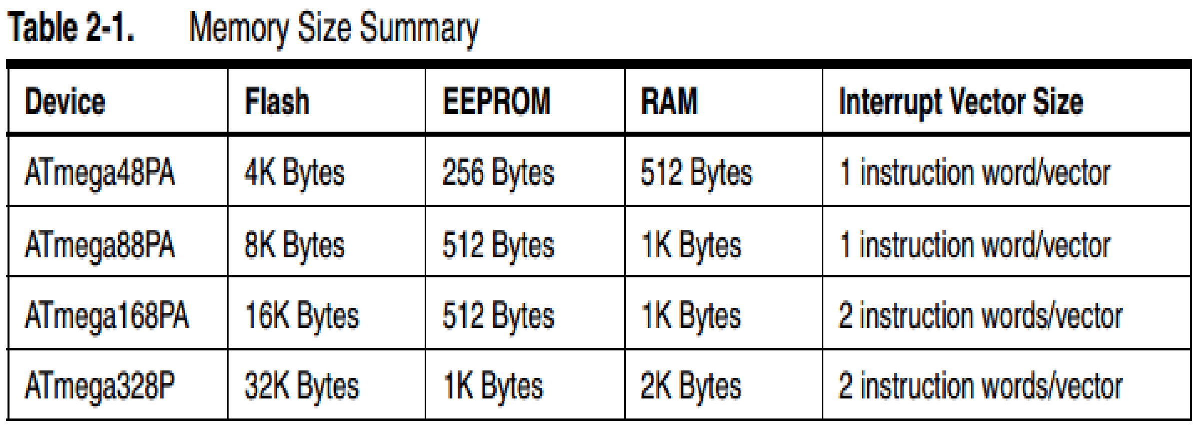


Table 2-1. Memory Size Summary

Device	Flash	EEPROM	RAM	Interrupt Vector Size
ATmega48PA	4K Bytes	256 Bytes	512 Bytes	1 instruction word/vector
ATmega88PA	8K Bytes	512 Bytes	1K Bytes	1 instruction word/vector
ATmega168PA	16K Bytes	512 Bytes	1K Bytes	2 instruction words/vector
ATmega328P	32K Bytes	1K Bytes	2K Bytes	2 instruction words/vector

Table 2.4.1: Comparison of different versions of ATMEGAs

ATmega88PA, ATmega168PA and ATmega328P support a real Read-While-Write Self-Programming mechanism. There is a separate Boot Loader Section, and the SPM instruction can only execute from there. In ATmega48PA, there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction can execute from the entire Flash.

2.4.2 POWER:

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts. The power pins are as follows:

• **VIN**. The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

• **5V.**This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.

• **3V3**. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

• **GND.** Ground pins.

2.4.3 Memory:

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

2.4.4 Input and Output

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

• **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the 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 connected to digital pin 13. When the pin is HIGH value, the

LED is on, when the pin is LOW, it's off.

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. Additionally, some pins have specialized functionality:

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

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.

See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

2.4.5 Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, 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. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Uno's digital pins. The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

2.4.6 Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the Tools > Board menu (according to the microcontroller on your board). For details, see the reference and tutorials. The ATmega328 on the Arduino Uno comes preburned with a 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, C header files). You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details. The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . 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 resetting 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 (Windows) or the DFU programmer (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 for more information.

2.4.7 Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno 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 uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment.

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 halfsecond 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 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; see this forum thread for details.

USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

2.5 Register File:

- 32 8-bit GP registers
- Part of SRAM memory space

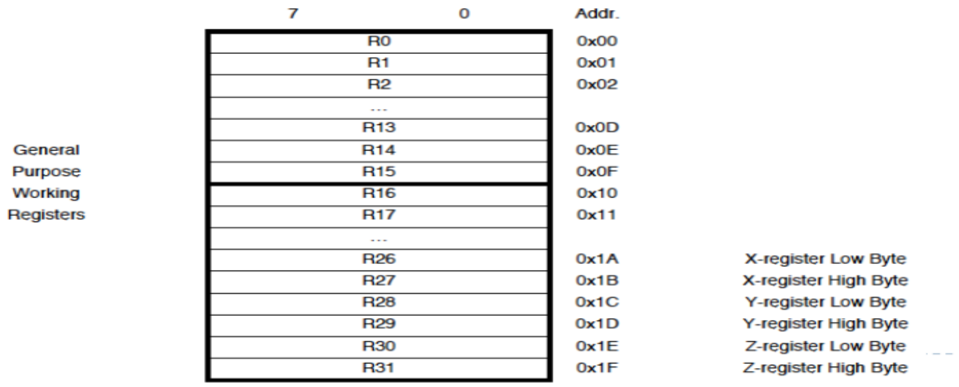


Fig 2.5.1: Register File

2.5.2: Special Addressing Registers

- X, Y and Z registers

  16-bit registers made using registers 26 – 31
- Support indirect addressing

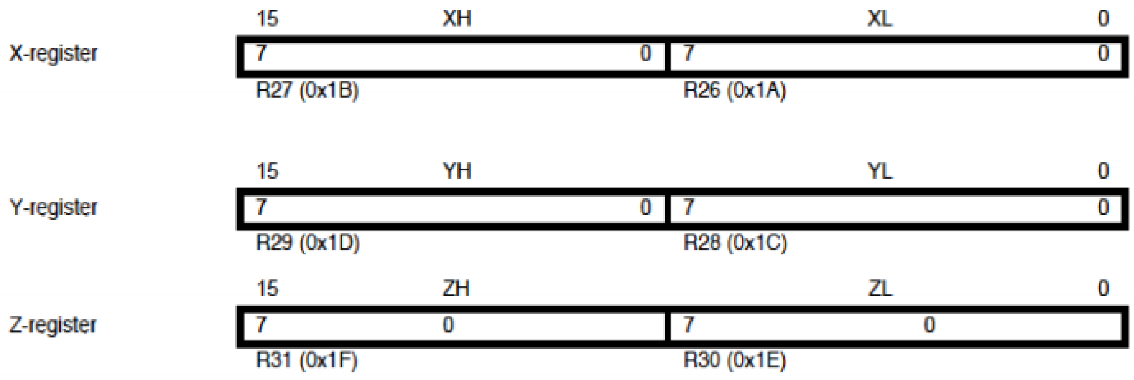


Fig 2.5.2: Special Addressing Registers

2.5.3 AVR Memory

- Program memory – Flash
- Data memory – SRAM

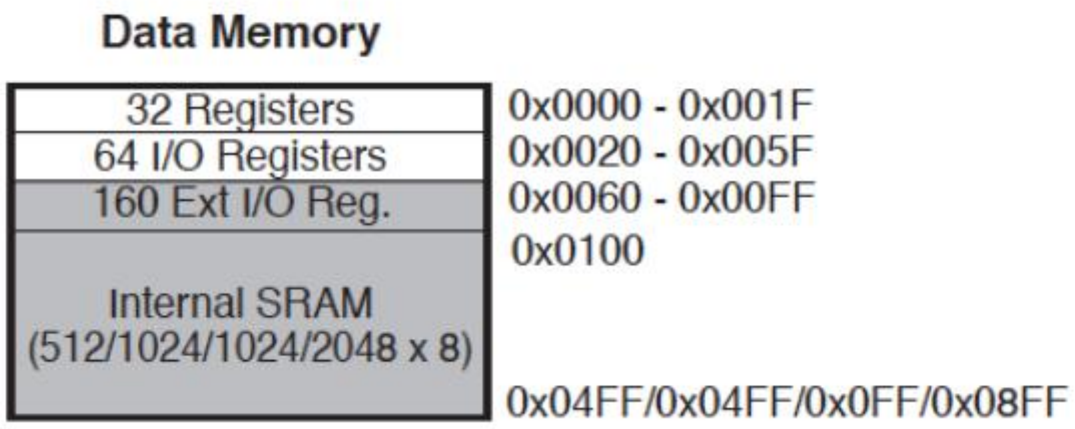


Fig 2.5.3: AVR Memory

2.6: Addressing Modes

2.6.1 Direct register addressing

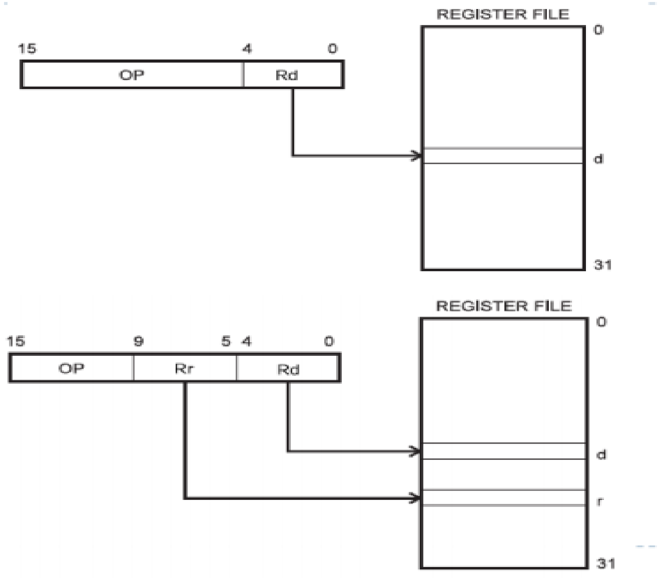


Fig 2.6.1: Direct Register Addressing

2.6.2 Direct I/O addressing

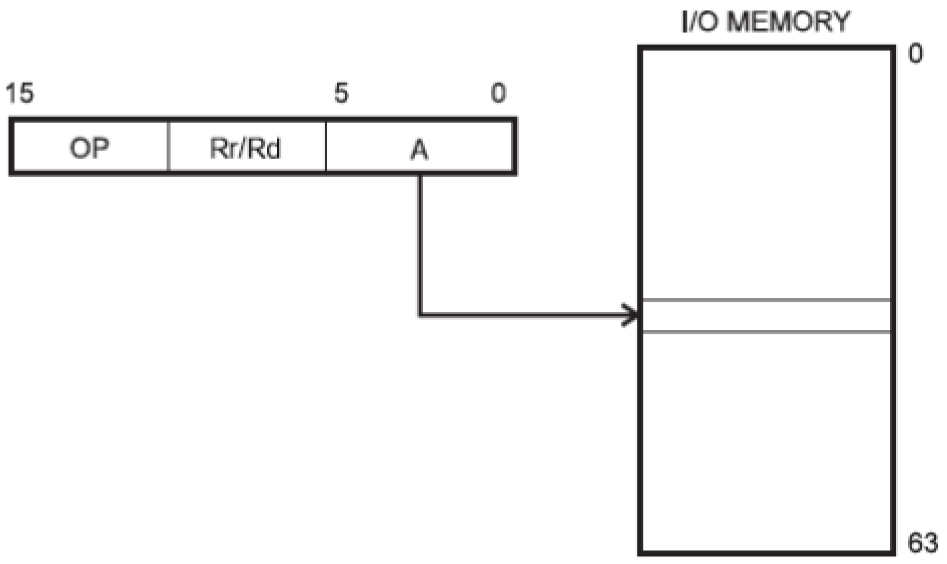


Fig 2.6.2: Direct I/O Addressing

2.6.3 Direct data memory addressing:

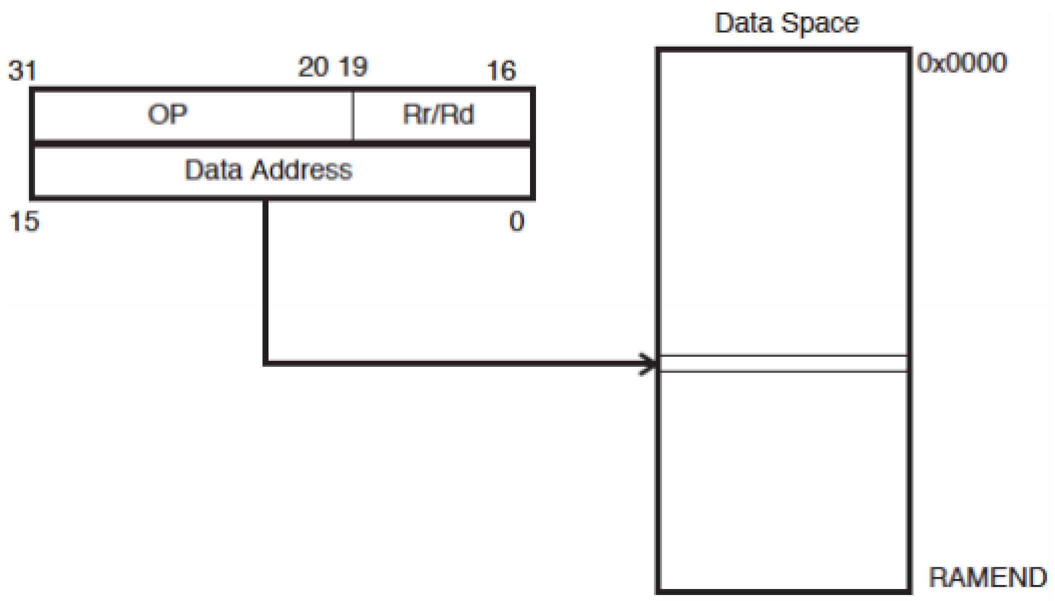


Fig 2.6.3 Direct data memory addressing:

2.6.4 Direct data memory with displacement addressing:

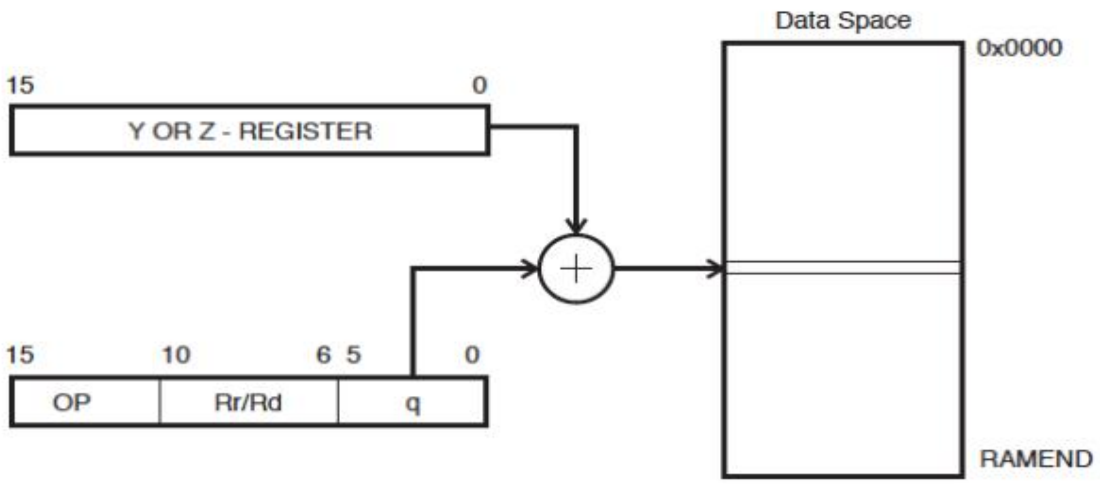


Fig: 2.6.4 Direct data memory with displacement addressing:

2.6.5 Indirect data memory addressing

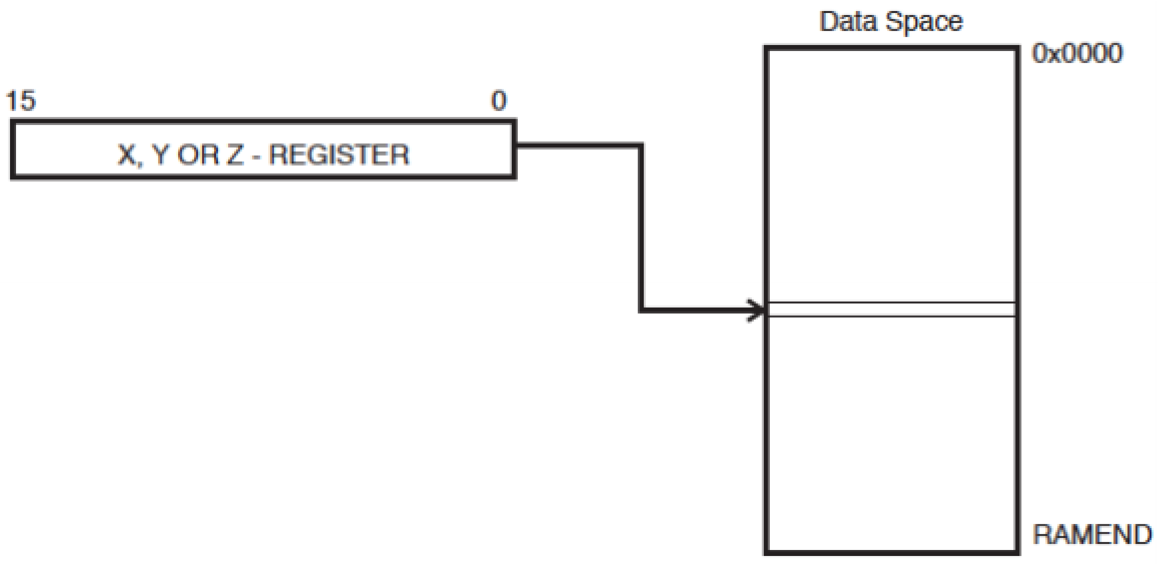


Fig 2.6.5 Indirect data memory addressing

2.6.6 Indirect data memory addressing with pre-decrement

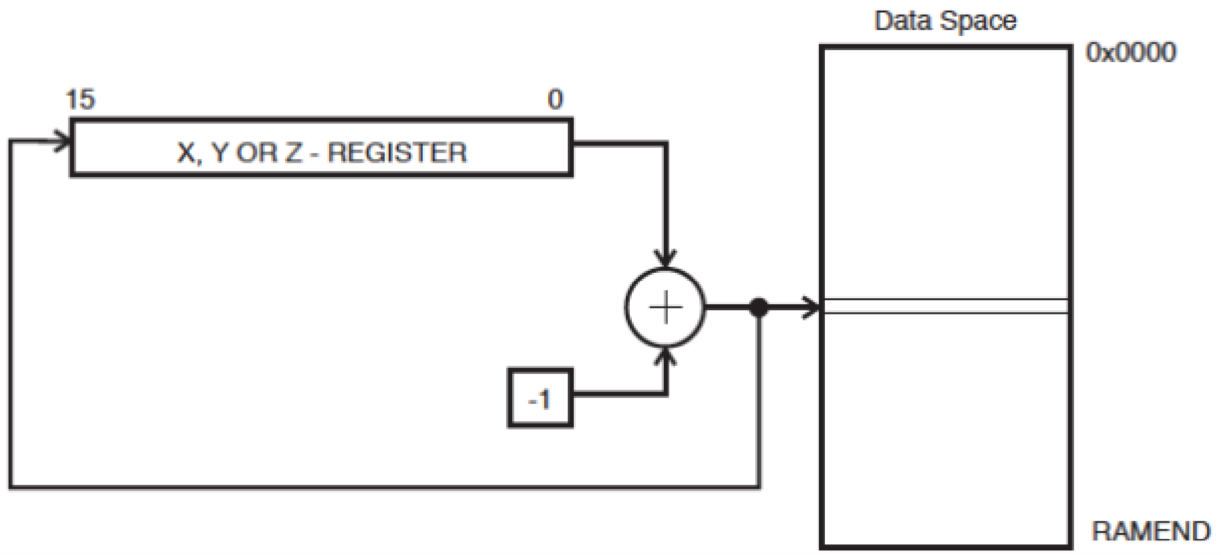


Fig 2.6.6 Indirect data memory addressing with pre-decrement

2.6.7 Indirect data memory addressing with post-increment

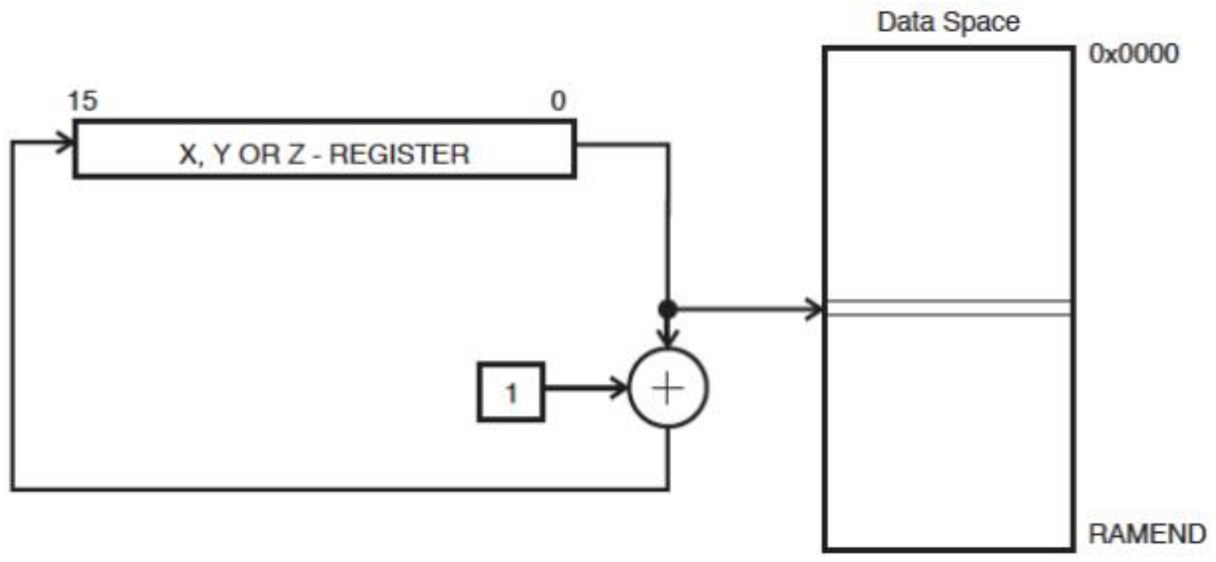


Fig: 2.6.7 Indirect data memory addressing with post-increment

2.6.8 Program memory addressing (constant data)

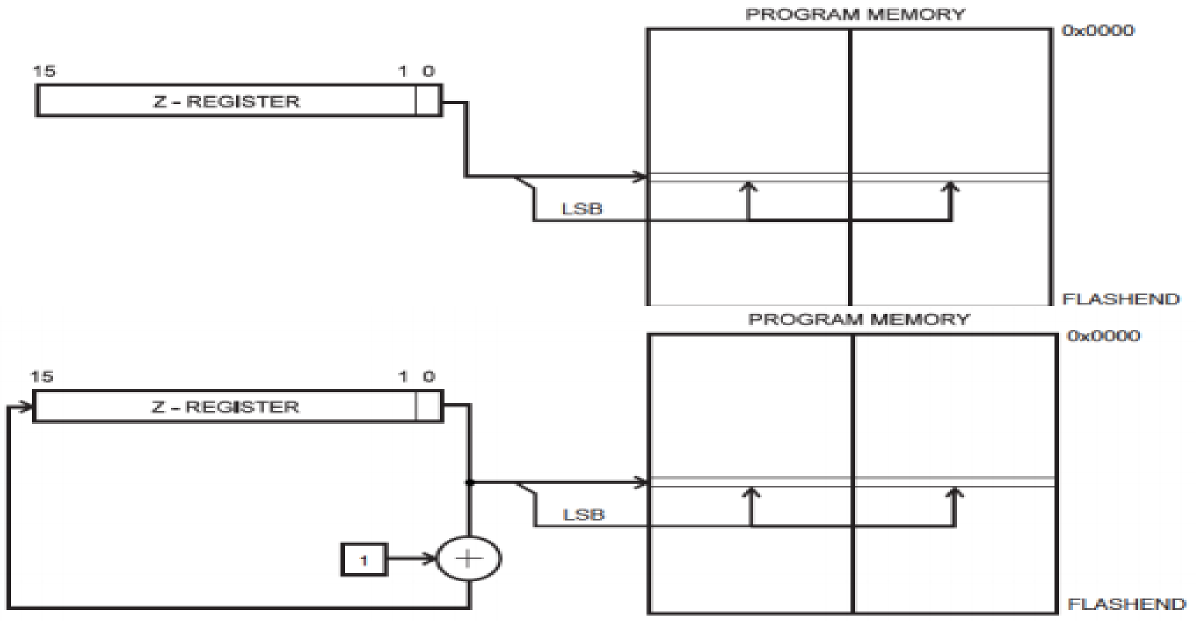
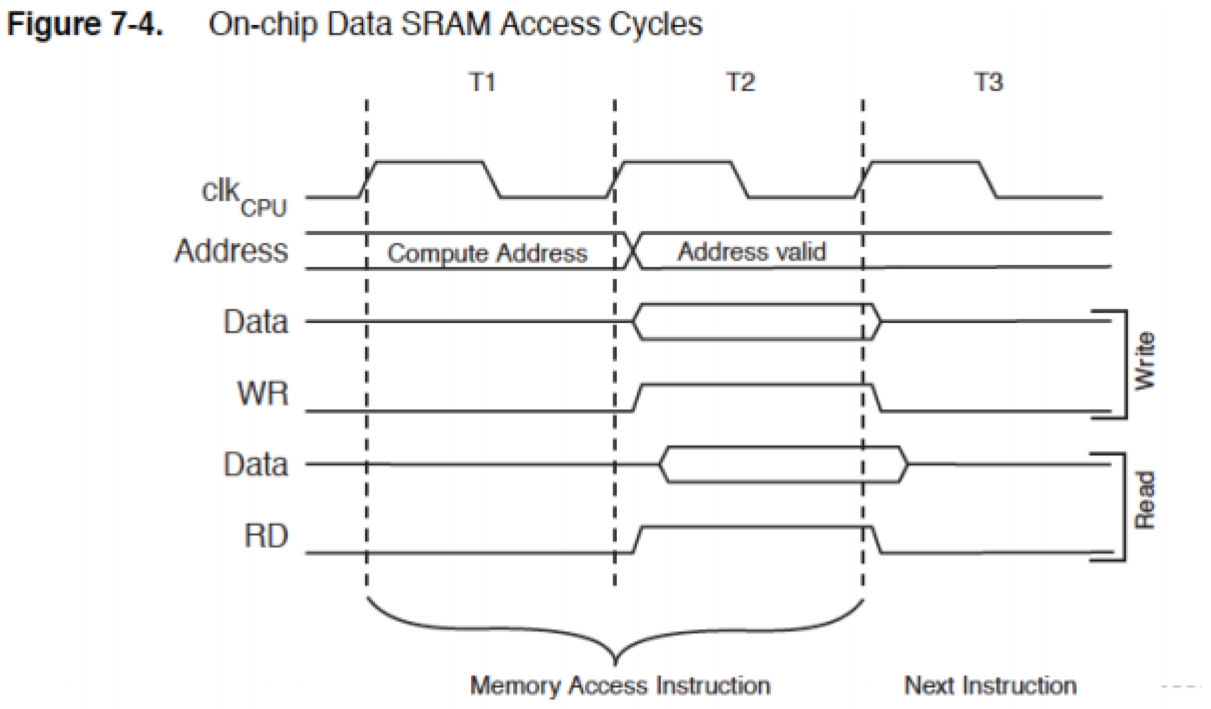


Fig 2.6.8 Program memory addressing

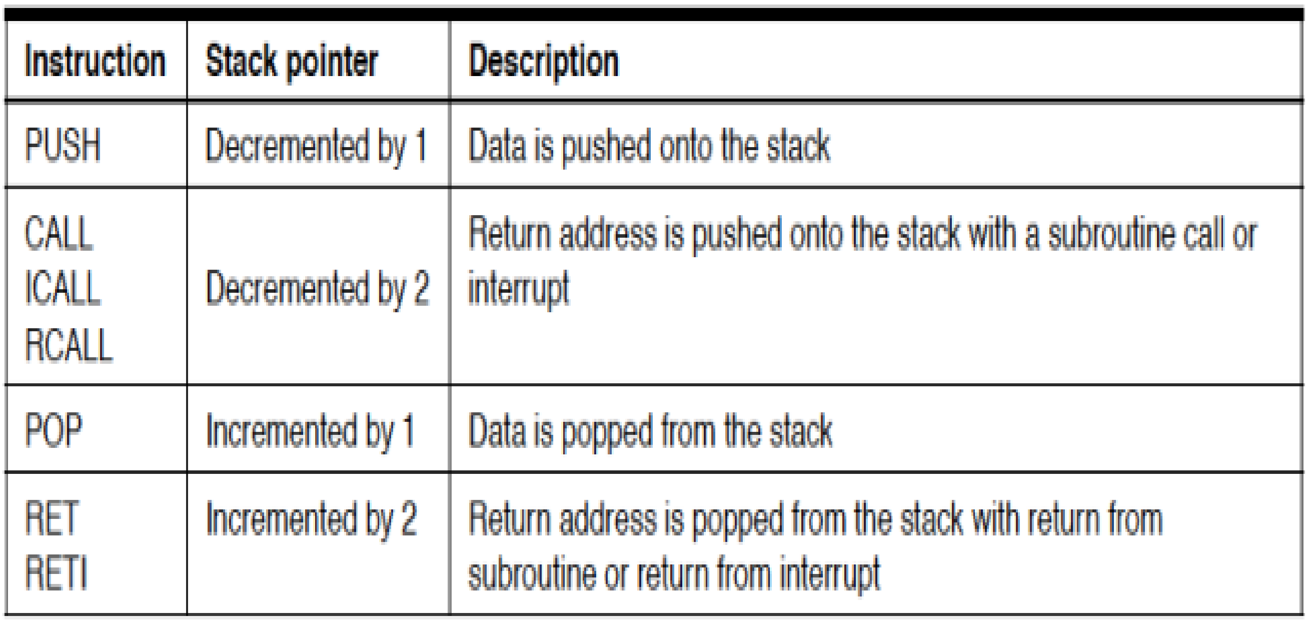
2.7 SRAM Read/Write Timing:



2.7 SRAM R/W Timings

Stack Pointer Register (SPR)

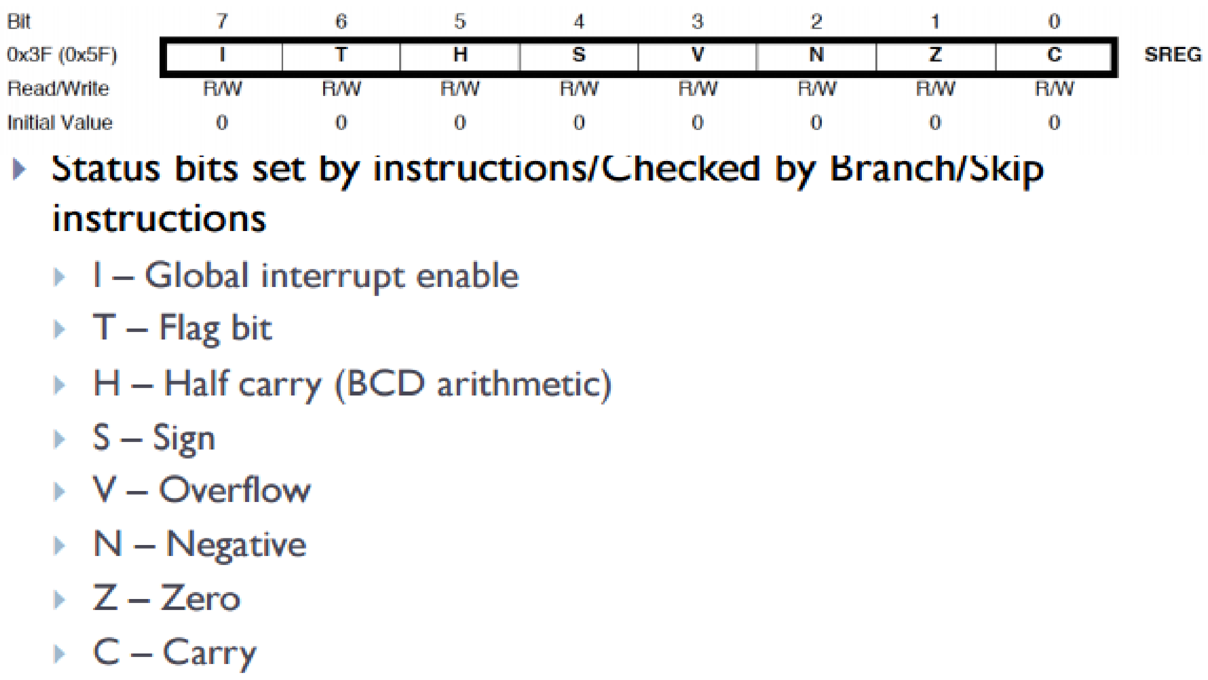
- Special register in I/O space [3E, 3D]
    - Enough bits to address data space
    - Initialized to RAMEND (address of highest memory address)
- Instructions that use the stack pointer



Instruction	Stack pointer	Description
PUSH	Decremented by 1	Data is pushed onto the stack
CALL ICALL RCALL	Decremented by 2	Return address is pushed onto the stack with a subroutine call or interrupt
POP	Incremented by 1	Data is popped from the stack
RET RETI	Incremented by 2	Return address is popped from the stack with return from subroutine or return from interrupt

Fig 2.7 Instruction Set of SPR

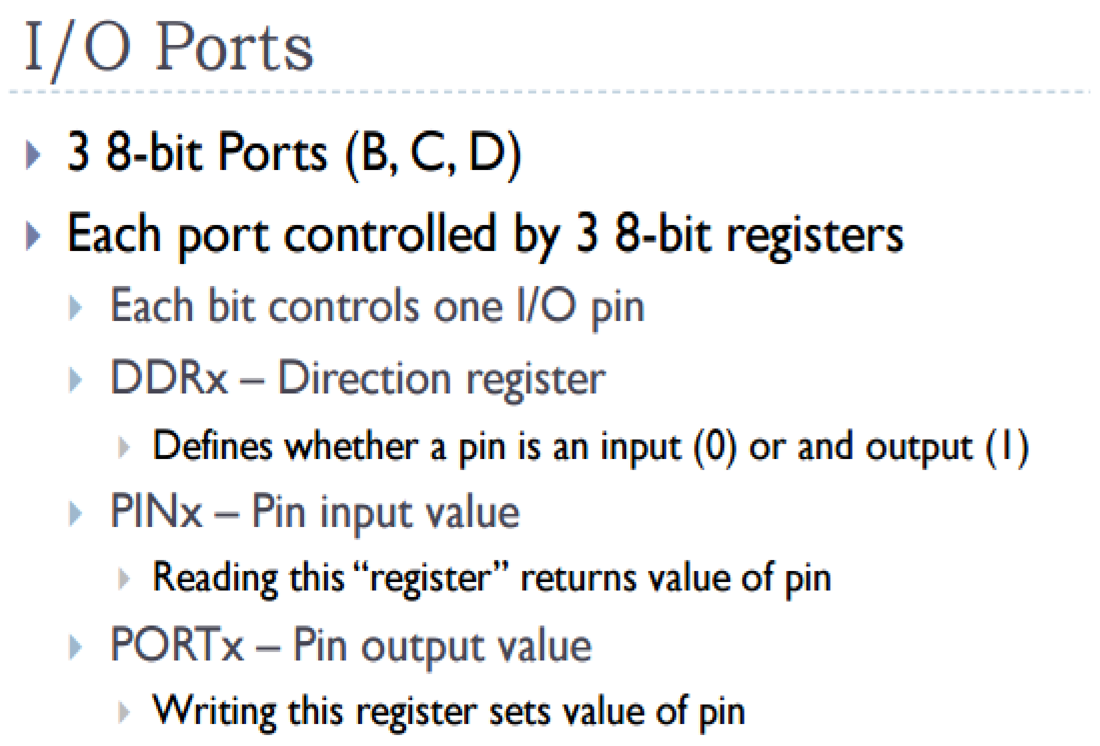
Program Status Register (PSR)



Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Status bits set by instructions/Checked by Branch/Skip instructions
  - I – Global interrupt enable
  - T – Flag bit
  - H – Half carry (BCD arithmetic)
  - S – Sign
  - V – Overflow
  - N – Negative
  - Z – Zero
  - C – Carry

Fig 2.7.2 Status Bits of PSR



I/O Ports

- 3 8-bit Ports (B, C, D)
- Each port controlled by 3 8-bit registers
  - Each bit controls one I/O pin
  - DDRx – Direction register
    - Defines whether a pin is an input (0) or and output (1)
  - PINx – Pin input value
    - Reading this "register" returns value of pin
  - PORTx – Pin output value
    - Writing this register sets value of pin

2.8 Ethernet Shield:

Your Ethernet Shield gets a total renewal now! This shield provides you instant Internet connectivity with a high spec Ethernet controller, W5200, with twice the buffer size of v1.0 and support for up to eight simultaneous TCP/UDP connections. An included SD slot enables applications that require storing large amounts of data, like IoT data logging. Thanks to a lowered RJ45 pot, you can flexibly add most of shields on top of this Ethernet Shield.

Model: SLD91000P

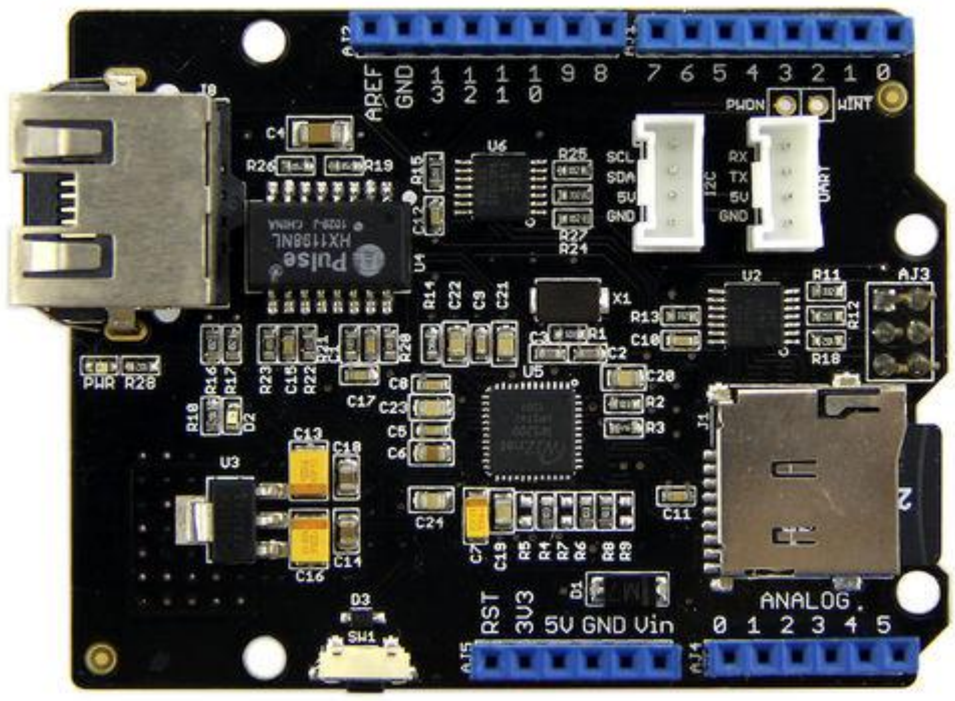


Fig 2.8.1: Ethernet Shield

2.8.1 Features

- High speed Ethernet controller W5200
- SPI interface
- 32 Kbytes inner buffer
- Minimal RJ45 Ethernet port
- Support up to eight simultaneous TCP/UDP connections
- Handy SD card function
- Grove ports for I2C and UART preformed

2.8.2 Interface

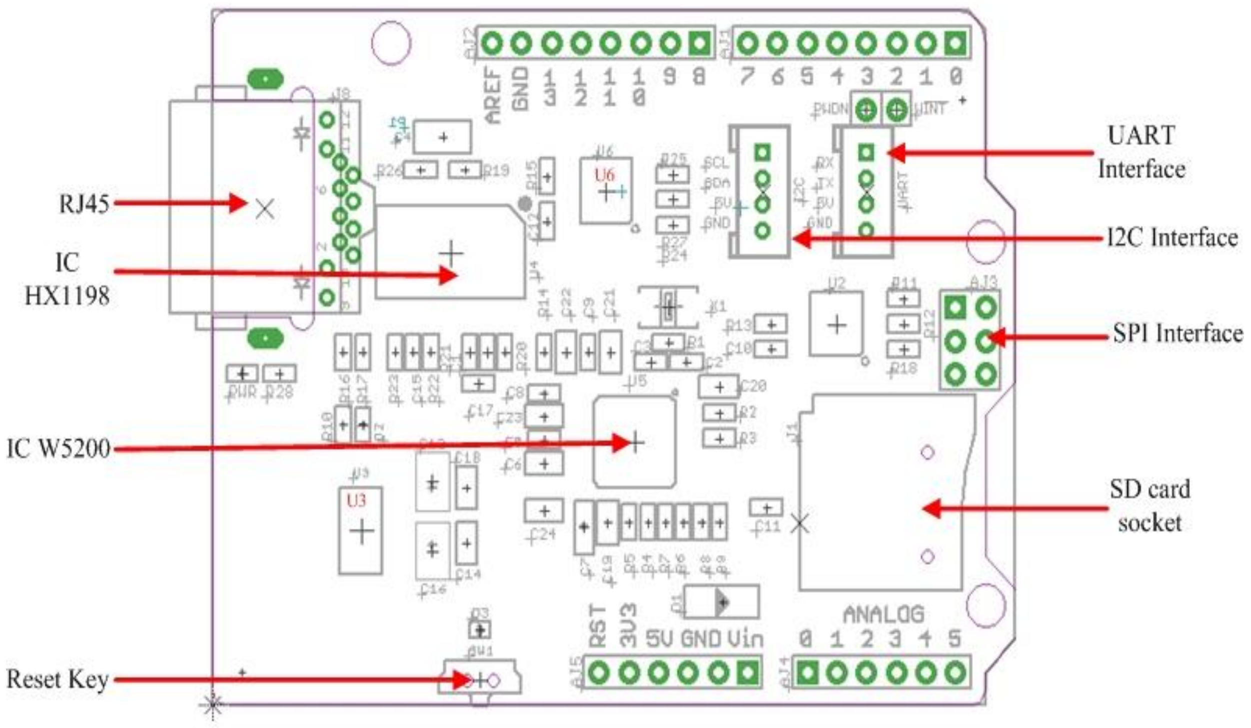


Fig 2.8.2: Interface of Ethernet Shield

Hardware Configuration

RJ45: Ethernet Port;

IC HX1198: 10/100BASE-T signal port;

IC W5200: a hardwired TCP/IP Ethernet Controller;

U3: IC CJ117, low dropout linear regulator;

U6: IC 74VHC125PW, quad buffer;

Reset KEY: Reset Ethernet shield and Arduino when pressed;

SD card: support Micro SD card in FAT16 or FAT32; maximum storage is 2GB.

Pins usage on Arduino

D4: SD card chip select

D10: W5200 Chip Select

D11: SPI MOSI

D12: SPI MISO

D13: SPI SCK

Notice:

Both W5200 and SD card communicate with Arduino via SPI bus. Pin 10 and pin 4 are chip select pins for W5200 and SD slot. They cannot be used as general I/O.

2.8.3 Usage Manual

We are going to build a simple web server that answer request from a client and store the readings from A0 through A5 to SD card.

Step 1: Hookup

1. Mount Ethernet Shield v2.0 to your Arduino;

2. Connect the shield to your computer or a network hub or a router with a standard Ethernet cable;

3. Connect Arduino to PC via USB cable;

4. Insert an SD card to the SD card slot.

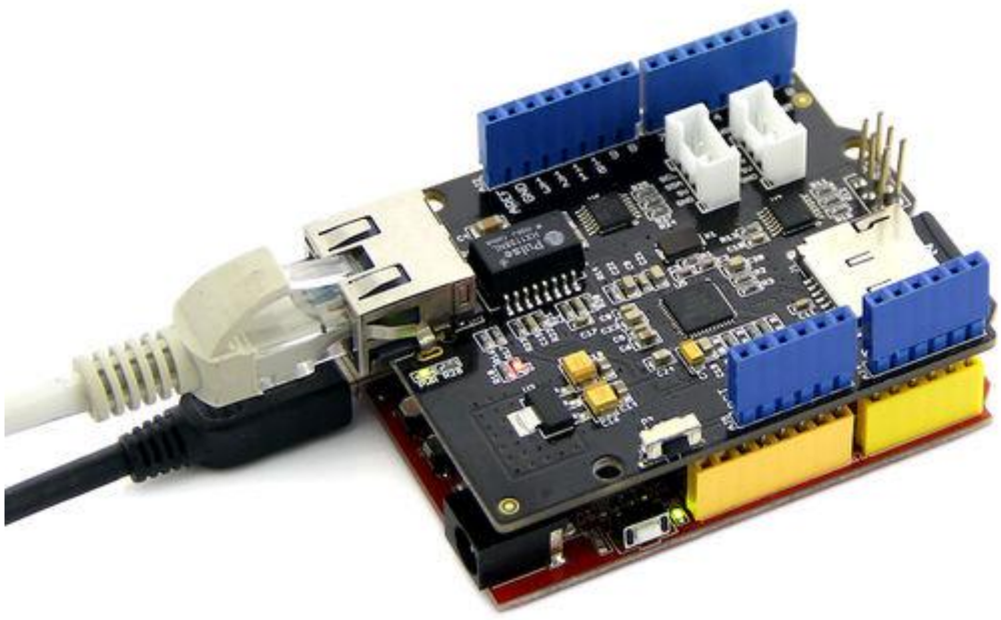


Fig 2.8.3: Ethernet fixed on ARDUINO

Step 2: Upload the program

1. Download the library: Ethernet Shield V2.0 Library
   Note: Depreciated/Old Library for 1.0.x IDE: Link.

2. Unzip and put it in the libraries file of Arduino IDE by the path: ..\arduino-1.0.1\libraries.

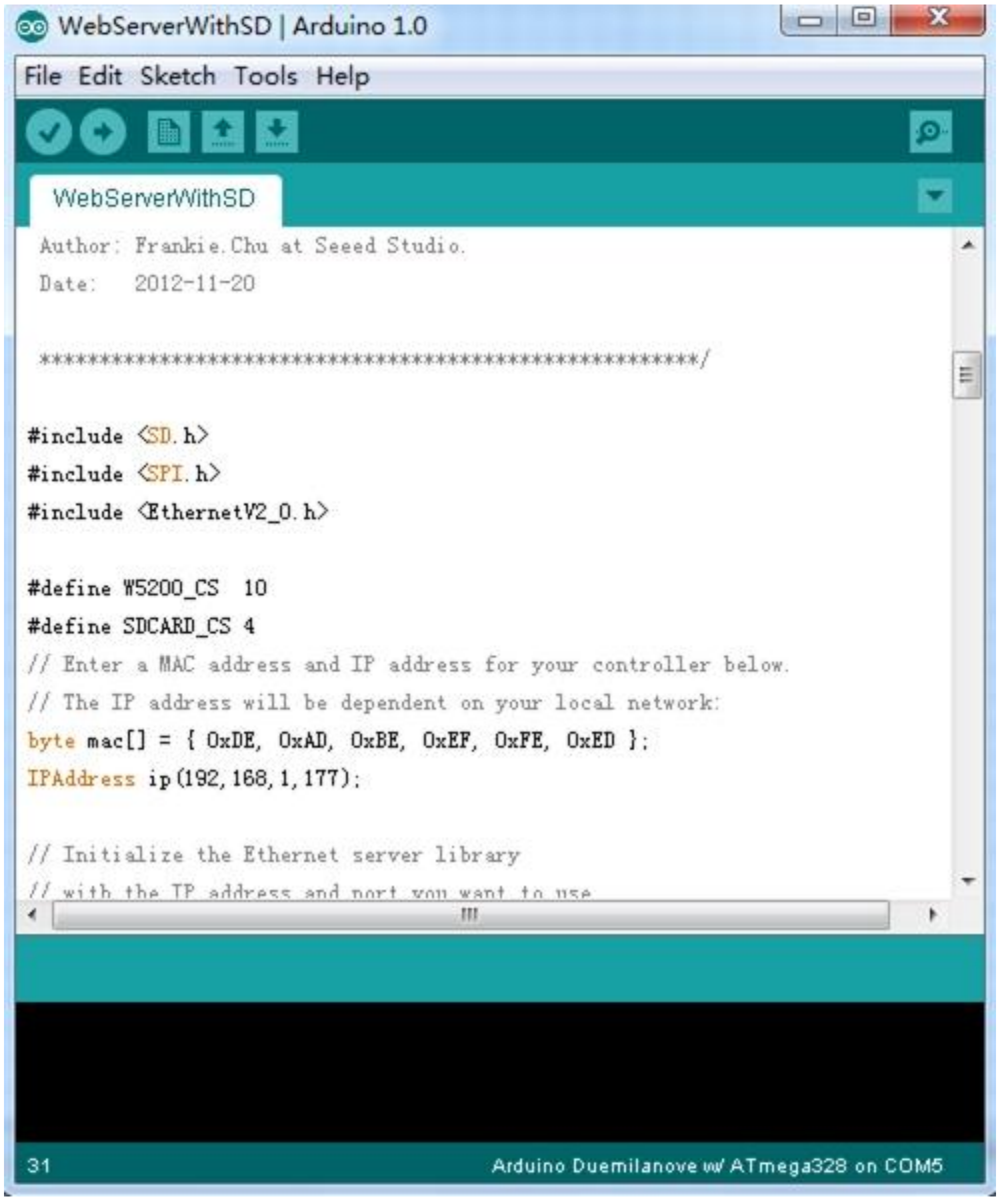
3. Restart the Arduino IDE.

4. Open the example "WebServerWithSD" via the path: File --> Examples --> EthernetV2.0 --> WebServerWithSD. This example shows you how to build up a simple web server that displays the readings of anolog A0 through A5 when requested. After that, store those readings into SD card.

Note:

This new library covers all functions included in the build-in Ethernet library of Arduino IDE. You can use other examples in the same way as that in the preceding Ethernet library.

5. Upload the program to Arduino. If you do not know how to upload code, please click here.


WebServerWithSD | Arduino 1.0
File Edit Sketch Tools Help
WebServerWithSD
31
Arduino Duemilanove w/ ATmega328 on COM5

```
 Author: Frankie.Chu at Seeed Studio.
 Date:   2012-11-20

 *******************************************************/

#include <SD.h>
#include <SPI.h>
#include <EthernetV2_0.h>

#define W5200_CS  10
#define SDCARD_CS 4
// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1,177);

// Initialize the Ethernet server library
// with the IP address and port you want to use
```

Fig 2.8.4 Webserver with SD Card

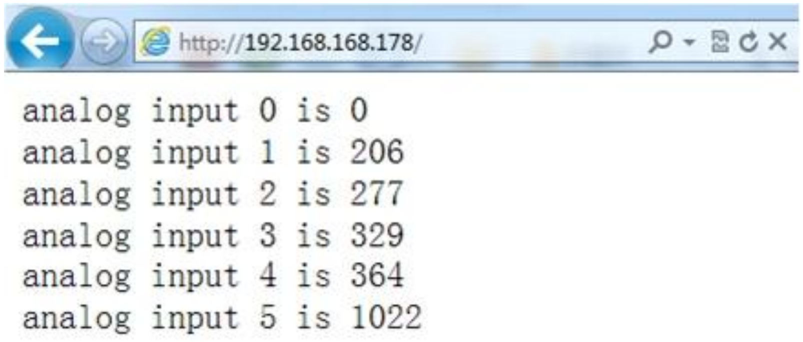
In this code ,we have defined pin 4 as SD card chip select port and pin 10 as W5200 chip select port.

Firstly, it will send a link to this client if there is a client requesting access to this server.Then send the value of each analog input pin to the network.

Finally you can view each analog pin by opening SD Card file.

Step 3: Results

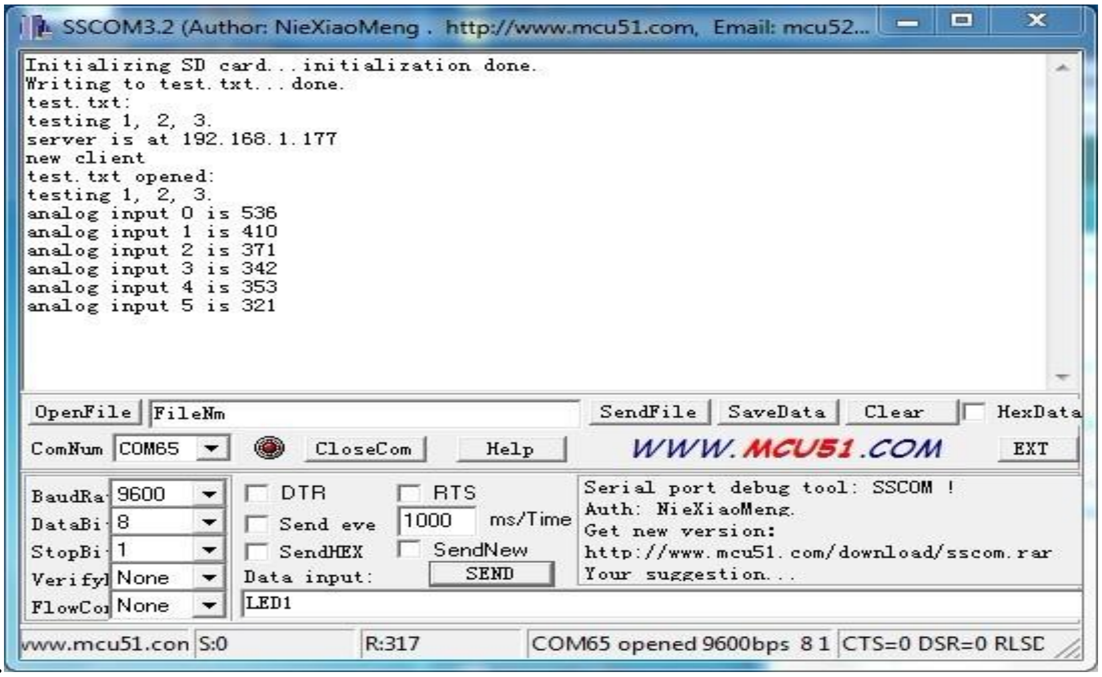
Open a web browser and enter the IP address of your controller. It's dependent on your local network but used to be 192.168.168.178. Then you should find the readings of A0 through A5 popping up as shown below.



```
analog input 0 is 0
analog input 1 is 206
analog input 2 is 277
analog input 3 is 329
analog input 4 is 364
analog input 5 is 1022
```

Fig 2.8.5 Analog results

To check what's going on to the SD card, open the serial monitor. You can use the built-in serial monitor of Arduino IDE or a serial monitor tool as us. After opening a serial monitor, you can read the content of file "test.txt" which we created to store the readings of analog pins.



```
Initializing SD card...initialization done.
Writing to test.txt...done.
test.txt:
testing 1, 2, 3.
server is at 192.168.1.177
new client
test.txt opened:
testing 1, 2, 3.
analog input 0 is 536
analog input 1 is 410
analog input 2 is 371
analog input 3 is 342
analog input 4 is 353
analog input 5 is 321
```

ComNum COM65
BaudRa 9600
DataBi 8
StopBi 1
Verifyl None
FlowCo None
Send eve 1000 ms/Time
Data input: LED1

```
Serial port debug tool: SSCOM !
Auth: NieXiaoMeng.
Get new version:
http://www.mcu51.com/download/sscom.rar
Your suggestion...
```

www.mcu51.con S:0 R:317 COM65 opened 9600bps 8 1 CTS=0 DSR=0 RLSD

Fig 2.8.6 Test.txt

Notice:

1） Make sure the Ethernet Shield and your computer are in the same local network.

2） Once the code has been successfully uploaded, it's fine to disconnect the board from your computer and apply independent power to it, leaving it run alone.

2.9 POWER SUPPLY:

WHAT IS A POWER SUPPLY?

A power supply is a device which delivers an exact voltage to another device as per its needs. There are many power supplies available today in the market like regulated, unregulated, variable etc, and the decision to pick the correct one depends entirely on what device you are trying to operate with the power supply. Power supplies, often called power adapters, or simply adapters, are available in various voltages, with varying current capacities, which is nothing but the maximum capacity of a power supply to deliver current to a load (Load is the device you are trying to supply power to).

WHY MAKE ONE YOURSELF?

`One would ask himself, "Why do I make it myself, when it available in the market?" Well, the answer is- even if you buy one, it is bound to stop working in a while (and believe me, power supplies stop working without any prior indication, one day they'll work, the next day they'll just stop working!). So, if you build one yourself, you will always know how to repair it, as you will know exactly what component/part of the circuit is doing what. And further, knowing how to build one, will allow you to repair the ones you have already bought, without wasting your money on a new one.

WHAT YOU NEED?

1. Copper wires, with at least 1A current carrying capacity for AC mains
2. Step Down Transformer
3. 1N4007 Silica Diodes (×4)
4. 1000µF Capacitor
5. 10µF Capacitor
6. Voltage regulator (78XX) (XX is the output voltage reqd. I'll explain this concept later)
7. Soldering iron
8. Solder
9. General Purpose PCB
10. Adapter jack (to provide the output voltage to a device with a particular socket)
11. 2 Pin plug

Optional

a. LED (for indication)
b. Resistor (Value explained later)
c. Heat Sink for The Voltage Regulator (For higher current outputs)
d. SPST Switch

SOME BASIC CONCEPTS RELATED TO POWER SUPPLIES

2.9.1 Transformers

Transformers are devices which step down a relatively higher AC input Voltage into a lower AC output voltage. To find the input and output terminals of a transformer is very tricky. Refer to the following illustration or the internet to understand where what is.

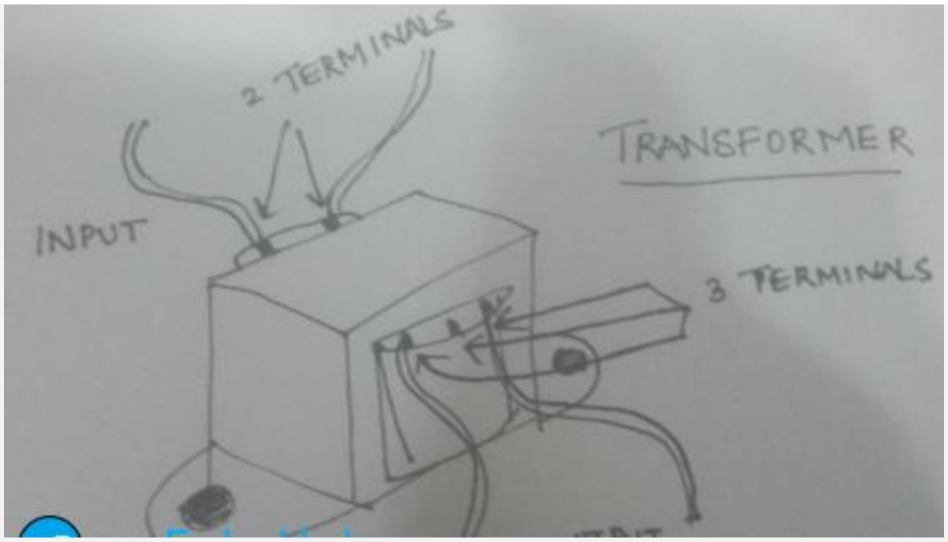


Fig 2.9.1: I/O Terminals of a Transformer

Basically, there are two sides in a transformer where the coil winding inside the transformer ends. Both ends have two wires each (unless you are using a center-tapped transformer for full wave rectification). On the transformer, one side will have three terminals and the other will have two. The one with the three terminals is the stepped down output of the transformer, and the one with the two terminals is where the input voltage is to be provided.

2.9.2 Voltage Regulators

The 78XX series of voltage regulators is a widely used range of regulators all over the world. The XX denotes the voltage that the regulator will regulate as output, from the input voltage. For instance, 7805, will regulate the voltage to 5V. Similarly, 7812 will regulate the voltage to 12V. The thing to remember with these voltage regulators is that they need at least 2 volts more than their output voltage as input. For instance, 7805 will need at least 7V, and 7812, at least 14 volts as inputs. This excess voltage which needs to be given to voltage regulators is called **Dropout Voltage**.

NOTE: The input pin is denoted as ‘1’, ground as ‘2’ and output as ‘3’.

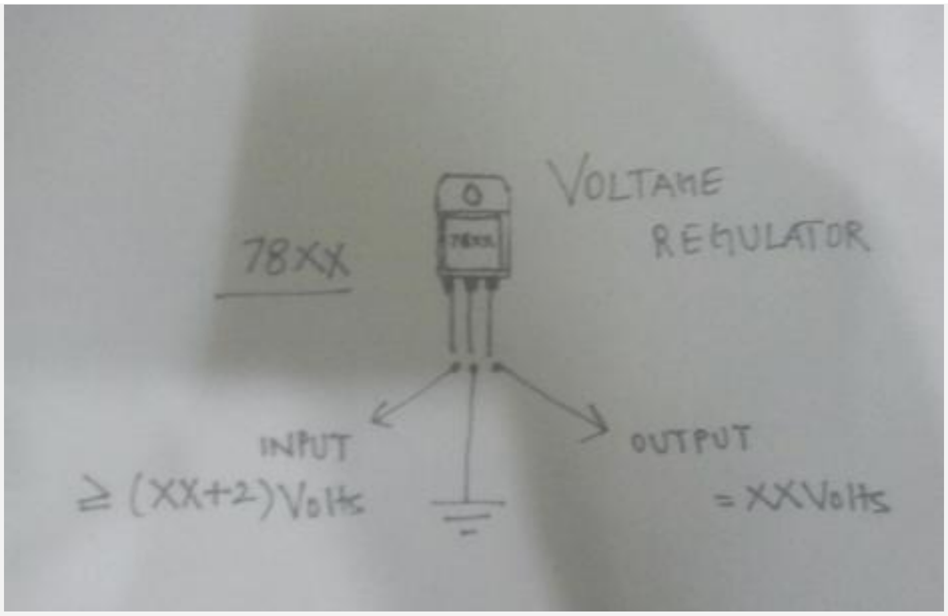


Fig 2.9.2 Voltage Regulator Schematic

2.9.3 Diode Bridge

A bridge rectifier consists of an assembly of four ordinary diodes, by means of which we can convert AC Voltage into DC Voltage. It is found to be the best model for AC to DC conversion, over Full wave and Half wave rectifiers. You can use any model you want, but I use this for the sake of high efficiency (If you are using the full wave rectifier model, you'll need a center-tapped transformer, and you will only be able to use half of the transformed voltage).

One thing to note about diodes is that they drop about 0.7V each when operated in forward bias. So, in bridge rectification we will drop 1.4V because at one instant two diodes are conducting and each will drop 0.7V. In case of Full wave rectifier, only 0.7V will be dropped.

So how does this drop affect us? Well, this comes in handy while choosing the correct step down voltage for the transformer. See, our voltage regulator needs 2 Volts more than its output voltage. For the sake of explanation, let's assume that we are making a 12V adapter. So the voltage regulator needs at least 14 Volts as input.

So the output of the diodes (which goes into the voltage regulator) will have to be more than or equal to 14 Volts. Now for the diodes' input voltage. They'll drop 1.4 Volts in total, so the input to them has to be greater than or equal to 14.0 + 1.4 = 15.4Volts. So I would probably use a 220 to 18 Volt step down transformer for that.

So basically, the transformer step down voltage should be at least 3.4V more than the desired Power Supply output.

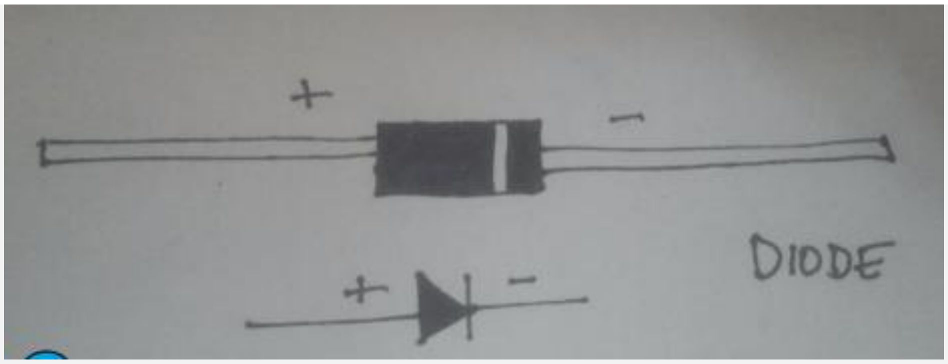


Fig 2.9.3: Schematic and Illustration of a Diode

2.9.4 Filter Circuit

We filter, at the output of the voltage rectifier in order to get the smoothest DC Voltage as possible, from our adapter, for which we use capacitors. Capacitors are the simplest current filters available, they let AC current pass through and block DC, so they are used in parallel to the output. Furthermore, if there is a ripple in the input or output, a capacitor rectifies it by discharging the charge stored in it.

HOW TO BUILD IT?

Here is the circuit diagram for the Power Supply:

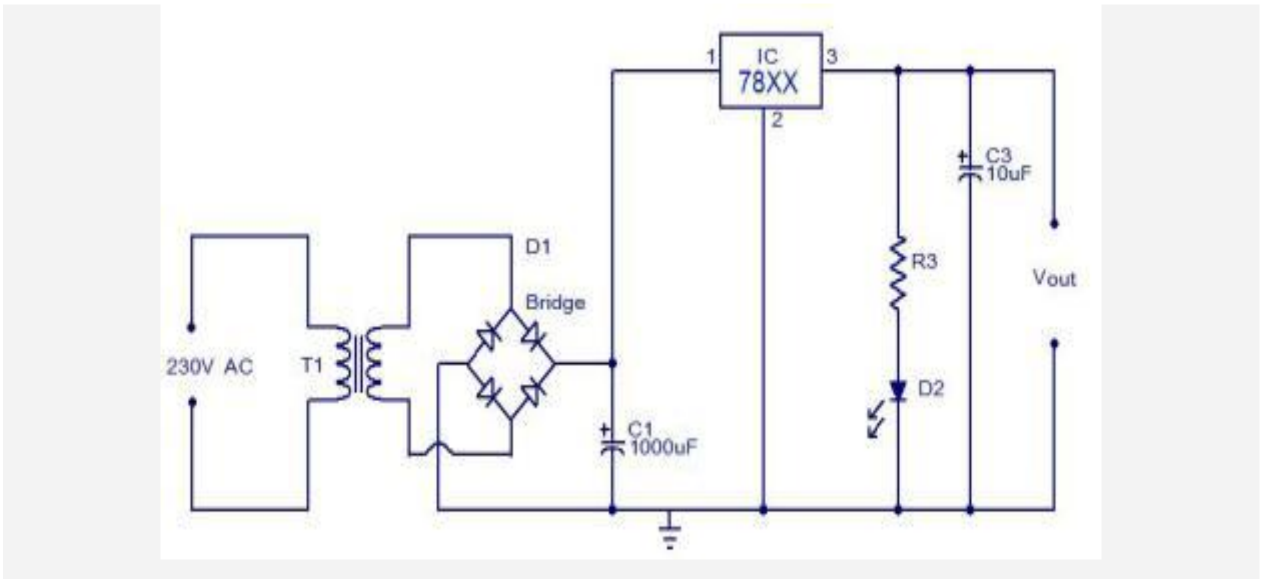


Fig 2.9.4 Circuit Diagram

How it works?

The AC mains are fed to the transformer, which steps down the 230 Volts to the desired voltage. The bridge rectifier follows the transformer thus converting AC voltage into a DC output and through a filtering capacitor feeds it directly into the input (Pin 1) of the voltage regulator. The common pin (Pin 2) of the voltage regulator is grounded. The output (Pin 3) of the voltage regulator is first filtered by a capacitor, and then the output is taken.
Make the circuit on a general purpose PCB and use a 2 Pin (5A) plug to connect the transformer input to the AC mains via insulated copper wires.

2.10 RELAYS:

WHAT IS A RELAY?

We know that most of the high end industrial application devices have relays for their effective working. Relays are simple switches which are operated both electrically and mechanically. Relays consist of a n electromagnet and also a set of contacts. The switching mechanism is carried out with the help of the electromagnet. There are also other operating principles for its working. But they differ according to their applications. Most of the devices have the application of relays.

WHY IS A RELAY USED?

The main operation of a relay comes in places where only a low-power signal can be used to control a circuit. It is also used in places where only one signal can be used to control a lot of circuits. The application of relays started during the invention of telephones. They played an important role in switching calls in telephone exchanges. They were also used in long distance telegraphy. They were used to switch the signal coming from one source to another destination. After the invention of computers they were also used to perform Boolean and other logical operations. The high end applications of relays require high power to be driven by electric motors and so on. Such relays are called contactors.

RELAY DESIGN

There are only four main parts in a relay. They are

- Electromagnet
- Movable Armature
- Switch point contacts
- Spring

The figures given below show the actual design of a simple relay.

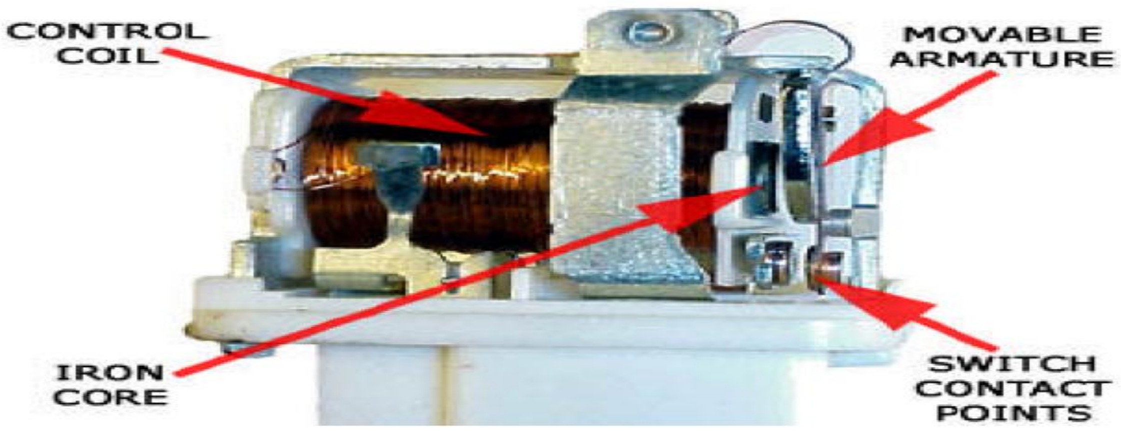


Fig 2.10.1 Relay Construction

It is an electro-magnetic relay with a wire coil, surrounded by an iron core. A path of very low reluctance for the magnetic flux is provided for the movable armature and also the switch point contacts. The movable armature is connected to the yoke which is mechanically connected to the switch point contacts. These parts are safely held with the help of a spring. The spring is used so as to produce an air gap in the circuit when the relay becomes de-energized.

HOW RELAY WORKS?

The working of a relay can be better understood by explaining the following diagram given below.

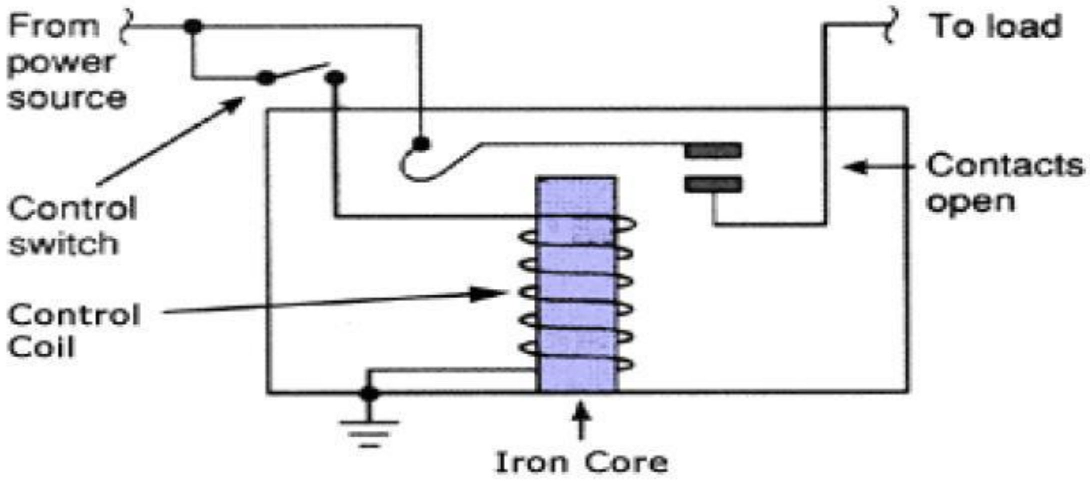


Fig 2.10.2 Relay Design

The diagram shows an inner section diagram of a relay. An iron core is surrounded by a control coil. As shown, the power source is given to the electromagnet through a control switch and through contacts to the load. When current starts flowing through the control coil, the electromagnet starts energizing and thus intensifies the magnetic field. Thus the upper contact arm starts to be attracted to the lower fixed arm and thus closes the contacts causing a short circuit for the power to the load. On the other hand, if the relay was already de-energized when the contacts were closed, then the contact move oppositely and make an open circuit.

As soon as the coil current is off, the movable armature will be returned by a force back to its initial position. This force will be almost equal to half the strength of the magnetic force. This force is mainly provided by two factors. They are the spring and also gravity.

Relays are mainly made for two basic operations. One is low voltage application and the other is high voltage. For low voltage applications, more preference will be given to reduce the noise of the whole circuit. For high voltage applications, they are mainly designed to reduce a phenomenon called arcing.

2.10.2 HOW TO CONNECT A SINGLE POLE SINGLE THROW (SPST) RELAY IN CIRCUIT

In order to know how to connect a single pole single throw (SPST) relay, you must know what each pin terminal represents and how the relay works.

Terminal Pins

A Single Pole Single Throw Relay comes with four terminal points. The terminals are COIL, COIL , COM, and N.O.

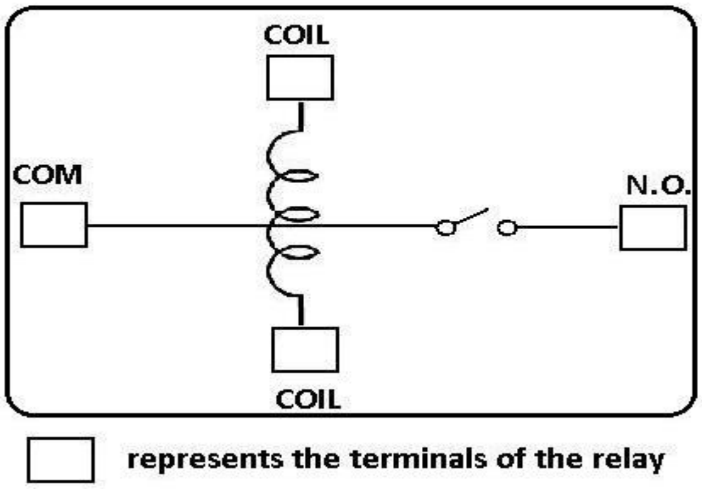


Fig 2.10.2.1: Terminals of relay

This correlates to the following in the relay:

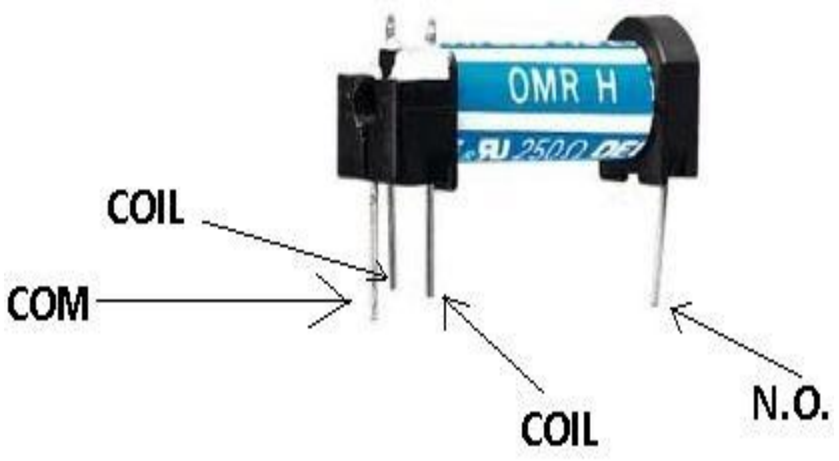


Fig 2.10.2.2: Relay Terminals

Terminal Descriptions

-COIL- This is one end of the coil.

COIL- This is the other end of the coil. These are the terminals where you apply voltage to in order to give power to the coils (which then will close the switch). Polarity does not matter. One side gets positive voltage and the other side gets negative voltage. Polarity only matters if a diode is used.

NO- This is Normally Open switch. This is the terminal where you connect the device that you want the relay to power when the relay is powered, meaning when the COIL receives sufficient voltage. The device connected to NO will be off when the relay has no power and will turn on when the relay receives power.

COM- This is the common of the relay. If the relay is powered and the switch is closed, COM and N.O. have continuity. This is the terminal of the relay where you connect the first part of your circuit to.

Now that we know what each terminal pin represents, we now wire it to a circuit for it to do a real-world function. We're going to connect a single pole single throw relay to a circuit to light up a LED. This is the circuit below to connect a single pole single throw relay to light a LED:

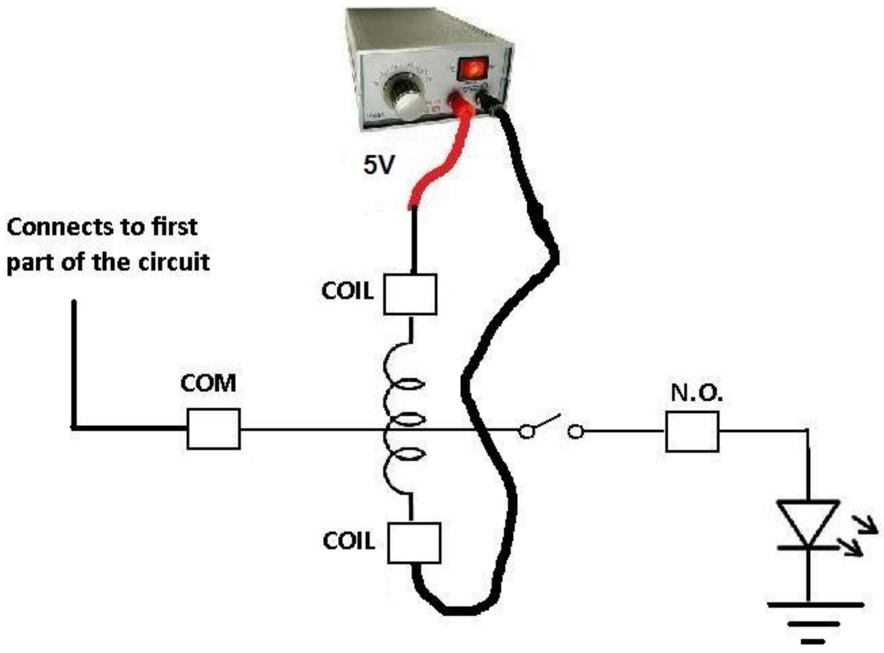


Fig 2.10.2.3 Connection of Relays

Since the relay is rated for 5V, it should receive 5 volts in order to power on. It may work with less voltage, but 5V is really what it should receive. This goes into either side of the COIL terminals. Even if you switched the positive and negative voltage of the power supply, it should work exactly the same.

The COM terminal of the relay gets connected to the first part of the circuit. If there is no first part of the circuit, this terminal can be left open.

The N.O. terminal of the relay gets connected to the output of the device that it powers on or off. In this case, we are using a LED as output. When the relay receives 5V of power in its coil, the LED will light up. If the 5v are cut off from it, the LED will no longer light.

2.10.3 HOW TO CONNECT A DPDT RELAY IN A CIRCUIT

In order to know how to connect a DPDT relay, you must know what each pin terminal represents and how the relay works.

Terminal Pins

A Double Pole Double Throw Relay comes with 8 terminal points. The terminals are COIL, COIL, COM, COM, NO, NO, NC, NC.

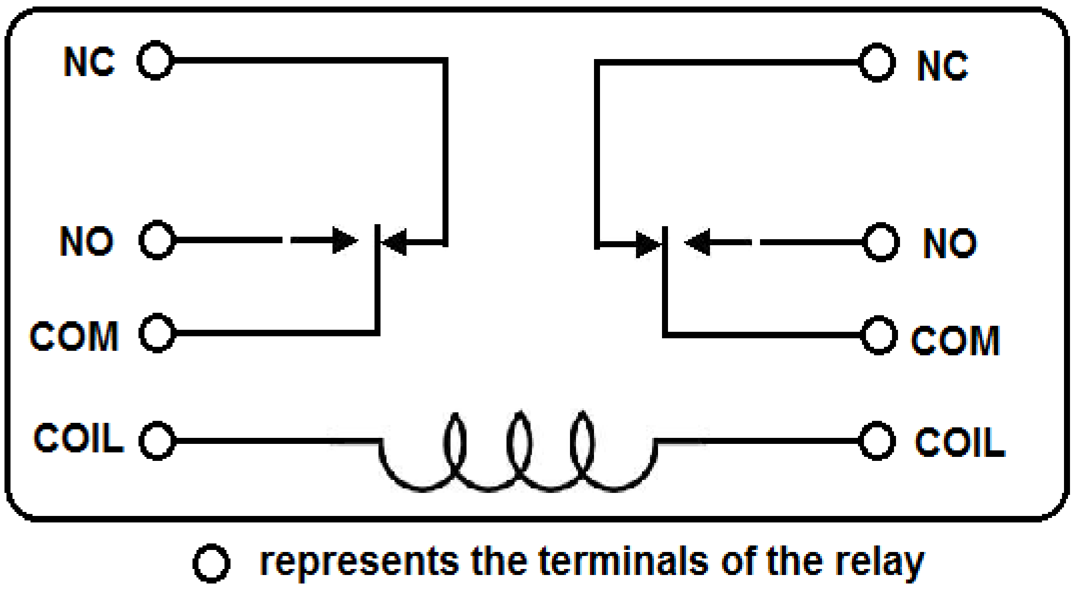


Fig 2.10.3.1 Terminals of DPDT Relay

This correlates to the following in the relay:

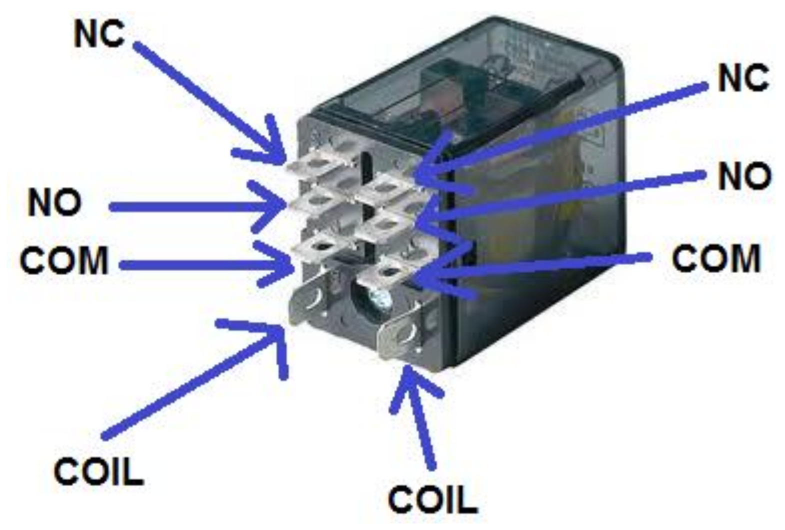


Fig 2.10.3.2 DPDT Relay pins

Terminal Descriptions

COIL- This the is the COIL terminal. These are the terminals where you apply voltage to in order to give power to the coils (which then will close the switch). Polarity does not matter. One side gets positive voltage and the other side gets negative voltage. It doesn't matter which order. Polarity only matters if a diode is used.

NO- This is Normally Open switch. This is the terminal where you connect the device that you want the relay to power, when the relay is powered, meaning when the COIL receives sufficient voltage. The device connected to NO will be off when the relay has no power and will turn on when the relay receives power.

NC- This is the Normally Closed Switch. This is the terminal where you connect the device that you want powered when the relay receives no power. The device connected to NC will be on when the relay has no power and will turn off when the relay receives power.

COM- This is the common of the relay. If the relay is powered and the switch is closed, COM and NO have continuity. If the relay isn't powered and the switch is open, COM and NC have continuity. This is the terminal of the relay where you connect the first part of your circuit to.

Now that we know what each terminal pin represents, we now wire it to a circuit for it to do a real-world function. We're going to connect a Double pole double throw relay to a circuit to light up LEDs. When the relay isn't powered, both the red LED and the DC fan are on,. When the relay is powered, the red LED and the fan shut off and the green LED and the DC motor turn on.

This is the circuit below:

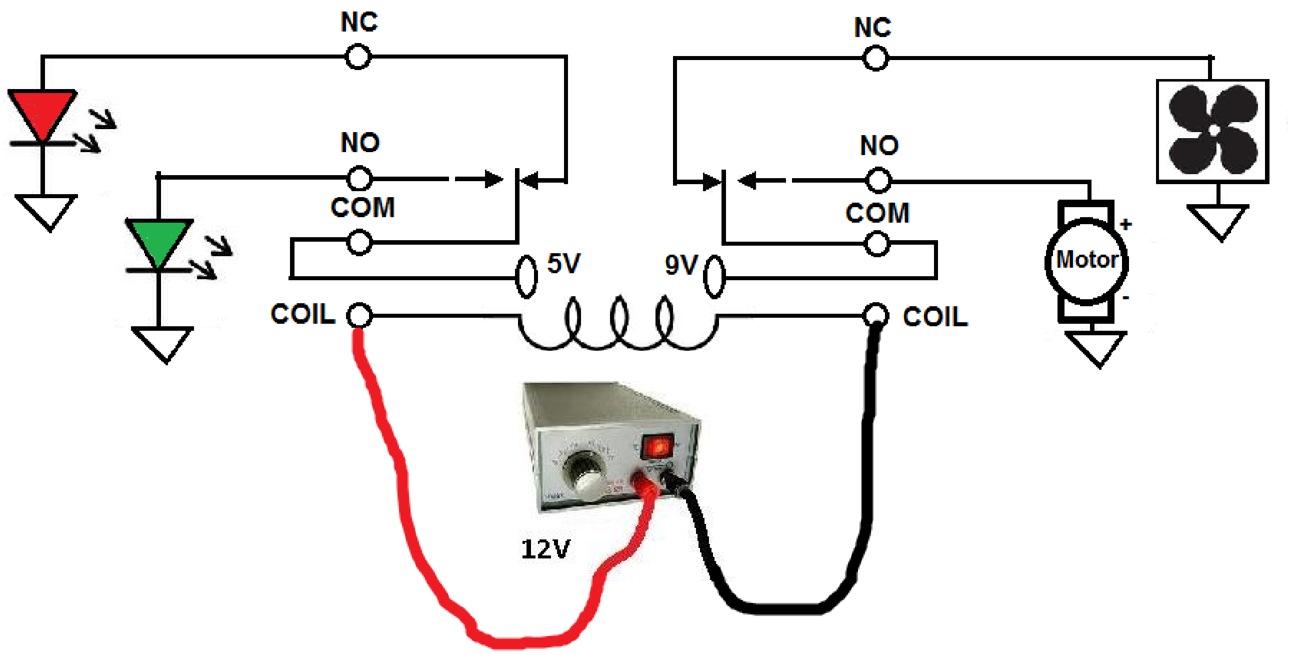


Fig 2.10.3.3 Circuit Diagram of DPDT Relays Connection

Since the relay is rated for 12V, it should receive 12 volts in order to power on. It may work with less voltage, but 12V is really what it should receive. This goes into either side of the COIL terminals. Even if you switched the positive and negative voltage of the power supply, it should work exactly the same.

The COM terminals of the relay get connected to the first part of the circuit. If there is no first part of the circuit, this terminal can be left open. In this case, the first part of the circuit is the 5-volt power supply and the 9-volt power supply to light the LEDs and the DC fan and DC motor. The NC terminals of the relay get power even when the relay isn't powered. This means that as long as the 5-volt power supply is on, the red LED and the DC fan will be on and operating.

The NO terminals of the relay get power only when the relay is powered. When the relay receives 12 volts of power, the relay snaps from the NC position to the NO position. The red LED and the DC fan now shut off and the green LED and the DC motor now turn on and operate.

SPDT RELAY

The Single Pole Double Throw SPDT relay is quite useful in certain applications because of its internal configuration. It has one common terminal and 2 contacts in 2 different configurations: one can be Normally Closed and the other one is opened or it can be Normally Open and the other one closed. So basically you can see the SPDT relay as a way of switching between 2 circuits: when there is no voltage applied to the coil one circuit “receives” current, the other one doesn’t and when the coil gets energized the opposite is happening.

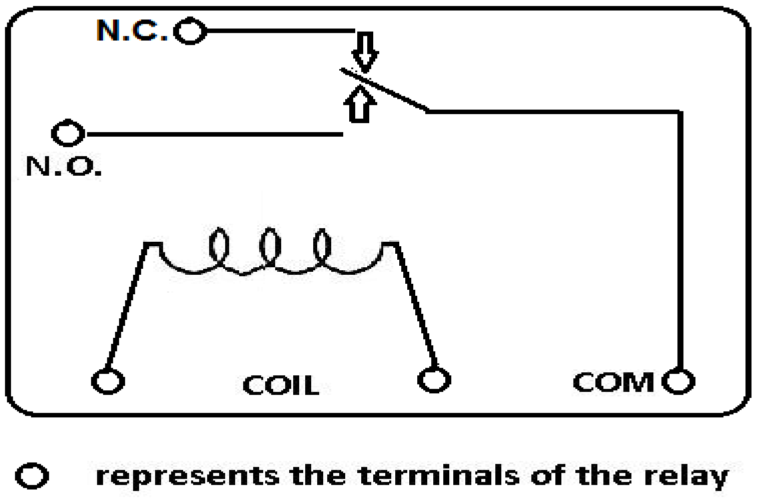


Fig 2.10.3.4: SPDT Relay

How does the Single Pole Double Throw relay works?

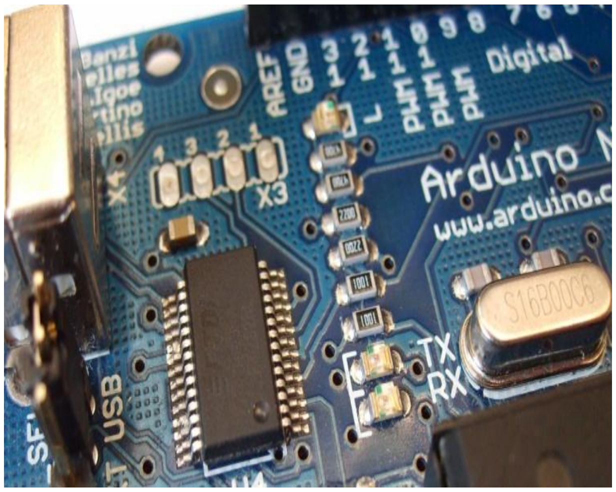
No DC voltage is applied to the coil so the terminal T is connected to contact 1 therefore the current can flow through 1 and it cannot flow through 2.

When DC voltage is applied to the coil and terminal T is now connected to contact 2 therefore the current doesn’t flow anymore through 1 but now it flows through 2.

Chapter 3

SOFTWARE REQUIREMENTS

3.1 ARDUINO IDE:



Depending on what type of Arduino you have, you may also have a USB connector to enable it to be connected to a PC or Mac to upload or retrieve data. The board exposes the microcontroller’s I/O (Input/Output) pins to enable you to connect those pins to other circuits or to sensors, etc.

Fig 3.1.1 ARDUINO board

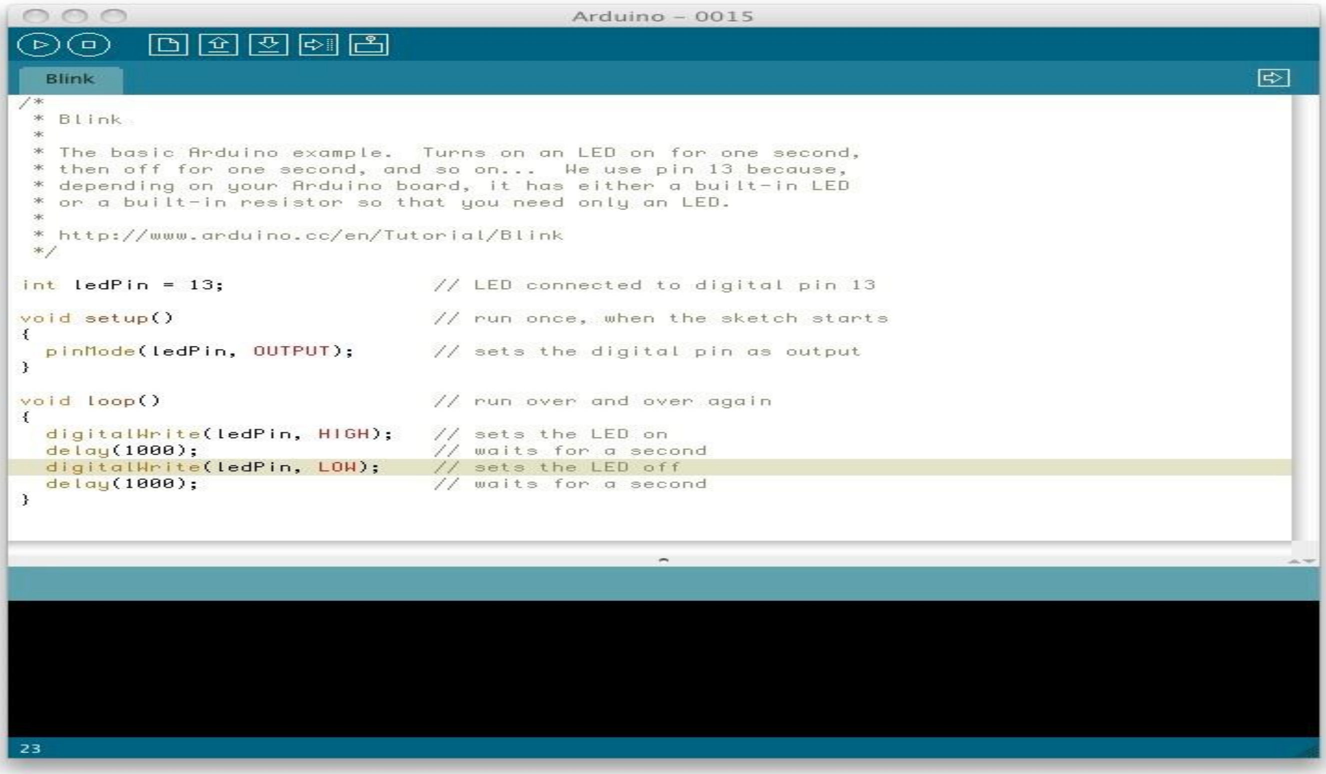
Now that you are a proud owner of an Arduino, or an Arduino clone, it might help if you knew what it was and what you can do with it.

In its simplest form, an Arduino is a tiny computer that you can program to process inputs and outputs going to and from the chip.

The Arduino is what is known as a Physical or Embedded Computing platform, which means that it is an interactive system that through the use of hardware and software can interact with it’s environment.

For example, a simple use of the Arduino would be to turn a light on for a set period of time, let’s say 30 seconds, after a button has been pressed (we will build this very same project later in the book). In this example, the Arduino would have a lamp connected to it as well as a button. The Arduino would sit patiently waiting for the button to be pressed. When you press the button it would then turn the lamp on and start counting. Once it had counted 30 seconds it would then turn the lamp off and then carry on sitting there waiting for another button press. You could use this set-up to control a lamp in an under-stairs cupboard for example. You could extend this example to sense when the cupboard door was opened and automatically turn the light on, turning it off after a set period of time. The Arduino can be used to develop stand-alone interactive objects or it can be connected to a computer to retrieve or send data to the Arduino and then act on that data (e.g. Send sensor data out to the internet). The Arduino can be connected to LED's. Dot Matrix displays, LED displays, buttons, switches, motors, temperature sensors, pressure sensors, distance sensors, webcams, printers, GPS receivers, ethernet modules,

The Arduino board is made of an an Atmel AVR Microprocessor, a crystal or oscillator (basically a crude clock that sends time pulses to the microcontroller to enable it to operate at the correct). To program the Arduino (make it do what you want it to) you also use the Arduino IDE (Integrated Development Environment), which is a piece of free software, that enables you to program in the language that the Arduino understands. In the case of the Arduino the language is C. The IDE enables you to write a computer program, which is a set of step-by- step instructions that you then upload to the Arduino. Then your Arduino will carry out those



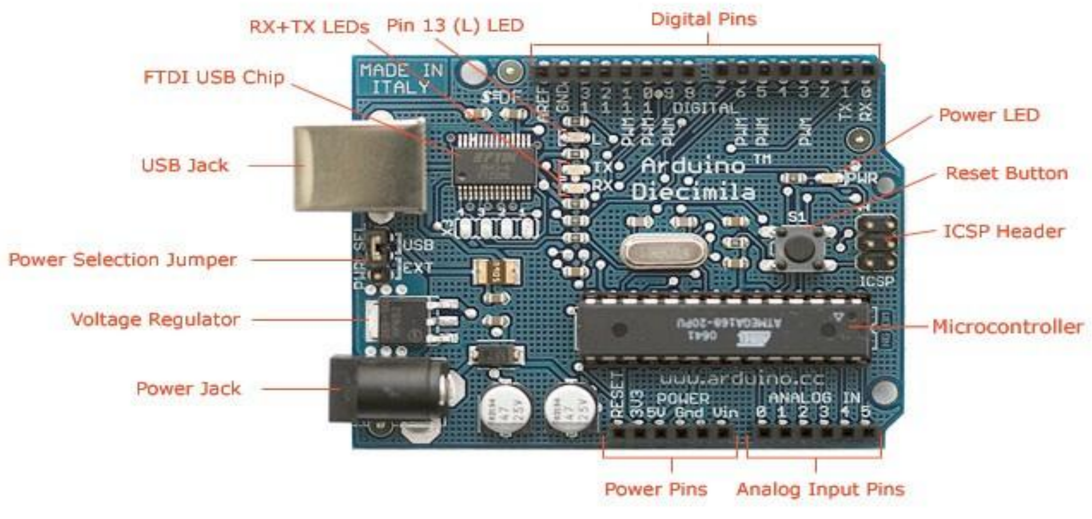
```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */

int ledPin = 13;                // LED connected to digital pin 13

void setup()                    // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()                     // run over and over again
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Fig 3.1.2 ARDUINO IDE

instructions and interact with the world outside. In the Arduino world, programs are known as sketches.

The Arduino hardware and software are both Open Source, which means the code, the schematics, design, etc. are all open for anyone to take freely and do what they like with it. This means there is nothing stopping anyone from taking the schematics and PCB designs of the Arduino and making their own and selling them. This is perfectly legal, and indeed the whole purpose of Open Source, and indeed the Fredonia that comes with the Earthshine Design Arduino Starter Kit is a perfect example of where someone has taken the Arduino PCB design, made their own and are selling it under the Fredonia name. You could even make your own Arduino, with just a few cheap components, on a breadboard.

The only stipulation that the Arduino development team put on outside developers is that the Arduino name can only be used exclusively by them on their own products and hence the clone boards have names such as Fredonia, Boarding, Borodino, etc. As the designs are open



Photograph by SparkFun Electronics. Used under the Creative Commons Attribution Share-Alike 3.0 license.

Fig 3.1.3 ARDUINO board

source, any clone board, such as the Fredonia, is 100% compatible with the Arduino and therefore any software, hardware, shields, etc. will all be 100% compatible with a genuine Arduino. The Arduino can also be extended with the use of shields which are circuit boards containing other devices (e.g. GPS receivers, LCD Displays, Ethernet connections, etc.) that you can simply slot into the top of your Arduino to get extra functionality. You don’t have to use a shield if you don’t want to as you can make the exact same circuitry using a breadboard, some veroboard or even by making your own PCB's.

There are many different variants of the Arduino available. The most common one is the Diecimila or the Duemilanove. You can also get Mini, Nano and Bluetooth Arduino's. New to the product line is the new Arduino Mega with increased memory and number of I/O pins. Probably the most versatile Arduino, and hence the reason it is the most popular, is the Duemilanove. This is because it uses a standard 28 pin chip, attached to an IC Socket. The beauty of this systems is that if you make something neat with the Arduino and then want to turn it into something permanent (e.g. Or under- stairs cupboard light), then instead of using the relatively expensive Arduino board, you can simply use the Arduino to develop your device, then pop the chip out of the board and place it into your own circuit board in your custom device. You would then have made a custom embedded device, which is really cool.

Then, for a couple of quid or bucks you can replace the AVR chip in your Arduino with a new one. The chip must be pre-programmed with the Arduino Bootloader to enable it to work with the Arduino IDE, but you can either burn the Bootloader yourself if you purchase an AVR Programmer, or you can buy these pre- programmed from many suppliers around the world. Of course, Earthshine Design provide pre- programmed Arduino chips in it' store for a very reasonable price.

If you do a search on the Internet by simply typing 'Arduino' into the search box of your favourite search engine, you will be amazed at the huge amount of websites dedicated to the Arduino. You can find a mind boggling amount of information on projects made with the Arduino and if you have a project in mind, will easily find information that will help you to get your project up and running easily.

The Arduino is an amazing device and will enable you to make anything from interactive works of art to robots. With a little enthusiasm to learn how to program the Arduino and make it interact with other components a well as a bit of imagination, you can build anything you want. This book and the kit will give you the necessary skills needed to get started in this exciting and creative hobby. So, now you know what an Arduino is and what you can do with it, let's open up the starter kit and dive right in.

Chapter 4

WORKING

4.1 Getting Started

This section will presume you have a PC running Windows or a Mac running OSX (10.3.9 or later). If you use Linux as your Operating System.

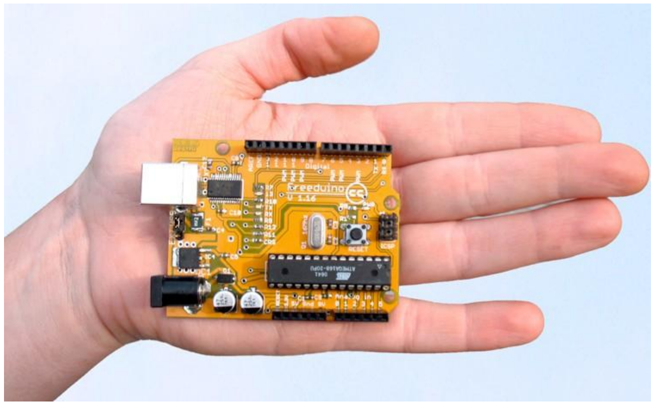


Fig 4.1.1 Freeduino Board

Get the Freeduino and the USB Cable

Firstly, get your Freeduino board and lay it on the table in front of you. Take the USB cable and plug the B plug (the fatter squarer end) into the USB socket on the Freeduino.

Fig 4.1.2 USB cable

At this stage do NOT connect the Freeduino to your PC or Mac yet.

Download the Arduino IDE

Download the Arduino IDE from the Arduino download page. As of the time of writing this book, the latest IDE version is 0015. The file is a ZIP file so you will need to uncompress it. Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is and do not make any changes.

If you double-click the folder, you will see a few files and sub-folders inside.

Install the USB Drivers

If you are using Windows you will find the drivers in the drivers/FTDI USB Drivers directory of the Arduino distribution. In the next stage ("Connect the Freeduino"), you will point W i n d o w ' s Add New Hardware wizard to these drivers.

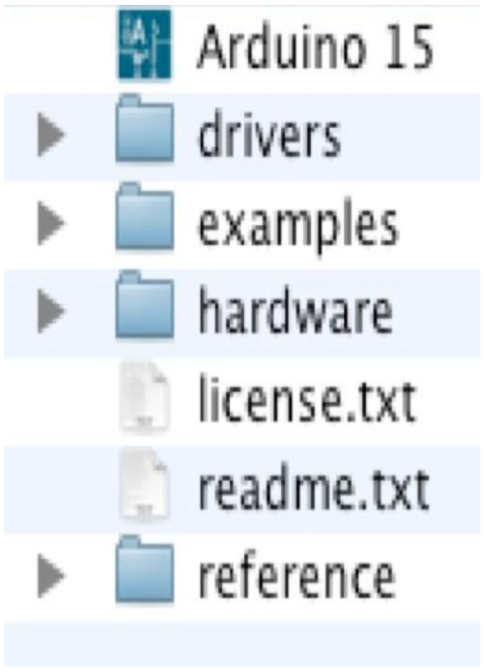


Fig 4.1.3 USB drivers

If you have a Mac these are in the drivers directory. If you have an older Mac like a PowerBook, iBook, G4 or G5, you should use the PPC drivers:

FTDIUSBSerialDriver_v2_1_9.dmg. If you have a newer Mac with an Intel chip, you need the Intel drivers: FTDIUSBSerialDriver_v2_2_9_Intel.dmg. Double-click to mount the disk image and run the included FTDIUSBSerialDriver.pkg.

The latest version of the drivers can be found on the FTDI website.

Connect the Freeduino:

First, make sure that the little power jumper, between the power and USB sockets, is set to USB and not EXTernal power (not applicable if you have a Roboduino board, which has an Auto Power Select function).

Using this jumper you can either power the board from the USB port (good for low current devices like LED's, etc.) or from an external power supply (6-12V DC).

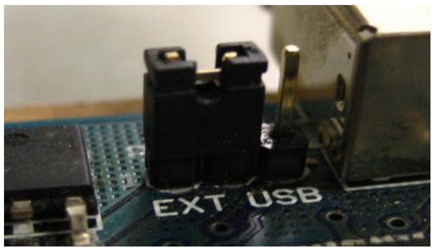
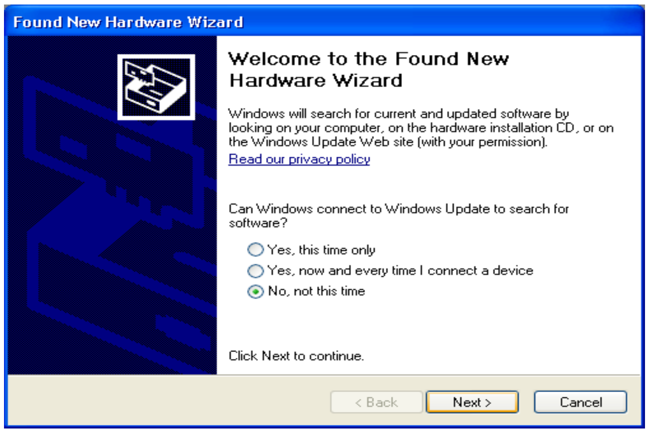


Fig 4.1.4: Roboduino Board

Now, connect the other end of the USB cable into the USB socket on your PC or Mac. You will now see the small power LED (marked PWR above the RESET switch) light up to show you have power to the board.

If you have a Mac, this stage of the process is complete and you can move on to the next Chapter. If you are using Windows, there are a few more steps to complete (Damn you Bill Gates!).


Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

Read our privacy policy

Can Windows connect to Windows Update to search for software?

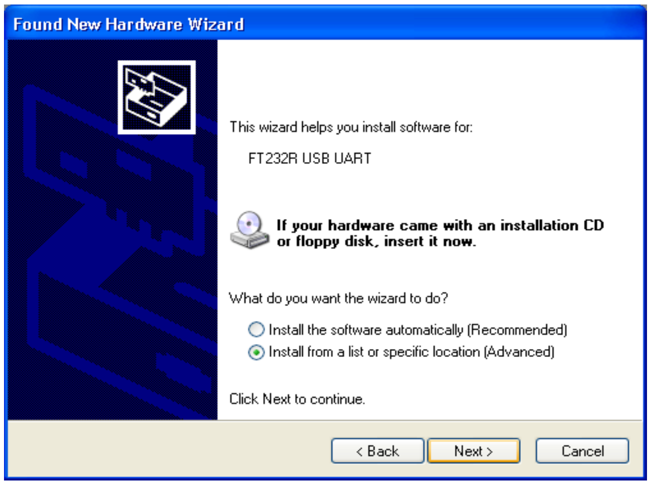
- Yes, this time only
- Yes, now and every time I connect a device
- [✓ No, not this time]

Click Next to continue.

Fig 4.1.5 Found New Hardware

On Windows the **Found New Hardware Wizard** will now open up as Windows will have detected that you have connected a new piece of hardware (your Freeduino board) to your PC. Tell it NOT to connect to Windows update (Select **No, not at this time**) and then click **Next**.

On the next page select "**Install from a list or specific location (Advanced)**" and click **Next**.


Found New Hardware Wizard

This wizard helps you install software for:

FT232R USB UART

If your hardware came with an installation CD or floppy disk, insert it now.

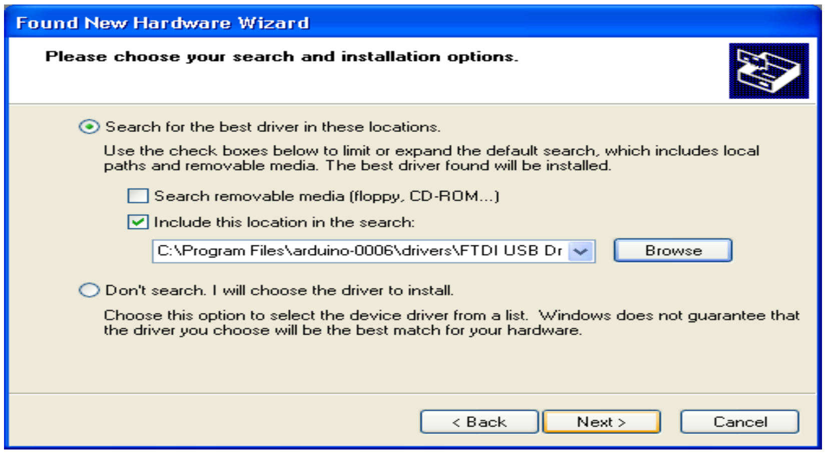
What do you want the wizard to do?

( ) Install the software automatically (Recommended)
(●) Install from a list or specific location (Advanced)

Click Next to continue.

Fig 4.1.6: Install Drivers

Make sure that "**Search for the best driver in these locations**" is checked. Uncheck "**Search removable media**". Check "**Include this location in the search**" and then click the **Browse**


Found New Hardware Wizard

Please choose your search and installation options.

(●) Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

[ ] Search removable media (floppy, CD-ROM...)
[✓] Include this location in the search:

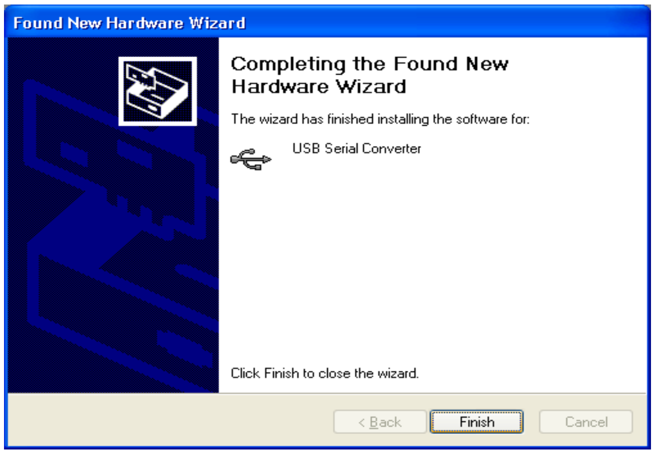
C:\Program Files\arduino-0006\drivers\FTDI USB Dr

( ) Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Fig 4.1.7 Browse

button. Browse to the location of the USB drivers and then click **Next**. The wizard will now search for a suitable driver and then tell you that a "USB Serial Convertor" has been found and that the hardware wizard is now complete. Click Finish.


Found New Hardware Wizard
Completing the Found New Hardware Wizard
< Back
Finish
Cancel

The wizard has finished installing the software for:

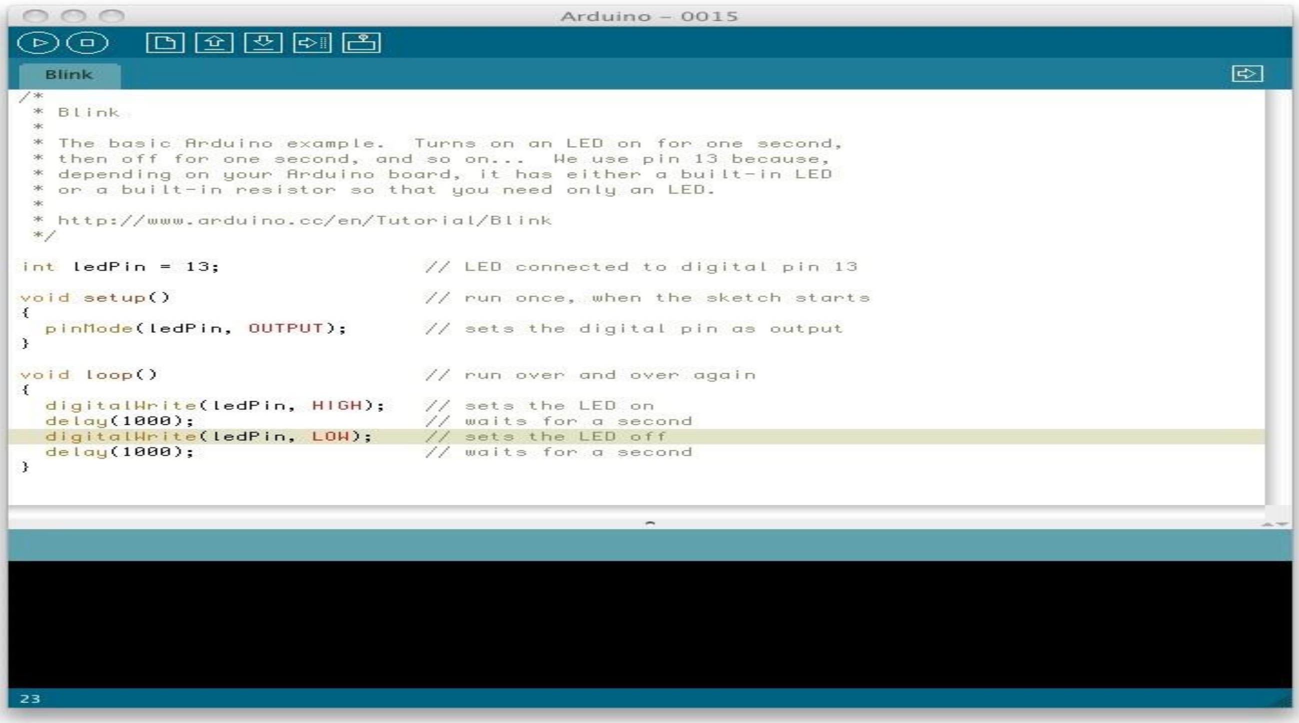
USB Serial Converter

Click Finish to close the wizard.

Fig 4.1.8 Finish

You are now ready to upload your first Sketch.

4.2 Upload your first Sketch:



```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */

int ledPin = 13;                // LED connected to digital pin 13

void setup()                    // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()                     // run over and over again
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Fig 4.2.1 Uploading Program

you will see the Sketch inside the white code window.

Now, before we upload the Sketch, we need to tell the IDE what kind of Arduino we are using and the details of our USB port. Go to the file menu and click Tools, then clock on Board. You will be presented with a list of all of the different kinds of Arduino board that can be connected to the IDE. Our Freeduino board will either be fitted with an Atmega328 or an Atmega168 chip so choose "Arduino Duemilanove w/ATmega328" if you have a 328 chip or "Arduino Diecimila or Duemilanove w/ ATmega328P" if you have a 328 chip.

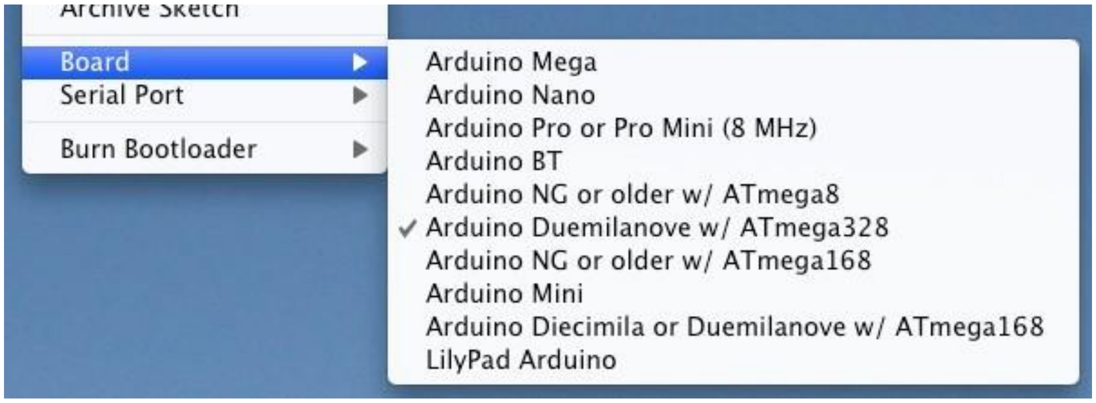

Archive Sketch
Board
Serial Port
Burn Bootloader
Arduino Mega
Arduino Nano
Arduino Pro or Pro Mini (8 MHz)
Arduino BT
Arduino NG or older w/ ATmega8
✓ Arduino Duemilanove w/ ATmega328
Arduino NG or older w/ ATmega168
Arduino Mini
Arduino Diecimila or Duemilanove w/ ATmega168
LilyPad Arduino

Fig 4.2.2 Board Selection

Now that your Freeduino has been connected and the drivers for the USB chip have been installed, we are now ready to try out the Arduino for the first time and upload your first Sketch Navigate to your newly unzipped Arduino folder and look for the Arduino IDE icon, which looks something like this....

Double click the ICON to open up the IDE. You will then be presented with a blue and white screen with a default sketch loaded inside.

This is the Arduino IDE (Integrated Development Environment) and is where you will write your Sketches (programs) to upload to your Arduino board.

We will take a look at the IDE in a little more detail in the next chapter. For now, simply click File in the file menu and scroll down to Sketchbook. Then scroll down to Examples and click it. You will be presented with a list of Example sketches that you can use to try out your Arduino. Now click on Digital and inside there you will find an example Sketch called Blink. Click on this. The Blink Sketch will now be loaded into the IDE and

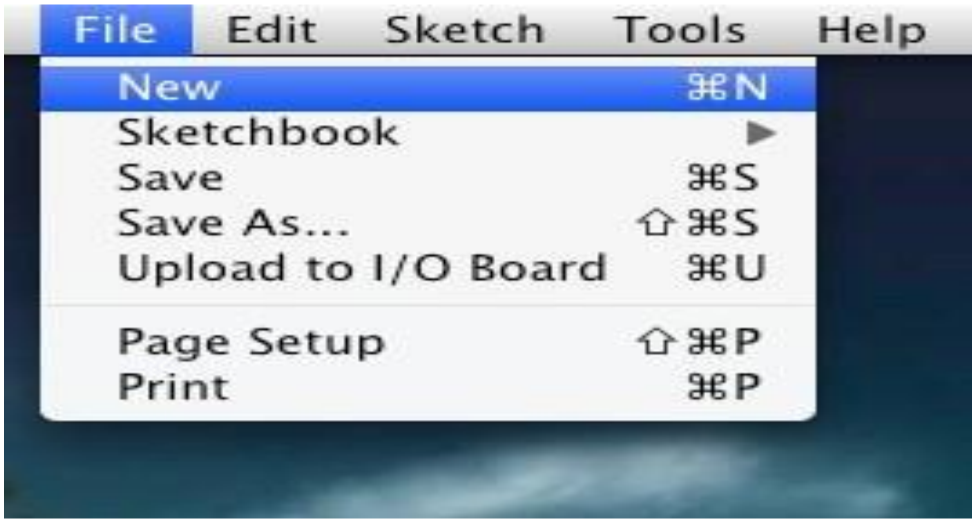


Fig 4.2.3 New Program

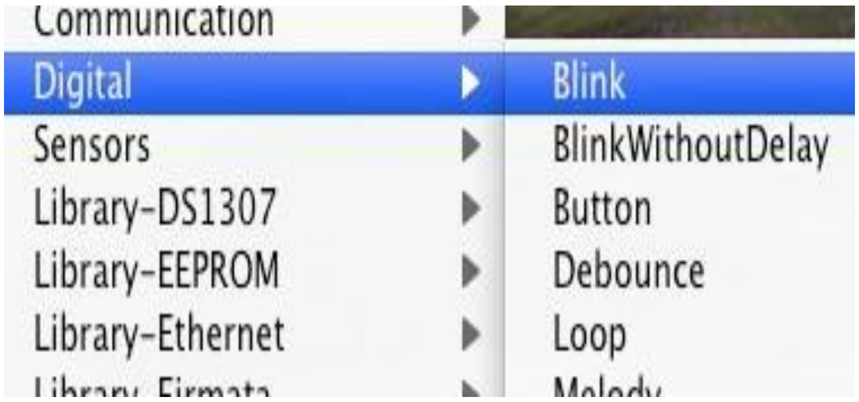


Fig 4.2.4 Blink the Digital output

Now you need to tell the IDE the details of your USB port, so now click on Tools again, scroll down to Serial Port and a list of the available serial ports on your system will be displayed. You need to choose the one that refers to your USB cable, which is usually listed as something like /dev/tty.usbserial-xxxx on a Mac or something like Com 4 on Windows so click on that. If not sure, try each one till you find one that works.

Now that you have selected the correct board and USB port you are ready to upload the Blink Sketch to the board.

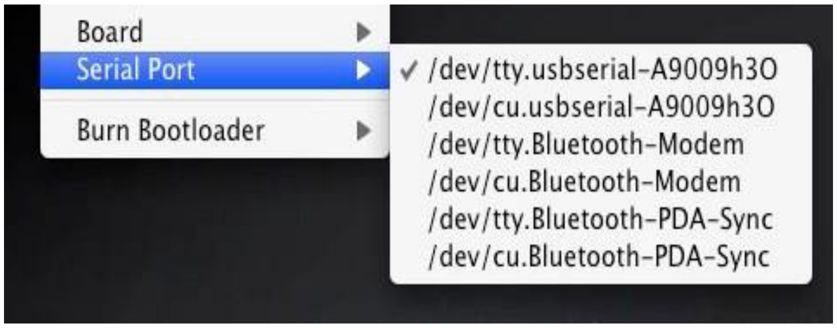

Board
Serial Port
Burn Bootloader
✓ /dev/tty.usbserial-A9009h3O
/dev/cu.usbserial-A9009h3O
/dev/tty.Bluetooth-Modem
/dev/cu.Bluetooth-Modem
/dev/tty.Bluetooth-PDA-Sync
/dev/cu.Bluetooth-PDA-Sync

Fig 4.2.5 Port Selection

You can either click the Upload button, which is the 6th button from the left at the top with an arrow pointing to the right (hover your mouse pointer over the buttons to see what they are) or by clicking on File in the file menu and scrolling down to Upload to I/O Board and clicking on that.

Presuming everything has been set up correctly you will now see the RX and TX LED's (and also LED 13) on the Freeduino flash on and off very quickly as data is uploaded to the board. You will see *Uploading to I/O Board....* Once the data has been uploaded to the board successfully you will get a Done Uploading message in the IDE and the RX/TX LED's will stop flashing. The Arduino will now reset itself and immediately start to run the Sketch that you have just uploaded.
Just below the code window too.

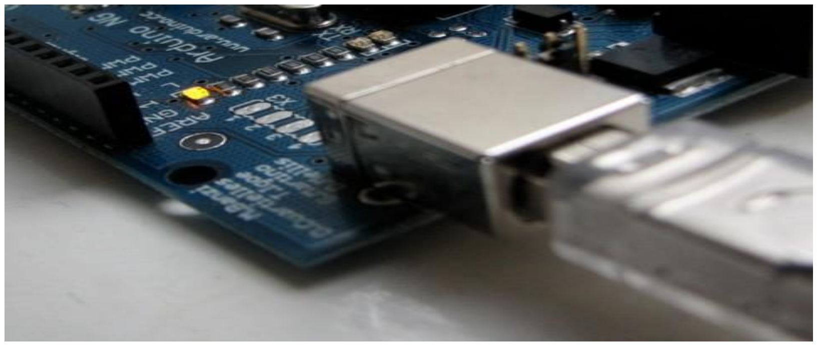
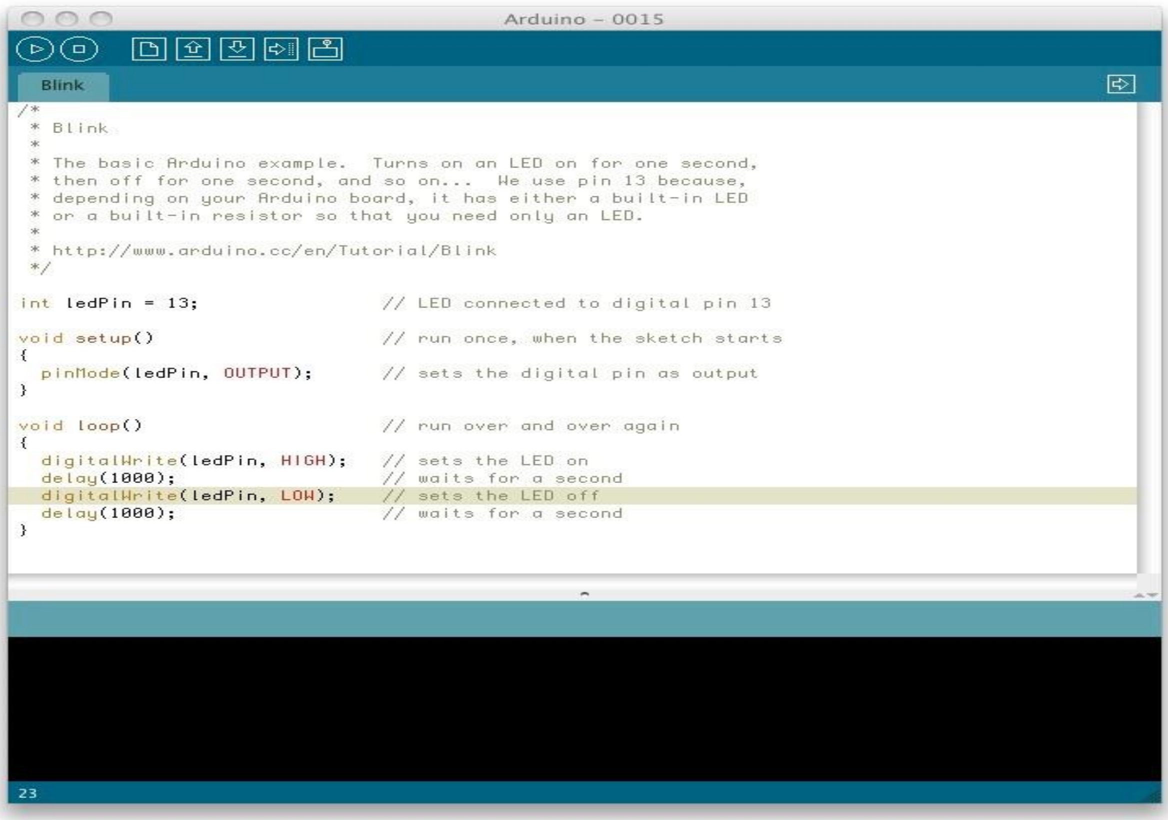


Fig 4.2.6 USB plug in

The Blink sketch is a very simple sketch that blinks LED 13, which is a tiny green LED soldered to the board and also connected to Digital Pin 13 from the Microcontroller, and will make it flash on and off every 1000 milliseconds, or 1 second. If your sketch has uploaded successfully, you will now see this LED happily flashing on and off slowly on your board.

If so, congratulations, you have just successfully installed your Arduino, uploaded and ran your first sketch.

We will now explain a bit more about the Arduino IDE and how to use it before moving onto the projects that you can carry out using the hardware supplied with the kit. For our first project we will carry out this Blink LED sketch again, but this time using an LED that we will physically connect to one of the digital output pins on the Arduino. We will also explain the hardware and software involved in this simple project. But first, let's take a closer look at the Arduino IDE.



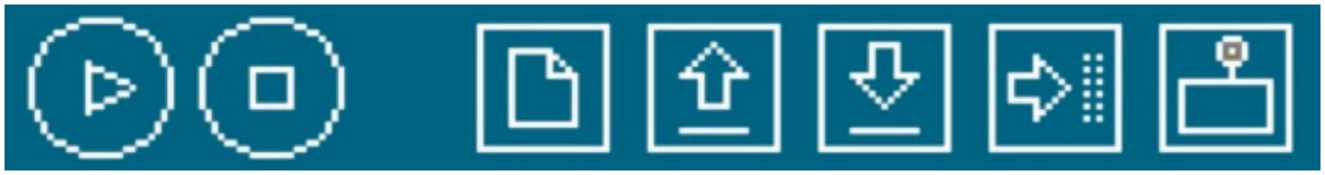
```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */

int ledPin = 13;                // LED connected to digital pin 13

void setup()                    // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()                     // run over and over again
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Fig 4.2.7 Arduino LED off

When you open up the Arduino IDE it will look very similar to the image above. If you are using Windows or Linux there will be some slight differences but the IDE is pretty much the same no matter what OS you are using.

The IDE is split up into the Toolbar across the top, the code or Sketch Window in the centre and the Serial Output window at the bottom. The Toolbar consists of 7 buttons, underneath the Toolbar is a tab, or set of tabs, with the filename of the code within the tab. There is also one further button on the far right hand side.

Along the top is the file menu with drop down menus headed under File, Edit, Sketch, Tools and Help. The buttons in the Toolbar provide convenient access to the most commonly used functions within this file menu.



Compile Stop New Open Save Upload Serial Monitor

Verify/ Fig 4.2.8 IDE Options

The Toolbar buttons are listed above. The functions of each button are as follows:-

Verify/Compi	Checks the code for errors
Stop	Stops the serial monitor, or un-highlights other
New	Creates a new blank Sketch
Open	Shows a list of Sketches in your sketchbook
Save	Saves the current Sketch
Upload	Uploads the current Sketch to the Arduino
Serial	Displays serial data being sent from the Arduino

Table 3.2 COMMANDS

The **Verify/Compile** button is used to check that your code is correct, before you upload it to your Arduino.

The **Stop** button will stop the Serial Monitor from operating. It will also un-highlight other selected buttons. Whilst the Serial Monitor is operating you may wish to press the Stop button to obtain a 'snapshot' of the serial data so far to examine it. This is particularly useful if you are sending data out to the Serial Monitor quicker than you can read it.

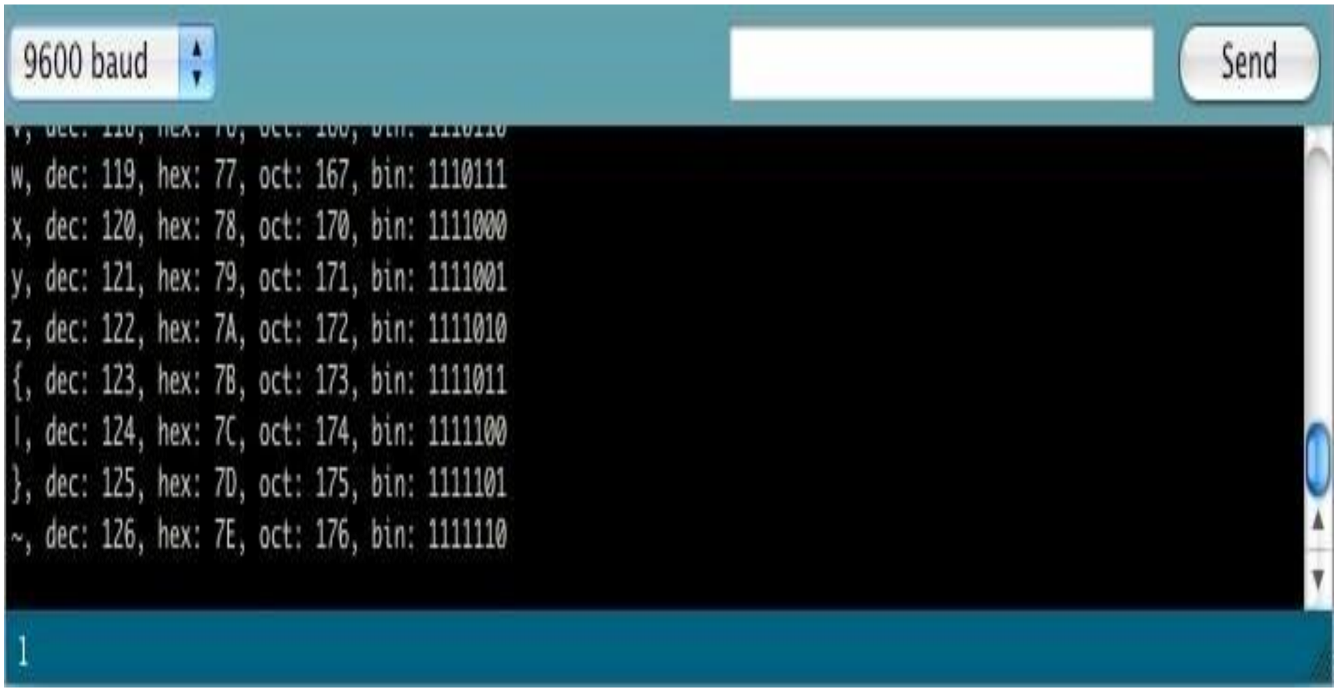
The **New** button will create a completely new and blank Sketch read for you to enter code into. The IDE will ask you to enter a name and a location for your Sketch (try to use the default location if possible) and will then give you a blank Sketch ready to be coded. The tab at the top of the Sketch will now contain the name you have given to your new sketch.
The **Open** button will present you with a list of Sketches stored within your sketchbook as well as a list of Example sketches you can try out with various peripherals once connected.

The **Save** button will save the code within the sketch window to your sketch file. Once complete you will get a 'Done Saving message at the bottom of the code window.

The **Upload to I/O Board** button will upload the code within the current sketch window to your Arduino. You need to make sure that you have the correct board and port selected (in the Tools menu) before uploading. It is essential that you Save your sketch before you upload it to your board in case a strange error causes your system to hang or the IDE to crash. It is also advisable to Verify/Compile the code before you upload to ensure there are no errors that need to be debugged first.

The **Serial Monitor** is a very useful tool, especially for debugging your code. The monitor displays serial data being sent out from your Arduino (USB or Serial board). You can also send serial data back to the Arduino using the Serial Monitor. If you click the Serial Monitor button you will be presented with an image like the one above.

On the left hand side you can select the Baud Rate that the serial data is to be sent to/from the Arduino. The Baud Rate is the rate, per second, that characters (data) is sent to/from the board. The default setting is 9600 baud, which means that if you were to send a text novel over the serial communications line (in this case your USB cable) then 9600 letters, or symbols, of the novel, would be sent per second.



```
w, dec: 119, hex: 77, oct: 167, bin: 1110111
x, dec: 120, hex: 78, oct: 170, bin: 1111000
y, dec: 121, hex: 79, oct: 171, bin: 1111001
z, dec: 122, hex: 7A, oct: 172, bin: 1111010
{, dec: 123, hex: 7B, oct: 173, bin: 1111011
|, dec: 124, hex: 7C, oct: 174, bin: 1111100
}, dec: 125, hex: 7D, oct: 175, bin: 1111101
~, dec: 126, hex: 7E, oct: 176, bin: 1111110
```

Fig: 4.2.9 Set Baud Rate

To the right of this is a blank text box for you to enter text to send back to the Arduino and a **Send** button to send the text within that field. Note that no serial data can be received by the Serial Monitor unless you have set up the code inside your sketch to do so. Similarly, the Arduino will not receive any data sent unless you have coded it to do so.

Finally, the black area is where your serial data will be displayed. In the image above, the Arduino is running the ASCIITable sketch, that can be found in the Communications examples. This program outputs ASCII characters, from the Arduino via serial (the USB cable) to the PC where the Serial monitor then displays them.

To start the Serial Monitor press the Serial Monitor button and to stop it press the Stop button. On a Mac or in Linux, Arduino board will reset itself (rerun the code from the beginning) when you click the Serial Monitor button.

Once you are proficient at communicating via serial to and from the Arduino you can use other programs such as Processing, Flash, MaxMSP, etc. To communicate between the Arduino and your PC.

We will make use of the Serial Monitor later on in our projects when we read data from sensors and get the Arduino to send that data to the Serial Monitor, in human readable form, for us to see.

The Serial Monitor window is also were you will see error messages (in red text) that the IDE will display to you when trying to connect to your board, upload code or verify code.

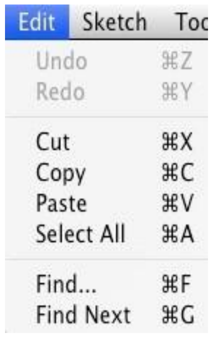
Below the Serial Monitor at the bottom left you will see a number. This is the current line that the cursor, within the code window, is at. If you have code in your window and you move down the lines of code (using the ↓ key on your keyboard) you will see the number increase as you move down the lines of code. This is useful for finding bugs highlighted by error messages. Across the top of the IDE window (or across the top of your screen if you are using a Mac) you will see the various menus that you can click on to access more menu items.

Fig 4.2.10 Toolbar

The menu bar across the top of the IDE looks like the image above (and slightly different in Windows and Linux). I will explain the menus as they are on a Mac, the details will also apply to the Windows and Linux versions of the IDE.

The first menu is the **Arduino menu.** Within this is the **About Arduino** option, which when pressed will show you the current version number, a list of the people involved in making this amazing device and some further information.

Underneath that is the **Preferences** option.The next menu is the **File** menu. In here you get access to options to create a New sketch, take a look at Sketches stored in your Sketchbook (as well as the Example Sketches), options to Save your Sketch (or Save As if you want to give it a different name). You also have the option to upload your sketch to the I/O Board (Arduino) as well as the Print options for printing out your code.Next is the **Edit** menu. In here you get options to enable you to Cut, Copy and Paste sections of code. Select All of your code as well as Find certain words or phrases within the code. Also included are the useful Undo and Redo options which come in handy when you make a mistake.



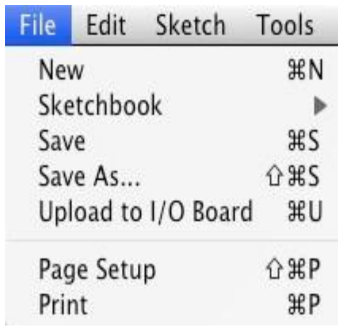


Fig 4.2.11: Edit menu Fig 4.2.12: File Menu

Our next menu is the **Sketch** menu which gives us access to the Verify/Compile functions and some other useful functions you will use later on. These include the Import Library option, which when clicked will bring up a list of the available libraries, stored within your The next menu in the IDE is the **Tools** menu. Within this are the options to select the Board and Serial Port we are using, as we did when setting up the Arduino for the first time. Also we have the Auto Format function that formats your code to make it look nicer.

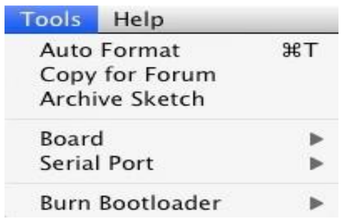


Fig 4.2.13 Tools menu

The Copy for Forum option will copy the code within the Sketch window, but in a format that when pasted into the Arduino forum (or most other Forums for that matter) will show up the same as it is in the IDE, along with syntax colouring, etc.

The Archive Sketch option will enable you to compress your sketch into a ZIP file and asks you were you want to store it.

As Ethernet provides inexpensive, relatively high speed network access to individual users and low delay that can support many applications. Ethernet continues to be enhanced with greater performance, higher determinism, and lower cost implementations and even consolidate control network applications. A real web server is implemented in a device in your own home, which is connected to your pc via a local area network. If we compared Ethernet Technology with other technologies like Bluetooth, Zig-bee, IR, RF-ID and GSM, it is having low response time, having very high speed, secured and also reliable.

Chapter 5

APPLICATIONS

- Home automation – This project can be used to control various Home Appliances
- This principle can be used anywhere even in industries for monitoring purposes.
- Mainly useful for Old People who can't afford Physical strain.
- Used mainly in energy conservation required places.
- Used in Industries especially large scale industries where it is easy for monitoring the loads using local area connectivity.
- Used in Traffic lights control in maintaining smart lights based on traffic density.
- Mainly used in remote monitoring and controlling of loads from anywhere network available places.

Chapter 6

ADVANTAGES & LIMITATIONS

ADVANTAGES:

- ✓ Increased comfort, convenience and security as we are using LAN cable
- ✓ Reduced energy consumption since low power is enough for it.
- ✓ Maximum flexibility due to battery-free, wireless devices like laptop can also easily handle it.
- ✓ We can manage a HTML page by hosting IP address
- ✓ Operation of the Project is easily understandable to anyone
- ✓ Easy to use

LIMITATIONS:

- ✗ We cannot see whether the bulb is really on based on feedback in webpage
- ✗ Any person who knows IP address can access the server if hosted online.

Chapter 7
FUTURE SCOPE

The Project can be developed furthermore by improving some of its features as below:

- The Home automation Devices should be disseminated across the globe for easier access.
- The IoT devices Should be embedded with Fingerprint for Biometric authentication of data for security purposes.
- Ethernet consumes power even when no data is transmitted. So, to reduce this power, we have specified a low-power state that puts the link into idle when there is no traffic.
- Use of Ethernet switch to handle inter-Virtual Machines traffic, instead of Virtual Switch, freeing up the server to run its actual application workloads. It gives data center operators visibility of all network traffic, including traffic between VMs, via the physical network switch.
- Ethernet Passive Optical Networks (EPON) are the emerging technology in access networks that provides a low-cost method of deploying optical access lines between link
- The Power Over Ethernet (POE), technology which uses spare Ethernet wire to deliver electrical power over a cable to the connected devices.
- We can see whether the bulb is really on based on feedback in webpage

Chapter 8
CONCLUSION

In this paper we presented concepts and a prototype system for “IoT based home automation using arduino via Ethernet” which can fit into a home appliance using Ethernet. Internet enabled hardware products are slowly becoming common place. Ethernet’s potential as a network for distributed measurement and control is virtually unlimited. The major advantage of this design implementation is the consumption of low power, secured and simple.

Chapter 9
REFERENCES

[1] Ryan J.L, “Home automation”, Electronics & Communication Engineering IEEE Journal, volume 1, issue 4, Page 185 – 192, 2002.

[2] Inoue M, Uemura K, Minagawa Y, Esaki, Mitsunobu, Honda Y, “A Home Automation System”, Consumer Electronics, IEEE Transactions,volume CE-31, issue 3, Page 516 – 527,2007.

[3] Van Der Werff, M.; Gui, X.; Xu, W.L., “A mobile-based home automation system”, Mobile Technology, Applications and Systems, 2005 2nd International Conference, Page: 5 pp. – 5, 2005.

[4] Gill, K.; Shuang-Hua Yang ; Fang Yao ; Xin Lu “A zigbee-based home automation system”, volume 55, Issue 2, Page 422 – 430, 2009.

[5] Felix, C.; Raglend, I.J., "Home automation using GSM”, Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), Page: 15 – 19, 2011.