IOT BASED HOME AUTOMATION

Submitted in partial fulfillment of the requirement for the requirement for the award of

DIPLOMA

In

ELECTRICAL AND ELECTRONICS ENGINEERING

By

NIMMALA JEEVAN SAI - PIN NO:16161-EE-014

Under the guidance of

K.SRAVANTHI MAM ,

Lecturer of DEEE

Department of electrical and electronics engineering

2016-18

Department of Electrical and Electronics Engineering

SINGARENI COLLIERIES POLYTECHNIC COLLEGE

(Promoted by singareni collieries educational society)
Approved by AICTE New Delhi and affiliated by SBTET, T.S
C.C.C Naspur ,Mancherial ,PIN:504302 ,Mancherial ,T.S

CERTIFICATE

This is to certify that the project report entitled “IOT BASED HOME AUTOMATION” has been carried by NIMMALA JEEVAN SAI(PIN:16161-EE-014) ,Submitted in partial fulfillment for the award of “DIPLOMA IN ELECTRICAL and ELECTRONICS ENGINEERING” SINGARENI COLLIERIES POLYTECHNIC affiliated to STATE BOARD OF TECHNICAL EDUCATION &TRAINING (SBTET), T.S during the year 2016-19.

INTERNAL GUIDE HEAD OF THE DEPT

EXTERNAL EXAMINER

IOT BASED HOME AUTOMATION

1. INTRODUCTION TO EMBEDDED SYSTEMS

1.1 Embedded System

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, sometimes with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems have become very important today as they control many of the common devices we use.

Since the embedded system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product, or increasing the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically, embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not an exactly defined term, as many systems have some element of programmability. For example, Handheld computers share some elements with embedded systems — such as the operating systems and microprocessors which power them — but are not truly embedded systems, because they allow different applications to be loaded and peripherals to be connected.

An embedded system is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular kind of application device. Industrial machines, automobiles, medical equipment, cameras, household appliances, airplanes, vending machines, and toys (as well as the more obvious cellular phone and PDA) are among the myriad possible hosts of an embedded system. Embedded systems that are programmable are provided with a programming interface, and embedded systems programming is a specialized occupation.

Certain operating systems or language platforms are tailored for the embedded market, such as Embedded Java and Windows XP Embedded. However, some low-end consumer products use very inexpensive microprocessors and limited storage, with the application and operating system both part of a single program. The program is written permanently into the system's memory in this case, rather than being loaded into RAM (random access memory), as programs on a personal computer are.

1.2 Applications of Embedded System

We are living in the Embedded World. You are surrounded with many embedded products and your daily life largely depends on the proper functioning of these gadgets. Television, Radio, CD player of your living room, Washing Machine or Microwave Oven in your kitchen, Card readers, Access Controllers, Palm devices of your work space enable you to do many of your tasks very effectively. Apart from all these, many controllers embedded in your car take care of car operations between the bumpers and most of the times you tend to ignore all these controllers.

In recent days, you are showered with variety of information about these embedded controllers in many places. All kinds of magazines and journals regularly dish out details about latest technologies, new devices; fast applications which make you believe that your basic survival is controlled by these embedded products. Now you can agree to the fact that these embedded products have successfully invaded into our world. You must be wondering about these embedded controllers or systems. What is this Embedded System?

The computer you use to compose your mails, or create a document or analyze the database is known as the standard desktop computer. These desktop computers are manufactured to serve many purposes and applications.

You need to install the relevant software to get the required processing facility. So, these desktop computers can do many things. In contrast, embedded controllers carryout a specific work for which they are designed. Most of the time, engineers design these embedded controllers with a specific goal in mind. So these controllers cannot be used in any other place.

Theoretically, an embedded controller is a combination of a piece of microprocessor based hardware and the suitable software to undertake a specific task.

These days designers have many choices in microprocessors/microcontrollers. Especially, in 8 bit and 32 bit, the available variety really may overwhelm even an experienced designer. Selecting a right microprocessor may turn out as a most difficult first step and it is getting complicated as new devices continue to pop-up very often.

In the 8 bit segment, the most popular and used architecture is Intel's 8031. Market acceptance of this particular family has driven many semiconductor manufacturers to develop something new based on this particular architecture. Even after 25 years of existence, semiconductor manufacturers still come out with some kind of device using this 8031 core.

[1] Military and aerospace software applications

From in-orbit embedded systems to jumbo jets to vital battlefield networks, designers of mission-critical aerospace and defense systems requiring real-time performance, scalability, and high-availability facilities consistently turn to the LynxOS® RTOS and the LynxOS-178 RTOS for software certification to DO-178B.

Rich in system resources and networking services, LynxOS provides an off-the-shelf software platform with hard real-time response backed by powerful distributed computing (CORBA), high reliability, software certification, and long-term support options.

The LynxOS-178 RTOS for software certification, based on the RTCA DO-178B standard, assists developers in gaining certification for their mission- and safety-critical systems. Real-time systems programmers get a boost with LynuxWorks' DO-178B RTOS training courses.

LynxOS-178 is the first DO-178B and EUROCAE/ED-12B certifiable, POSIX®-compatible RTOS solution.

[2] Communications applications

"Five-nines" availability, Compact PCI hot swap support, and hard real-time response—LynxOS delivers on these key requirements and more for today's carrier-class systems. Scalable kernel configurations, distributed computing capabilities, integrated communications stacks, and fault-management facilities make LynxOS the ideal choice for companies looking for a single operating system for all embedded telecommunications applications—from complex central controllers to simple line/trunk cards.

LynuxWorks Jumpstart for Communications package enables OEMs to rapidly develop mission-critical communications equipment, with pre-integrated, state-of-the-art, data networking and porting software components—including source code for easy customization.

The Lynx Certifiable Stack (LCS) is a secure TCP/IP protocol stack designed especially for applications where standards certification is required.

[3] Electronics applications and consumer devices

As the number of powerful embedded processors in consumer devices continues to rise, the BlueCat® Linux® operating system provides a highly reliable and royalty-free option for systems designers.

And as the wireless appliance revolution rolls on, web-enabled navigation systems, radios, personal communication devices, phones and PDAs all benefit from the cost-effective dependability, proven stability and full product life-cycle support opportunities associated with BlueCat embedded Linux. BlueCat has teamed up with industry leaders to make it easier to build Linux mobile phones with Java integration.

For makers of low-cost consumer electronic devices who wish to integrate the LynxOS real-time operating system into their products, we offer special MSRP-based pricing to reduce royalty fees to a negligible portion of the device's MSRP.

[4] Industrial automation and process control software

Designers of industrial and process control systems know from experience that LynuxWorks operating systems provide the security and reliability that their industrial applications require.

From ISO 9001 certification to fault-tolerance, POSIX conformance, secure partitioning and high availability, we've got it all. Take advantage of our 20 years of experience.

1.3 Microcontroller Versus Microprocessor

What is the difference between a Microprocessor and Microcontroller? By microprocessor is meant the general purpose Microprocessors such as Intel's X86 family (8086, 80286, 80386, 80486, and the Pentium) or Motorola's 680X0 family (68000, 68010, 68020, 68030, 68040, etc). These microprocessors contain no RAM, no ROM, and no I/O ports on the chip itself. For this reason, they are commonly referred to as general-purpose Microprocessors.

A system designer using a general-purpose microprocessor such as the Pentium or the 68040 must add RAM, ROM, I/O ports, and timers externally to make them functional. Although the addition of external RAM, ROM, and I/O ports makes these systems bulkier and much more expensive, they have the advantage of versatility such that the designer can decide on the amount of RAM, ROM and I/O ports needed to fit the task at hand. This is not the case with Microcontrollers.

A Microcontroller has a CPU (a microprocessor) in addition to a fixed amount of RAM, ROM, I/O ports, and a timer all on a single chip. In other words, the processor, the RAM, ROM, I/O ports and the timer are all embedded together on one chip; therefore, the designer cannot add any external memory, I/O ports, or timer to it. The fixed amount of on-chip ROM, RAM, and number of I/O ports in Microcontrollers makes them ideal for many applications in which cost and space are critical.

In many applications, for example a TV remote control, there is no need for the computing power of a 486 or even an 8086 microprocessor. These applications most often require some I/O operations to read signals and turn on and off certain bits.

1.4 Microcontrollers For Embedded Systems

In the Literature discussing microprocessors, we often see the term Embedded System. Microprocessors and Microcontrollers are widely used in embedded system products. An embedded system product uses a microprocessor (or Microcontroller) to do one task only. A printer is an example of embedded system since the processor inside it performs one task only; namely getting the data and printing it. Contrast this with a Pentium based PC. A PC can be used for any number of applications such as word processor, print-server, bank teller terminal, Video game, network server, or Internet terminal. Software for a variety of applications can be loaded and run. Of course the reason a pc can perform myriad tasks is that it has RAM memory and an operating system that loads the application software into RAM memory and lets the CPU run it.

In an Embedded system, there is only one application software that is typically burned into ROM. An x86 PC contains or is connected to various embedded products such as keyboard, printer, modem, disk controller, sound card, CD-ROM drives, mouse, and so on. Each one of these peripherals has a Microcontroller inside it that performs only one task. For example, inside every mouse there is a Microcontroller to perform the task of finding the mouse position and sending it to the PC. Table 1-1 lists some embedded products.

2. AVR MICROCONTROLLERS

2.1 Atmel mega AVR microcontrollers:

Atmel® megaAVR® microcontrollers (MCUs) are the ideal choice for designs that need some extra muscle. For applications requiring large amounts of code, megaAVR devices offer substantial program and data memories with performance up to 20 MIPS. Meanwhile, innovative Atmel picoPower® technology minimizes power consumption. All megaAVR devices offer self-programmability for fast, secure, cost-effective in-circuit upgrades. You can even upgrade the Flash memory while running your application.

Based on proven, industry-leading technology, the megaAVR family offers our widest selection of devices in terms of memories, pin-counts and peripherals. These include everything from general-purpose devices to models with specialized peripherals like Peripheral Touch Controller (PTC), USB, LCD controllers, as well as CAN, LIN and Power Stage Controllers (PSC). You will easily find the perfect fit for your project in the megaAVR product family. All these devices are supported by the Atmel Studio development platform, which further reduces your time-to-market.

2.2 Key Features:

- **Broad family** — The megaAVR family offers our widest selection of devices in terms of memories, pin counts and peripherals, enabling reuse of code and knowledge across projects.

- **picoPower technology** — Selected megaAVR devices feature ultra-low power consumption and individually selectable low-power sleep modes that make them ideal for battery-powered applications.

- **High integration** — Devices feature on-chip Flash, SRAM, internal EEPROM, SPI, TWI (I^2C), and USART, USB, CAN, and LIN, watchdog timer, a choice of internal or external precision oscillator, and general-purpose I/O pins, simplifying your design and reducing the bill-of-materials.

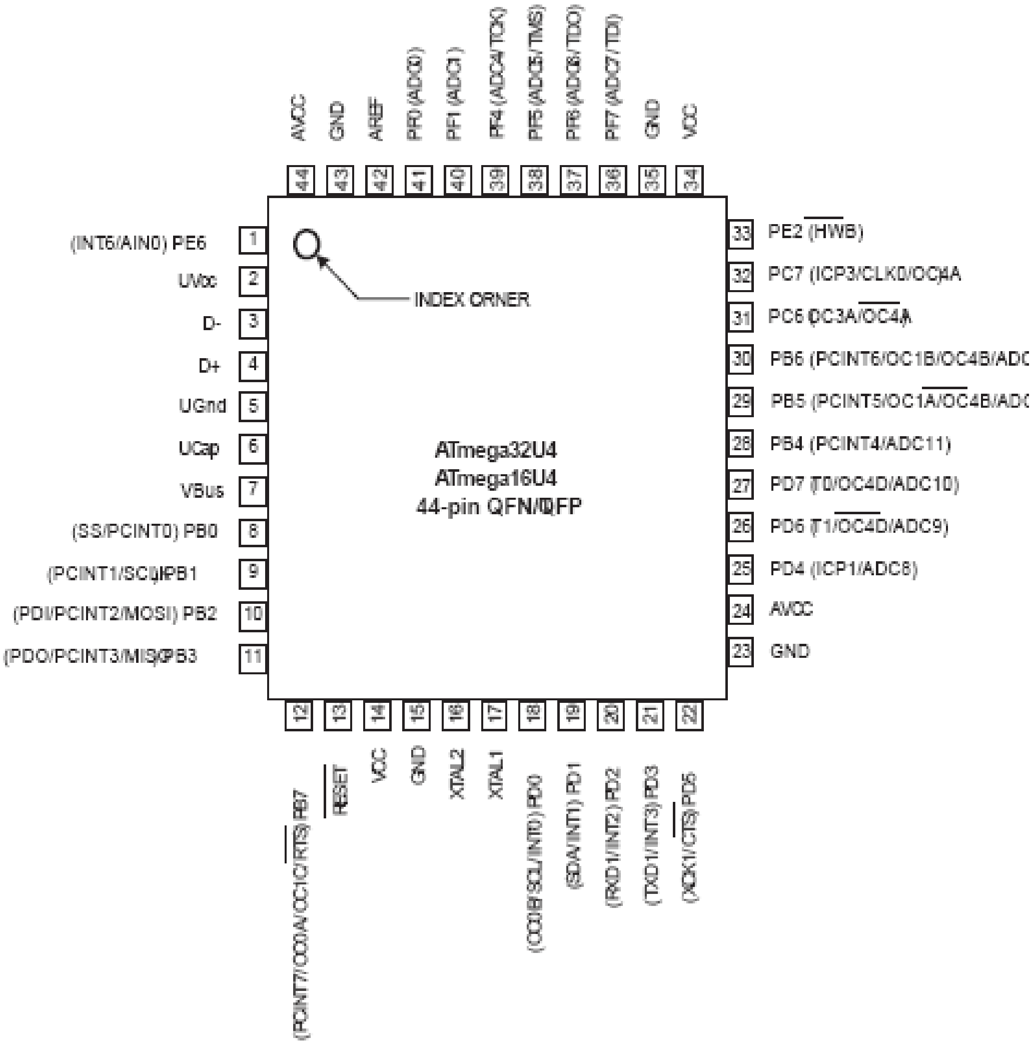
- **Analog functions** — Advanced analog capabilities include ADC, DAC, a built-in temperature sensor and internal voltage reference, brown out detector, a fast analog comparator and a programmable analog gain amplifier. This high level of integration allows designs with fewer external analog components.
- **Rapid development** — megaAVR MCUs speed development with powerful in-system programming and on-chip debug. In addition, in-system programming simplifies production line programming and field upgrades.
- **IoT ready** — The IoT (Internet of Things) can extend to almost any application—from typical building and home automation to medical and healthcare systems. IoT designs typically require some form of processing power to perform embedded computing tasks and transmit data to the Internet. Increasingly, these devices are battery driven, thus power consumption often becomes the key success factor for a user-friendly IoT-enabled product. megaAVR devices are among the best MCUs in the world when it comes to power consumption, making them a natural choice for IoT applications.
- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture – 135 Powerful Instructions – Most Single Clock Cycle Execution – 32 x 8 General Purpose Working Registers – Fully Static Operation – Up to 16 MIPS Throughput at 16MHz – On-Chip 2-cycle Multiplier
- Non-volatile Program and Data Memories – 16/32KB of In-System Self-Programmable Flash – 1.25/2.5KB Internal SRAM – 512Bytes/1KB Internal EEPROM – Write/Erase Cycles: 10,000 Flash/100,000 EEPROM C(1)°C/ 100 years at 25°– Data retention: 20 years at 85 – Optional Boot Code Section with Independent Lock Bits In-System Programming by On-chip Boot Program True Read-While-Write Operation Parts using external XTAL clock are pre-programed with a default USB bootloader – Programming Lock for Software Security

- JTAG (IEEE® std. 1149.1 compliant) Interface – Boundary-scan Capabilities According to the JTAG Standard – Extensive On-chip Debug Support – Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface

- USB 2.0 Full-speed/Low Speed Device Module with Interrupt on Transfer Completion – Complies fully with Universal Serial Bus Specification Rev 2.0 – Supports data transfer rates up to 12Mbit/s and 1.5Mbit/s – Endpoint 0 for Control Transfers: up to 64-bytes – Six Programmable Endpoints with IN or Out Directions and with Bulk, Interrupt or Isochronous Transfers – Configurable Endpoints size up to 256 bytes in double bank mode – Fully independent 832 bytes USB DPRAM for endpoint memory allocation – Suspend/Resume Interrupts – CPU Reset possible on USB Bus Reset detection – 48MHz from PLL for Full-speed Bus Operation – USB Bus Connection/Disconnection on Microcontroller Request – Crystal-less operation for Low Speed mode

- Peripheral Features – On-chip PLL for USB and High Speed Timer: 32 up to 96MHz operation – One 8-bit Timer/Counter with Separate Prescaler and Compare Mode

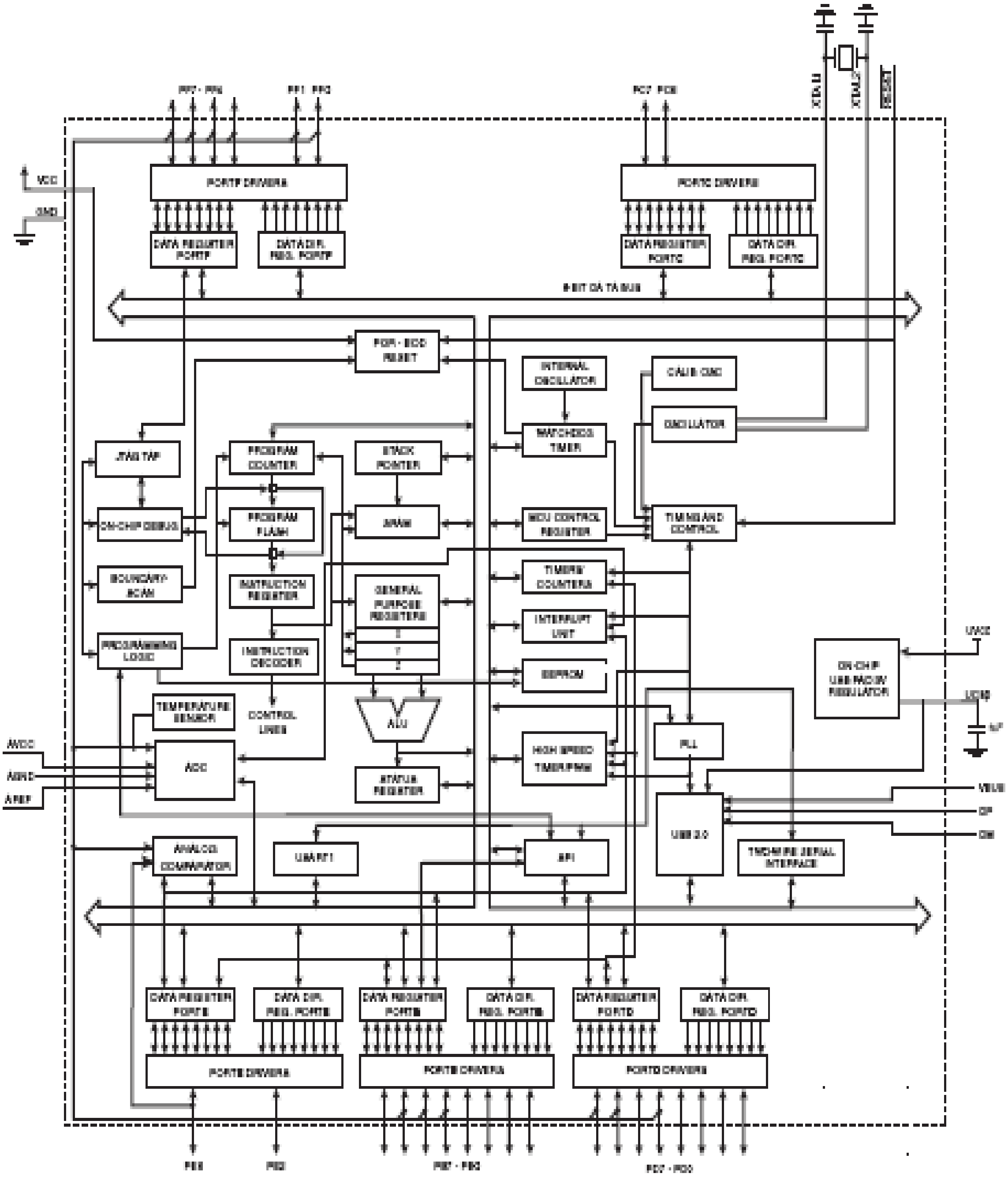
2.3 Pin Configuration:



2.4 Overview:

The ATmega16U4/ATmega32U4 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 device achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

Block diagram:



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. The device provides the following features: 16/32K bytes of In-System Programmable Flash with Read-While Write capabilities, 512Bytes/1K bytes EEPROM, 1.25/2.5K bytes SRAM, 26 general purpose I/O lines (CMOS outputs and LVTTL inputs), 32 general purpose working registers, four flexible Timer/Counters with compare modes and PWM, one more high-speed Timer/Counter with compare modes and PLL adjustable source, one USART (including CTS/RTS flow control signals), a byte oriented 2-wire Serial Interface, a 12-channels 10-bit ADC with optional differential input stage with programmable gain, an on-chip calibrated temperature sensor, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, IEEE std. 1149.1 compliant JTAG test interface, also used for accessing the On-chip Debug system and programming and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, 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. The ADC Noise Reduction mode stops the CPU and all I/O modules except 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 the Atmel® high-density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile 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 device is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications. The ATmega16U4/ATmega32U4 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.

ATMEGA328:

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 program 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 I2C 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

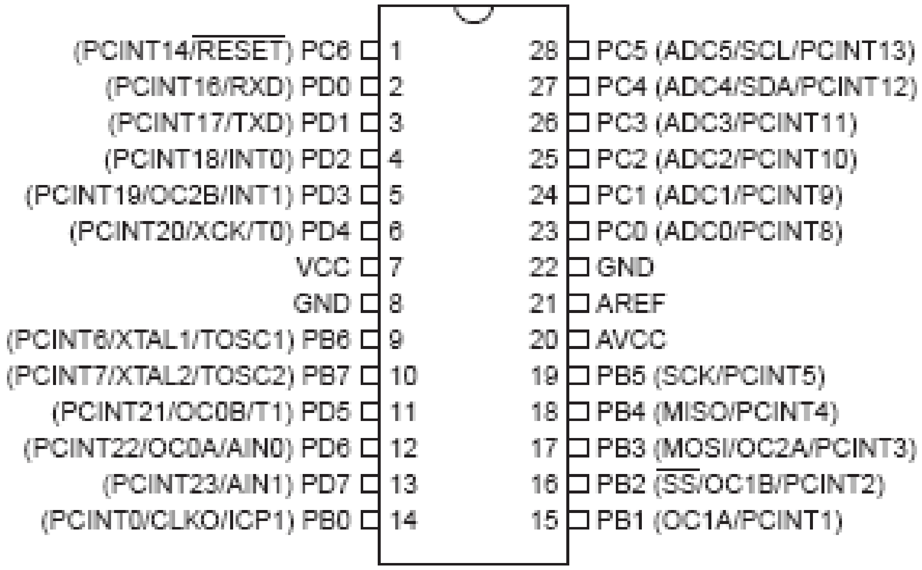
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:

– Active Mode: 0.2 mA

– Power-down Mode: 0.1 µA

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

PDIP



1.1 Pin Descriptions

1.1.1 VCC Digital supply voltage.

1.1.2 GND Ground.

1.1.3 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 it). 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 82 and ”System Clock and Clock Options” on page 26.

1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each it). 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.

1.1.5 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 un programmed, 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 318. 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 85.

1.1.6 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

88.

1.1.7 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.

1.1.8 AREF

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

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

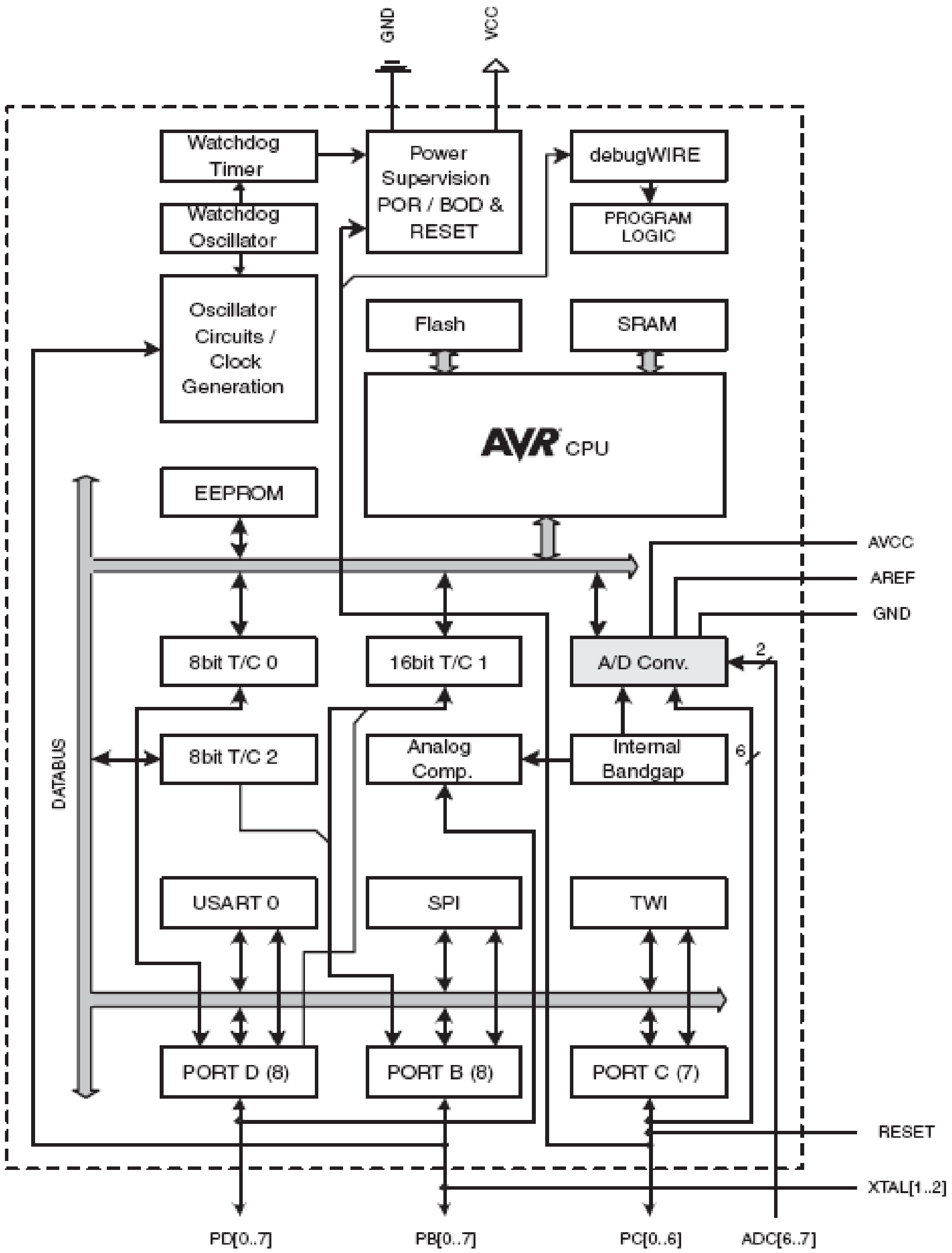
In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

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.

2.1 Block Diagram

Figure 2-1. Block Diagram



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.

The ATmega48PA/88PA/168PA/328P provides the following features: 4K/8K bytes of In-System 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.

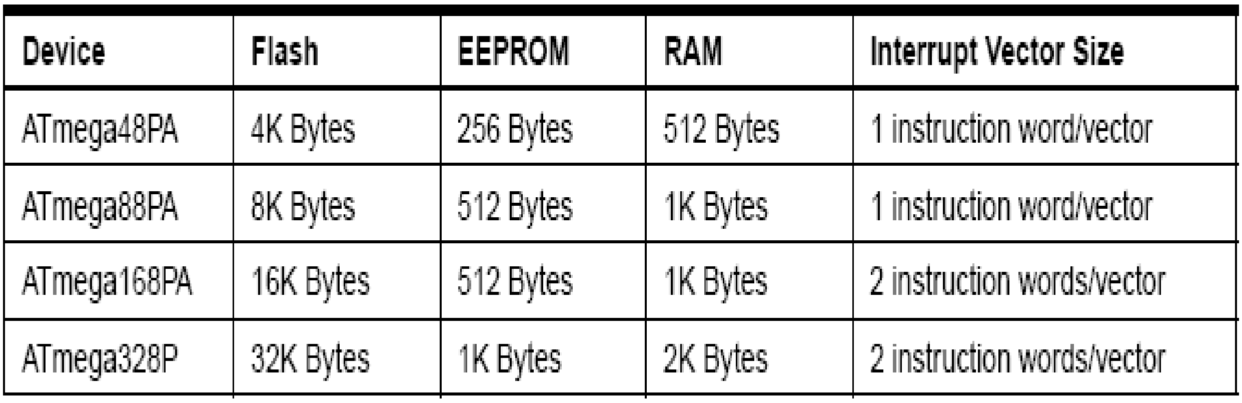
In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby 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.

2.2 Comparison Between ATmega48PA, ATmega88PA, ATmega168PA and ATmega328P

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

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.

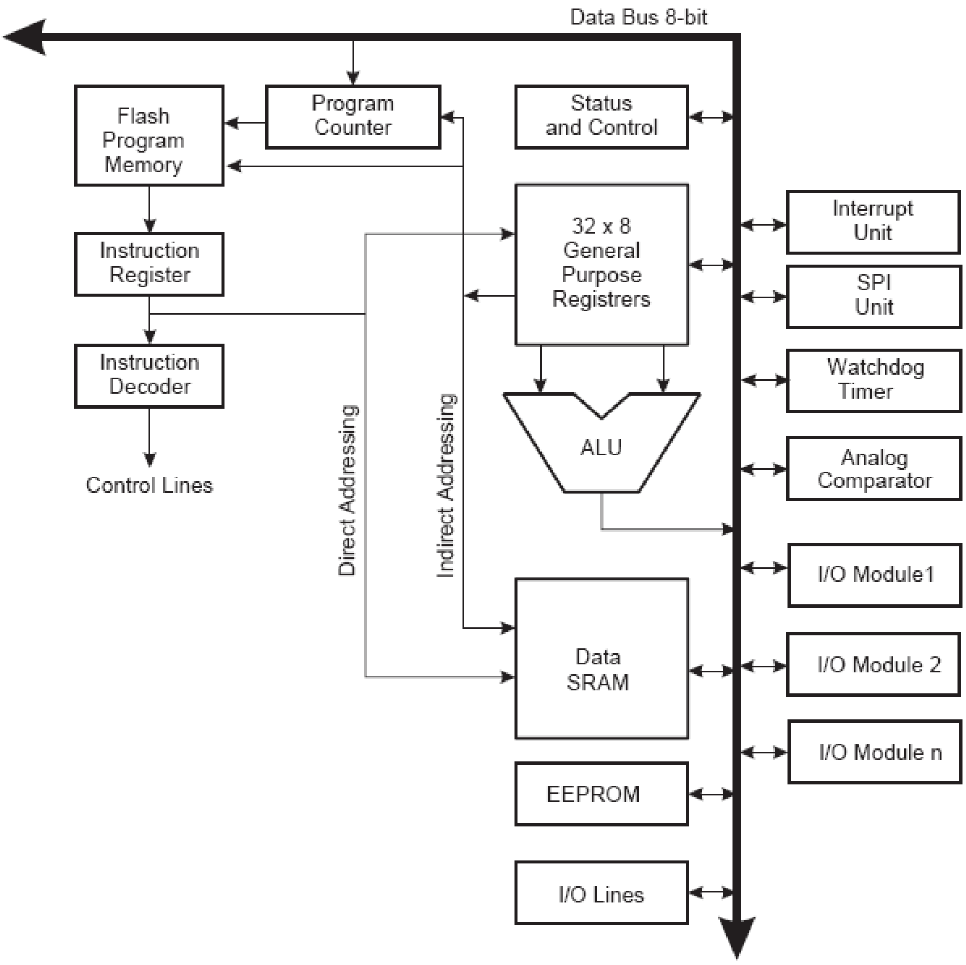
ATmega88PA, ATmega168PA and ATmega328P support a real Read-While-Write Self-Programming mechanism. There is a separate Boot Loader Section, and the SPM instruction can only execute from there. In ATmega48PA, there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction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

AVR CPU Core

6.1 Overview

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

Figure 6-1. Block Diagram of the AVR Architecture



In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.

The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle. Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing

– enabling efficient address calculations. One of the these address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the Reset routine (before subroutines or interrupts are executed). The Stack Pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position.

The lower the Interrupt Vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers,

SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F. In addition, the

ATmega48PA/88PA/168PA/328P has Extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

6.2 ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See the "Instruction Set" section for a detailed description.

6.3 Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code. The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

AVR Memories

7.1 Overview

This section describes the different memories in the ATmega48PA/88PA/168PA/328P. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the ATmega48PA/88PA/168PA/328P features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

7.2 In-System Reprogrammable Flash Program Memory

The ATmega48PA/88PA/168PA/328P contains 4/8/16/32K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 2/4/8/16K x 16. For software security, the Flash Program memory space is divided into two sections, Boot Loader Section and Application Program

Section in ATmega88PA and ATmega168PA. See SELFPRGEN description in section ”SPMCSR – Store Program Memory Control and Status Register” on page 292 for more details.

The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega48PA/88PA/168PA/328P Program Counter (PC) is 11/12/13/14 bits wide, thus addressing the 2/4/8/16K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in ”Self-Programming the Flash, ATmega48PA” on page 269 and ”Boot Loader Support – Read-While-Write Self-Programming, ATmega88PA, ATmega168PA and ATmega328P” on page 277. ”Memory Programming” on page 294 contains a detailed description on Flash Programming in SPI- or Parallel Programming mode.

Constant tables can be allocated within the entire program memory address space (see the LPM – Load Program Memory instruction description).

SRAM Data Memory

The ATmega48PA/88PA/168PA/328P is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in the Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used. The lower 768/1280/1280/2303 data memory locations address both the Register File, the I/O memory, Extended I/O memory, and the internal data SRAM. The first 32 locations address the Register File, the next 64 location the standard I/O memory, then 160 locations of Extended I/O memory, and the next 512/1024/1024/2048 locations address the internal data SRAM.

The five different addressing modes for the data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.

The direct addressing reaches the entire data space. The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.

When using register indirect addressing modes with automatic pre-decrement and post-increment, the address registers X, Y, and Z are decremented or incremented. The 32 general purpose working registers, 64 I/O Registers, 160 Extended I/O Registers, and the 512/1024/1024/2048 bytes of internal data SRAM in the ATmega48PA/88PA/168PA/328P are all accessible through all these addressing modes.

EEPROM Data Memory

The ATmega48PA/88PA/168PA/328P contains 256/512/512/1K bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

7.4.1 EEPROM Read/Write Access

The EEPROM Access Registers are accessible in the I/O space.

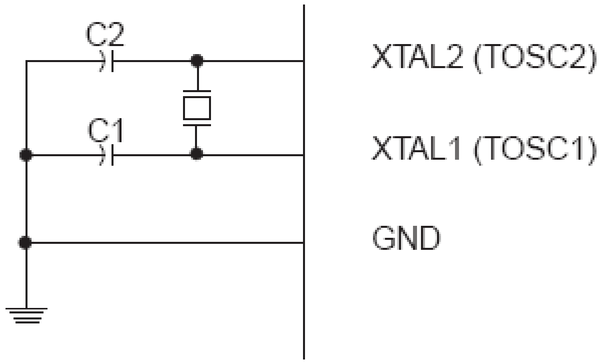
lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies, VCC is likely to rise or fall slowly on power-up/down. This causes the device for some period of time to run at a voltage lower than specified as minimum for the clock frequency used. In order to prevent unintentional EEPROM writes, a specific write procedure must be followed. Refer to the description of the EEPROM Control Register for details on this. When the EEPROM is read, the CPU is halted for four clock cycles before the next instruction is executed. When the EEPROM is written, the CPU is halted for two clock cycles before the next instruction is executed.

Low Power Crystal Oscillator

Pins XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator, Either a quartz crystal or a ceramic resonator may be used. This Crystal Oscillator is a low power oscillator, with reduced voltage swing on the XTAL2 output.

It gives the lowest power consumption, but is not capable of driving other clock inputs, and may be more susceptible to noise in noisy environments. C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. For ceramic resonators, the capacitor values given by the manufacturer should be used.

Figure 8-2. Crystal Oscillator Connections



Watchdog Timer

Features

- Clocked from separate On-chip Oscillator
- 3 Operating modes

– Interrupt

– System Reset

– Interrupt and System Reset

- Selectable Time-out period from 16ms to 8s
- Possible Hardware fuse Watchdog always on (WDTON) for fail-safe mode

Overview

ATmega48PA/88PA/168PA/328P has an Enhanced Watchdog Timer (WDT). The WDT is a timer counting cycles of a separate on-chip 128 kHz oscillator. The WDT gives an interrupt or a system reset when the counter reaches a given time-out value. In normal operation mode, it is required that the system uses the WDR - Watchdog Timer Reset - instruction to restart the counter before the time-out value is reached. If the system doesn't restart the counter, an interrupt or system reset will be issued.

In Interrupt mode, the WDT gives an interrupt when the timer expires. This interrupt can be used to wake the device from sleep-modes, and also as a general system timer. One example is to limit the maximum time allowed for certain operations, giving an interrupt when the operation has run longer than expected. In System Reset mode, the WDT gives a reset when the timer expires. This is typically used to prevent system hang-up in case of runaway code. The third mode, Interrupt and System Reset mode, combines the other two modes by first giving an interrupt and then switch to System Reset mode. This mode will for instance allow a safe shutdown by saving critical parameters before a system reset.

The Watchdog always on (WDTON) fuse, if programmed, will force the Watchdog Timer to System Reset mode. With the fuse programmed the System Reset mode bit (WDE) and Interrupt mode bit (WDIE) are locked to 1 and 0 respectively. To further ensure program security, alterations to the Watchdog set-up must follow timed sequences. The sequence for clearing WDE and changing time-out configuration is as follows:

1. In the same operation, write a logic one to the Watchdog change enable bit (WDCE) and WDE. A logic one must be written to WDE regardless of the previous value of the WDE bit.

2. Within the next four clock cycles, write the WDE and Watchdog prescaler bits (WDP) as desired, but with the WDCE bit cleared. This must be done in one operation.

The following code example shows one assembly and one C function for turning off the Watchdog Timer. The example assumes that interrupts are controlled (e.g. by disabling interrupts globally) so that no interrupts will occur during the execution of these functions.

8-bit Timer/Counter0 with PWM

Features

- Two Independent Output Compare Units

- Double Buffered Output Compare Registers

- Clear Timer on Compare Match (Auto Reload)

- Glitch Free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- Three Independent Interrupt Sources (TOV0, OCF0A, and OCF0B)

Overview

Timer/Counter0 is a general purpose 8-bit Timer/Counter module, with two independent Output Compare Units, and with PWM support. It allows accurate program execution timing (event management) and wave generation.

CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold.

2.5 Pin Descriptions

2.5.1 VCC Digital supply voltage**.**

2.5.2 GND Ground.

2.5.3 Port B (PB7..PB0)

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tristated when a reset condition becomes active, even if the clock is not running. Port B has better driving capabilities than the other ports. Port B also serves the functions of various special features of the device as listed on page 74.

2.5.4 Port C (PC7,PC6)

Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tristated when a reset condition becomes active, even if the clock is not running. Only bits 6 and 7 are present on the product pinout. Port C also serves the functions of special features of the device as listed on page 77.

2.5.5 Port D (PD7..PD0)

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

2.5.6 Port E (PE6,PE2)

Port E is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port E output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port E pins that are externally pulled low will source current if the pull-up resistors are activated. The Port E pins are tristated when a reset condition becomes active, even if the clock is not running. Only bits 2 and 6 are present on the product pinout.

2.5.7 Port F (PF7..PF4, PF1,PF0)

Port F serves as analog inputs to the A/D Converter. Port F also serves as an 8-bit bi-directional I/O port, if the A/D Converter channels are not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port F output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port F pins that are externally pulled low will source current if the pull-up resistors are activated. The Port F pins are tri-stated when a reset condition becomes active, even if the clock is not running. Bits 2 and 3 are not present on the product pinout. Port F also serves the functions of the JTAG interface. If the JTAG interface is enabled, the pull-up resistors on pins PF7(TDI), PF5(TMS), and PF4(TCK) will be activated even if a reset occurs.

2.5.8 DUSB

Full speed / Low Speed Negative Data Upstream Port. Should be connected to the USB D- connector pin with a serial 22□resistor.

2.5.9 D+

USB Full speed / Low Speed Positive Data Upstream Port. Should be connected to the USB D+ connector pin with a serial 22□resistor.

2.5.10 UGND

USB Pads Ground.

2.5.11 UVCC

USB Pads Internal Regulator Input supply voltage.

2.5.12 UCAP

USB Pads Internal Regulator Output supply voltage. Should be connected to an external capacitor (1µF).

2.5.13 VBUS

USB VBUS monitor input.

2.5.14 RESET

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.

2.5.15 XTAL1

Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

2.5.16 XTAL2

Output from the inverting Oscillator amplifier.

2.5.17 AVCC

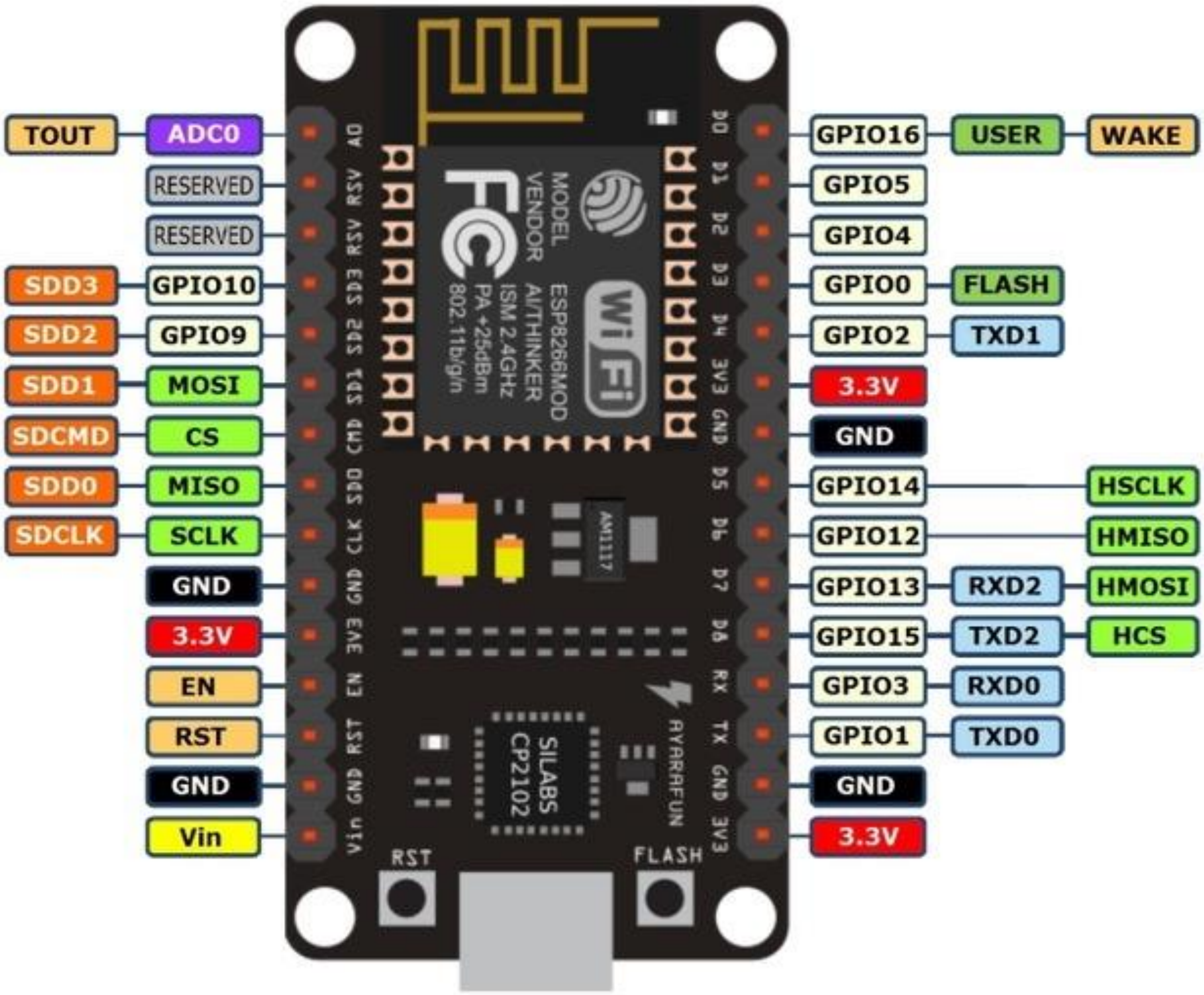
AVCC is the supply voltage pin (input) for all the A/D Converter channels. If the ADC is not used, it should be externally connected to VCC. If the ADC is used, it should be connected to VCC through a low-pass filter.

2.5.18 AREF

This is the analog reference pin (input) for the A/D Converter.

2.6 Node MCU Arduino core:

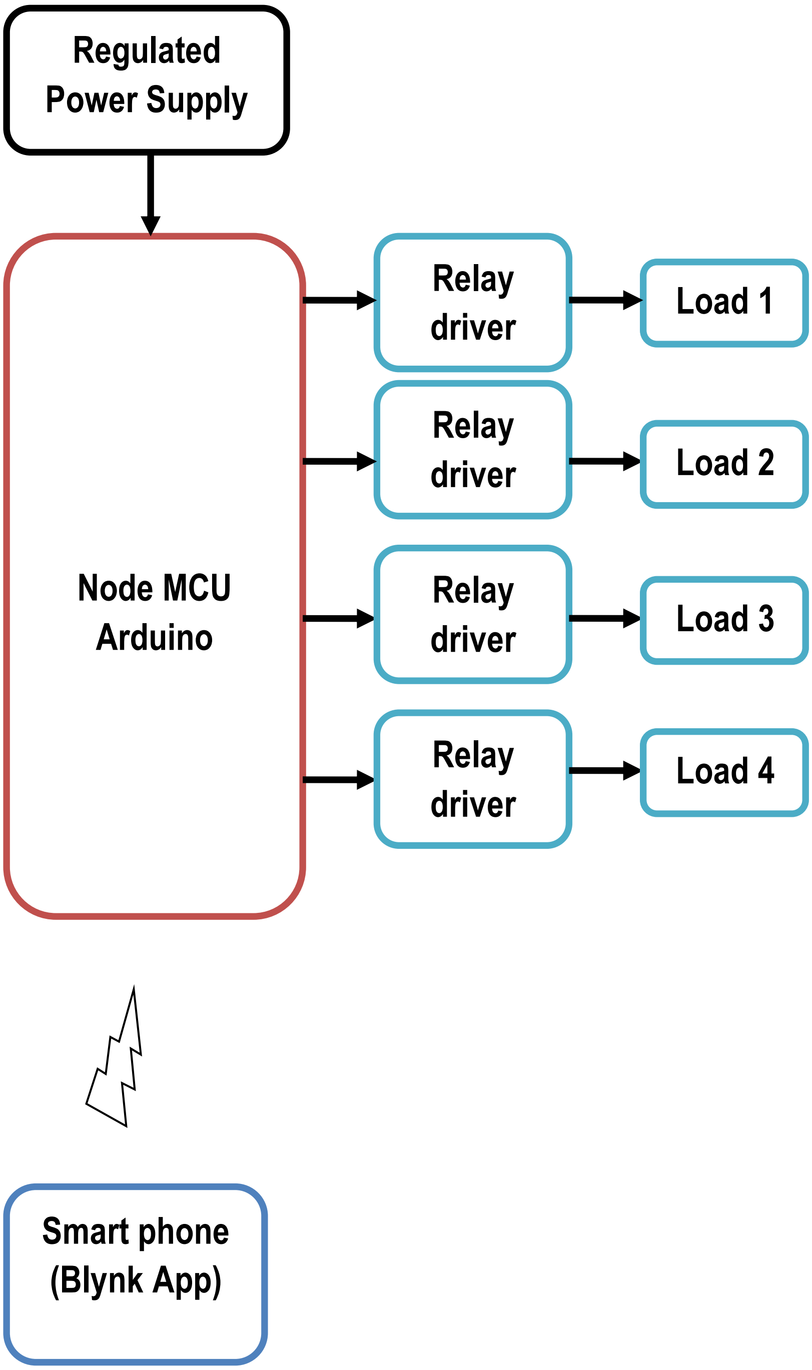
NodeMCU is an open source IoT platform. It includes firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module.[6][7] The term "NodeMCU" by default refers to the firmware rather than the development kits. The firmware uses the Lua scripting language. It is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open source projects, such as lua-cjson, and spiffs.



As Arduino.cc began developing new MCU boards based on non-AVR processors like the ARM/SAM MCU and used in the Arduino Due, they needed to modify the Arduino IDE so that it would be relatively easy to change the IDE to support alternate tool chains to allow Arduino C/C++ to be compiled down to these new processors. They did this with the introduction of the Board Manager and the SAM Core. A "core" is the collection of software components required by the Board Manager and the Arduino IDE to compile an Arduino C/C++ source file down to the target MCU's machine language. Some creative ESP8266 enthusiasts have developed an Arduino core for the ESP8266 WiFi SoC that is available at the GitHub ESP8266 Core webpage. This is what is popularly called the "ESP8266 Core for the Arduino IDE" and it has become one of the leading software development platforms for the various ESP8266 based modules and development boards, including NodeMCUs.

3. HARDWARE IMPLEMENTATION

3.1 Block diagram:



3.2 Regulated Power Supply:

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

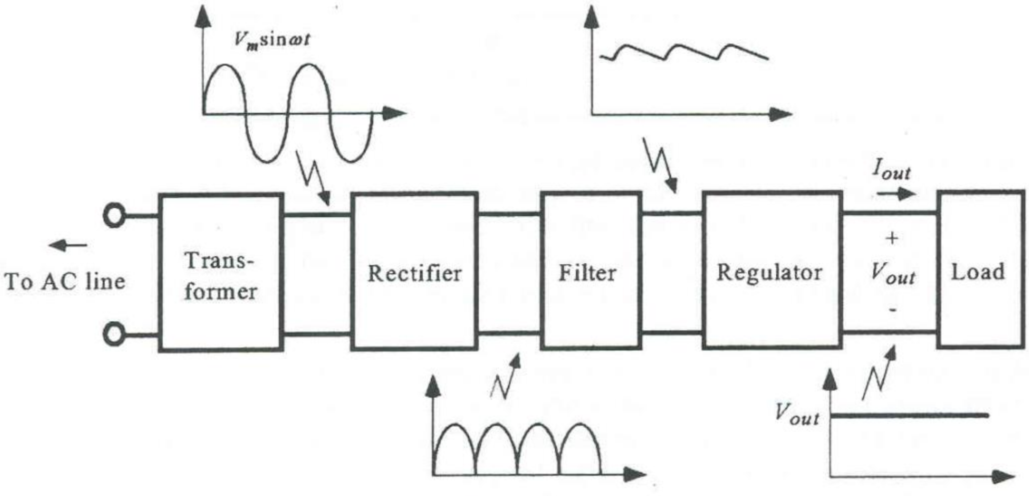
This type of regulation is ideal for having a simple variable bench power supply. Actually this is quite important because one of the first projects a hobbyist should undertake is the construction of a variable regulated power supply. While a dedicated supply is quite handy ,it's much handier to have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

Mainly the microcontroller needs 5 volt power supply. To use these parts we need to build a regulated 5 volt source. Usually you start with an unregulated power To make a 5 volt power supply, we use a 7805 voltage regulator IC (Integrated Circuit).

Circuit Features:-

V_{out} range	1.25V - 37V
V_{in} - V_{out} difference	3V - 40V
Operation ambient temperature	0 - 125°C
Output I_{max}	<1.5A
Minimum Load Current$_{max}$	10Ma

Block Diagram



Components of a typical linear power supply

Power supply:

Every electrical and electronic device that we use in our day-to-day life will require a power supply. In general, we use an AC supply of 230V 50Hz, but this power has to be changed into the required form with required values or voltage range for providing power supply to different types of devices. There are various types of power electronic converters such as step-down converter, step-up converter, voltage stabilizer, AC to DC converter, DC to DC converter, DC to AC converter, and so on. For example, consider the microcontrollers that are used frequently for developing many embedded systems' based projects and kits used in real-time applications. These microcontrollers require a 5V DC supply, so the AC 230V needs to be converted into 5V DC using the step-down converter in their power supply circuit.

Power supply circuit, the name itself indicates that this circuit is used to supply the power to other electrical and electronic circuits or devices. There are different types of power supply circuits based on the power they are used to provide for devices. For example, the micro-controller based circuits, usually the 5V DC regulated power supply circuits, are used, which can be designed using different techniques for converting the available 230V AC power to 5V DC power. Generally the converters with output voltage less than the input voltage are called as step-down converters.

Step Down the Voltage Level

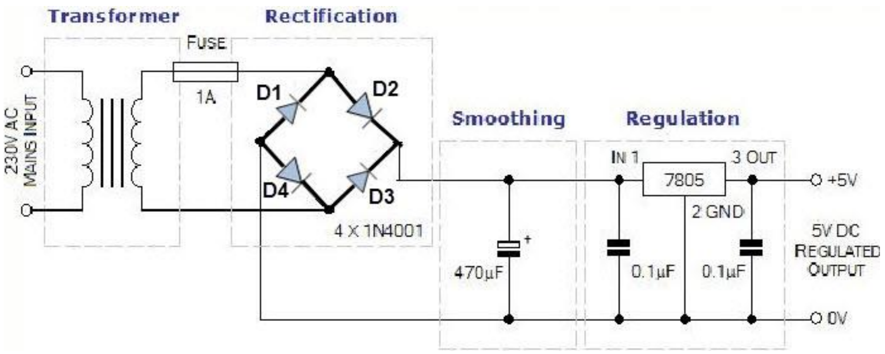
The step-down converters are used for converting the high voltage into low voltage. The converter with output voltage less than the input voltage is called as a step-down converter, and the converter with output voltage greater than the input voltage is called as step-up converter. There are step-up and step-down transformers which are used to step up or step down the voltage levels. 230V AC is converted into 12V AC using a step-down transformer. 12V output of stepdown transformer is an RMS value and its peak value is given by the product of square root of two with RMS value, which is approximately 17V.

Step-down Transformer

Step-down transformer consists of two windings, namely primary and secondary windings where primary can be designed using a less-gauge wire with more number of turns as it is used for carrying low-current high-voltage power, and the secondary winding using a high-gauge wire with less number of turns as it is used for carrying high-current low-voltage power. Transformers works on the principle of Faraday’s laws of electromagnetic induction.

Convert AC to DC

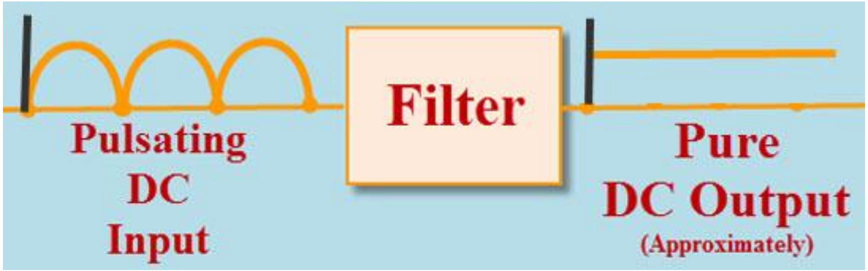
230V AC power is converted into 12V AC (12V RMS value wherein the peak value is around 17V), but the required power is 5V DC; for this purpose, 17V AC power must be primarily converted into DC power then it can be stepped down to the 5V DC. But first and foremost, we must know how to convert AC to DC? AC power can be converted into DC using one of the power electronic converters called as Rectifier. There are different types of rectifiers, such as half-wave rectifier, full-wave rectifier and bridge rectifier. Due to the advantages of the bridge rectifier over the half and full wave rectifier, the bridge rectifier is frequently used for converting AC to DC.



Bridge rectifier consists of four diodes which are connected in the form a bridge. We know that the diode is an uncontrolled rectifier which will conduct only forward bias and will not conduct during the reverse bias. If the diode anode voltage is greater than the cathode voltage then the diode is said to be in forward bias. During positive half cycle, diodes D2 and D4 will conduct and during negative half cycle diodes D1 and D3 will conduct. Thus, AC is converted into DC; here the obtained is not a pure DC as it consists of pulses. Hence, it is called as pulsating DC power. But voltage drop across the diodes is (2*0.7V) 1.4V; therefore, the peak voltage at the output of this rectifier circuit is 15V (17-1.4) approx.

Smoothing the Ripples using Filter

15V DC can be regulated into 5V DC using a step-down converter, but before this, it is required to obtain pure DC power. The output of the diode bridge is a DC consisting of ripples also called as pulsating DC. This pulsating DC can be filtered using an inductor filter or a capacitor filter or a resistor-capacitor-coupled filter for removing the ripples. Consider a capacitor filter which is frequently used in most cases for smoothing.

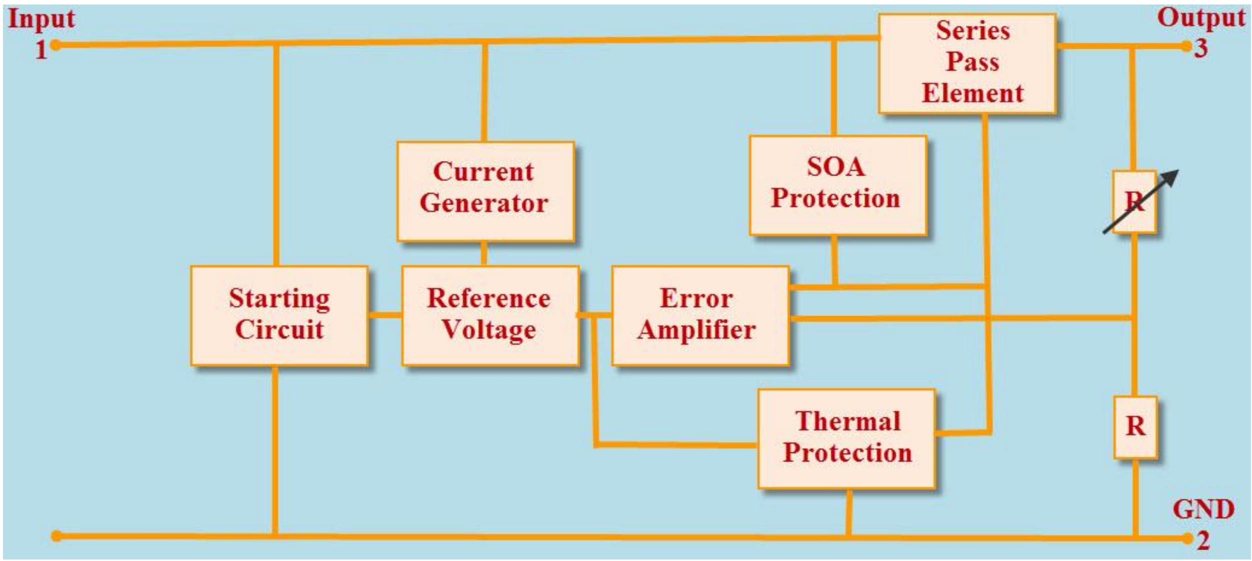


Filter

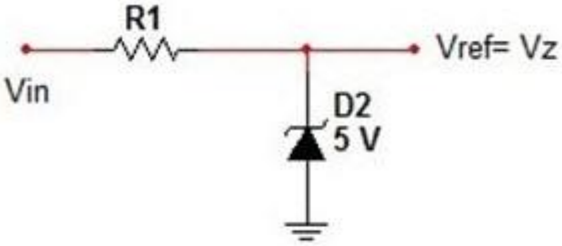
We know that a capacitor is an energy storing element. In the circuit, capacitor stores energy while the input increases from zero to a peak value and, while the supply voltage decreases from peak value to zero, capacitor starts discharging. This charging and discharging of the capacitor will make the pulsating DC into pure DC, as shown in figure.

Regulating 12V DC into 5V DC using Voltage Regulator

15V DC voltage can be stepped down to 5V DC voltage using a DC step-down converter called as voltage regulator IC7805. The first two digits '78' of IC7805 voltage regulator represent positive series voltage regulators and the last two digits '05' represents the output voltage of the voltage regulator.



The block diagram of IC7805 voltage regulator is shown in the figure consists of an operating amplifier acting as error amplifier, zener diode used for providing voltage reference, as shown in the figure.



Zener Diode as Voltage Reference

Transistor as a series pass element used for dissipating extra energy as heat; SOA protection (Safe Operating Area) and heat sink are used for thermal protection in case of excessive supply voltages. In general, an IC7805 regulator can withstand voltage ranging from 7.2V to 35V and gives maximum efficiency of 7.2V voltage and if the voltage exceeds 7.2V, then there is loss of energy in the form of heat. To protect the regulator from over heat, thermal protection is provided using a heat sink. Thus, a 5V DC is obtained from 230V AC power.

Internet of Things:

The Internet of Things (IoT) is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

A thing, in the Internet of Things, can be a person with a heart monitor implant, a farm animal with a biochip transponder, an automobile that has built-in sensors to alert the driver when tire pressure is low -- or any other natural or man-made object that can be assigned an IP address and provided with the ability to transfer data over a network.

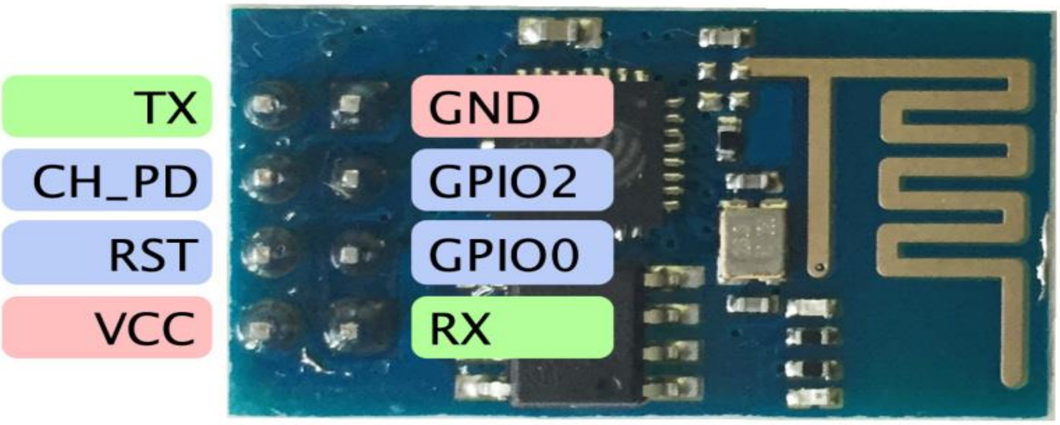
IoT has evolved from the convergence of wireless technologies, micro-electromechanical systems (MEMS), micro services and the internet. The convergence has helped tear down the silo walls between operational technology (OT) and information technology (IT), allowing unstructured machine-generated data to be analyzed for insights that will drive improvements.

The Internet of things (stylized Internet of Things or IoT) is the internetworking of physical devices, vehicles (also referred to as "connected devices" and "smart devices"), buildings, and other items—embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data. In 2013 the Global Standards Initiative on Internet of Things (IoT-GSI) defined the IoT as "the infrastructure of the information society." The IoT allows objects to be sensed and/or controlled remotely across existing network infrastructure, creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit in addition to reduced human intervention. When IoT is augmented with sensors and actuators, the technology becomes an instance of the more general class of cyber-physical systems, which also encompasses technologies such as smart grids, smart homes, intelligent transportation and smart cities. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure. Experts estimate that the IoT will consist of almost 50 billion objects by 2020.

The Internet of Things (IoT) is the network of everyday objects — physical things embedded with electronics, software, sensors, and connectivity enabling data exchange. Basically, a little networked computer is attached to a thing, allowing information exchange to and from that thing. Be it lightbulbs, toasters, refrigerators, flower pots, watches, fans, planes, trains, automobiles, or anything else around you, a little networked computer can be combined with it to accept input (esp. object control) or to gather and generate informational output (typically object status or other sensory data). This means computers will be permeating everything around us — ubiquitous embedded computing devices, uniquely identifiable, interconnected across the Internet. Because of low-cost, networkable micro-controller modules, the Internet of Things is really starting to take off.

ESP 8266:

ESP8266 (presently ESP8266EX) is a chip with which manufacturers are making wirelessly networkable micro-controller modules. More specifically, ESP8266 is a system-on-a-chip (SoC) with capabilities for 2.4 GHz Wi-Fi (802.11 b/g/n, supporting WPA/WPA2), general-purpose input/output (16 GPIO), Inter-Integrated Circuit (I²C), analog-to-digital conversion (10-bit ADC), Serial Peripheral Interface (SPI), I²S interfaces with DMA (sharing pins with GPIO), UART (on dedicated pins, plus a transmit-only UART can be enabled on GPIO2), and pulse-width modulation (PWM). It employs a 32-bit RISC CPU based on the Tensilica Xtensa LX106 running at 80 MHz (or overclocked to 160 MHz). It has a 64 KB boot ROM, 64 KB instruction RAM and 96 KB data RAM. External flash memory can be accessed through SPI.



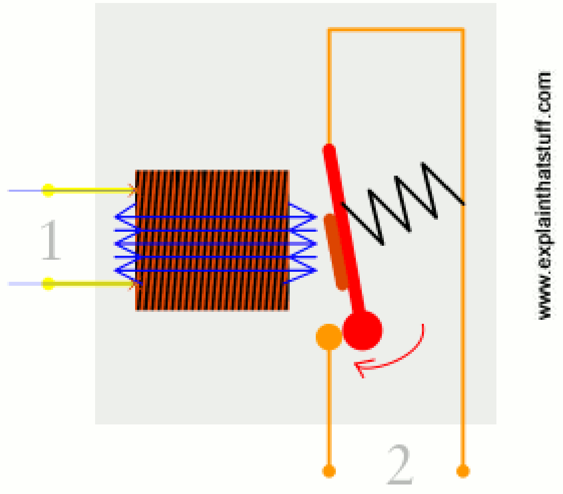
Various vendors have consequently created a multitude of modules containing the ESP8266 chip at their cores. Some of these modules have specific identifiers, including monikers such as "Wi07c" and "ESP-01" through "ESP-13"; while other modules might be ill-labeled and merely referred to by a general description — e.g., "ESP8266 Wireless Transceiver." ESP8266-based modules have demonstrated themselves as a capable, low-cost, networkable foundation for facilitating end-point IoT developments. Espressif's official module is presently the ESP-WROOM-02. The AI-Thinker modules are succinctly labeled ESP-01 through ESP-13. NodeMCU boards extend upon the AI-Thinker modules. Olimex, Adafruit, Spark fun, WeMos, ESPert (ESPresso) all make various modules as well.

Relay:

A relay is an electrically operated switch. Many relays use an electromagnet to mechanically operate a switch, but other operating principles are also used, such as solid-state relays. Relays are used where it is necessary to control a circuit by a separate low-power signal, or where several circuits must be controlled by one signal. The first relays were used in long distance telegraph circuits as amplifiers: they repeated the signal coming in from one circuit and re-transmitted it on another circuit. Relays were used extensively in telephone exchanges and early computers to perform logical operations.

The heart of a relay is an electromagnet (a coil of wire that becomes a temporary magnet when electricity flows through it). You can think of a relay as a kind of electric lever: switch it on with a tiny current and it switches on ("leverages") another appliance using a much bigger current. Why is that useful? As the name suggests, many sensors are incredibly *sensitive* pieces of electronic equipment and produce only small electric currents. But often we need them to drive bigger pieces of apparatus that use bigger currents. Relays bridge the gap, making it possible for small currents to activate larger ones. That means relays can work either as switches (turning things on and off) or as amplifiers (converting small currents into larger ones).

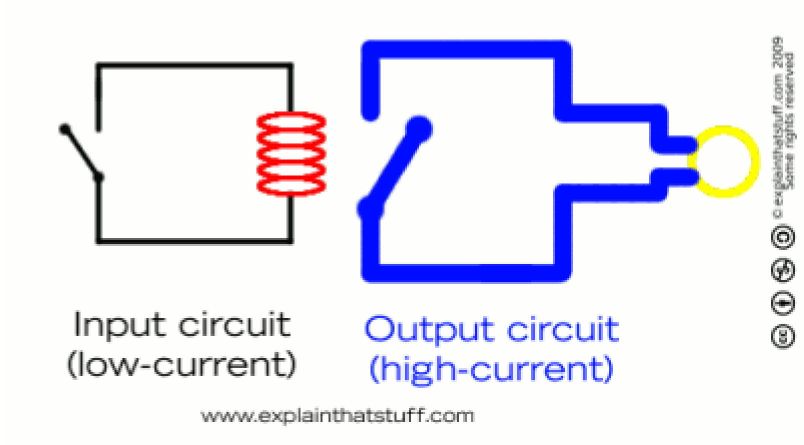
Here are two simple animations illustrating how relays use one circuit to switch on a second circuit.



When power flows through the first circuit (1), it activates the electromagnet (brown), generating a magnetic field (blue) that attracts a contact (red) and activates the second circuit (2). When the power is switched off, a spring pulls the contact back up to its original position, switching the second circuit off again.

This is an example of a "normally open" (NO) relay: the contacts in the second circuit are not connected by default, and switch on only when a current flows through the magnet. Other relays are "normally closed" (NC; the contacts are connected so a current flows through them by default) and switch off only when the magnet is activated, pulling or pushing the contacts apart. Normally open relays are the most common.

Here's another animation showing how a relay links two circuits together. It's essentially the same thing drawn in a slightly different way. On the left side, there's an input circuit powered by a switch or a sensor of some kind. When this circuit is activated, it feeds current to an electromagnet that pulls a metal switch closed and activates the second, output circuit (on the right side). The relatively small current in the input circuit thus activates the larger current in the output circuit:

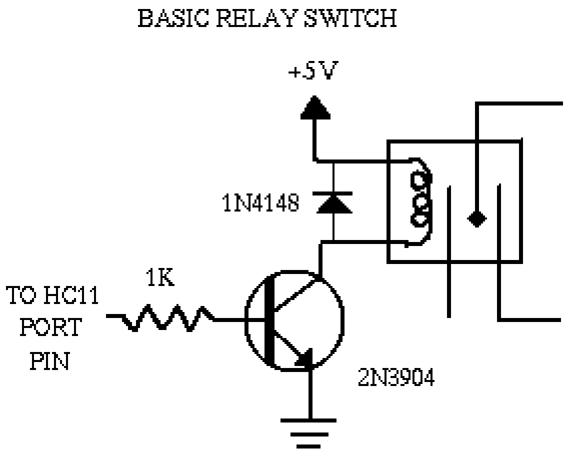


The input circuit (black loop) is switched off and no current flows through it until something (either a sensor or a switch closing) turns it on. The output circuit (blue loop) is also switched off.

When a small current flows in the input circuit, it activates the electromagnet (shown here as a red coil), which produces a magnetic field all around it.

The energized electromagnet pulls the metal bar in the output circuit toward it, closing the switch and allowing a much bigger current to flow through the output circuit.

The output circuit operates a high-current appliance such as a lamp or an electric motor.



A type of relay that can handle the high power required to directly control an electric motor or other loads is called a contactor. Solid-state relays control power circuits with no moving parts, instead using a semiconductor device to perform switching. Relays with calibrated operating characteristics and sometimes multiple operating coils are used to protect electrical circuits from overload or faults; in modern electric power systems these functions are performed by digital instruments still called "protective relays".

Magnetic latching relays require one pulse of coil power to move their contacts in one direction, and another, redirected pulse to move them back. Repeated pulses from the same input have no effect. Magnetic latching relays are useful in applications where interrupted power should not be able to transition the contacts.

Magnetic latching relays can have either single or dual coils. On a single coil device, the relay will operate in one direction when power is applied with one polarity, and will reset when the polarity is reversed. On a dual coil device, when polarized voltage is applied to the reset coil the contacts will transition. AC controlled magnetic latch relays have single coils that employ steering diodes to differentiate between operate and reset commands.

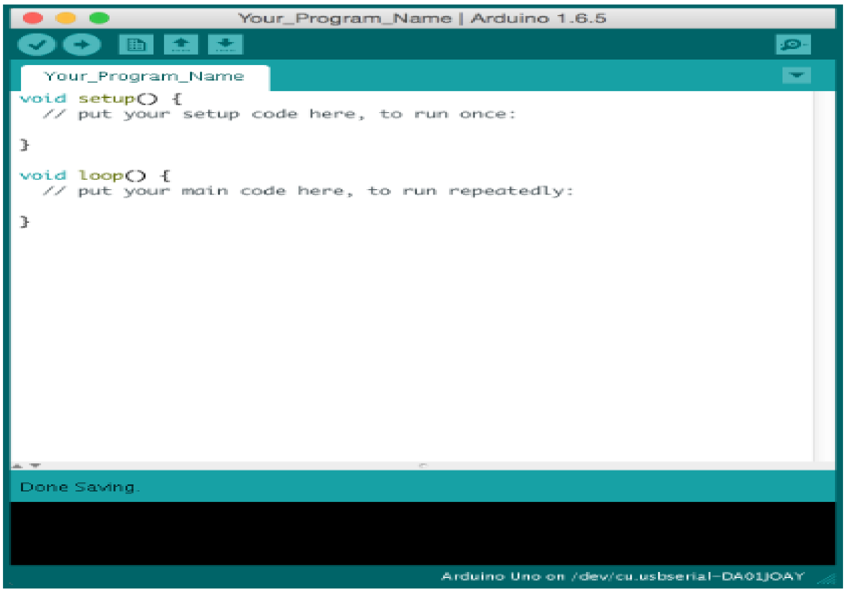
4. SOFTWARE IMPLEMENTATION

4.1 Arduino Software

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

The key features are:

- Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.
- You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).
- Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.
- Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.
- Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Done Saving.

Arduino Uno on /dev/cu.usbserial-DA01JOAY

Arduino Installation:

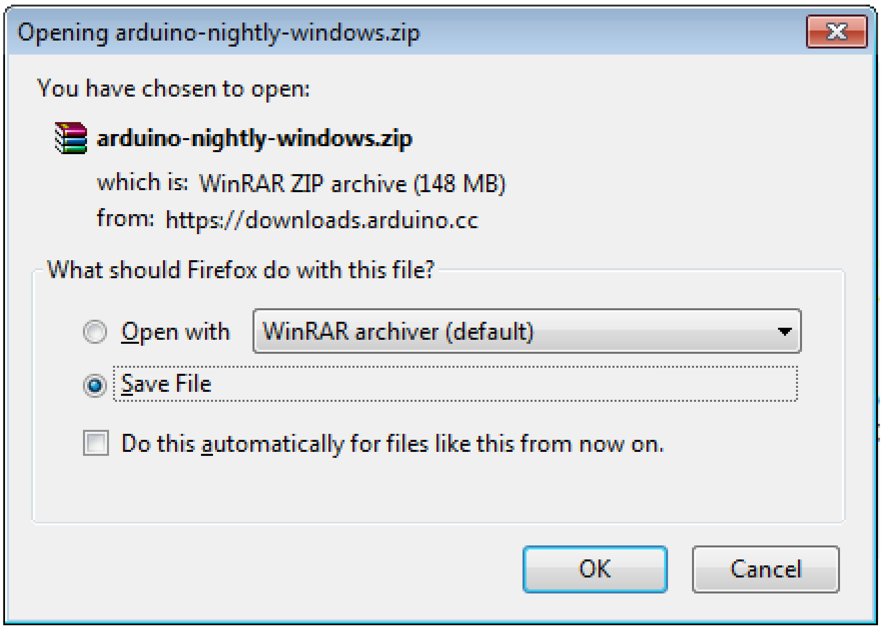
After learning about the main parts of the Arduino UNO board, we are ready to learn how to set up the Arduino IDE. Once we learn this, we will be ready to upload our program on the Arduino board. In this section, we will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

Step 1: First you must have your Arduino board (you can choose your favorite board) and USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega 2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind youwould connect to a USB printer as shown in the following image.

In case you use Arduino Nano, you will need an A to Mini-B cable instead as shown in the following image.

Step 2: Download Arduino IDE Software.

You can get different versions of Arduino IDE from the Download page on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.



Opening arduino-nightly-windows.zip

You have chosen to open:

arduino-nightly-windows.zip

which is: WinRAR ZIP archive (148 MB)

from: https://downloads.arduino.cc

What should Firefox do with this file?

○ Open with WinRAR archiver (default)

● Save File

□ Do this automatically for files like this from now on.

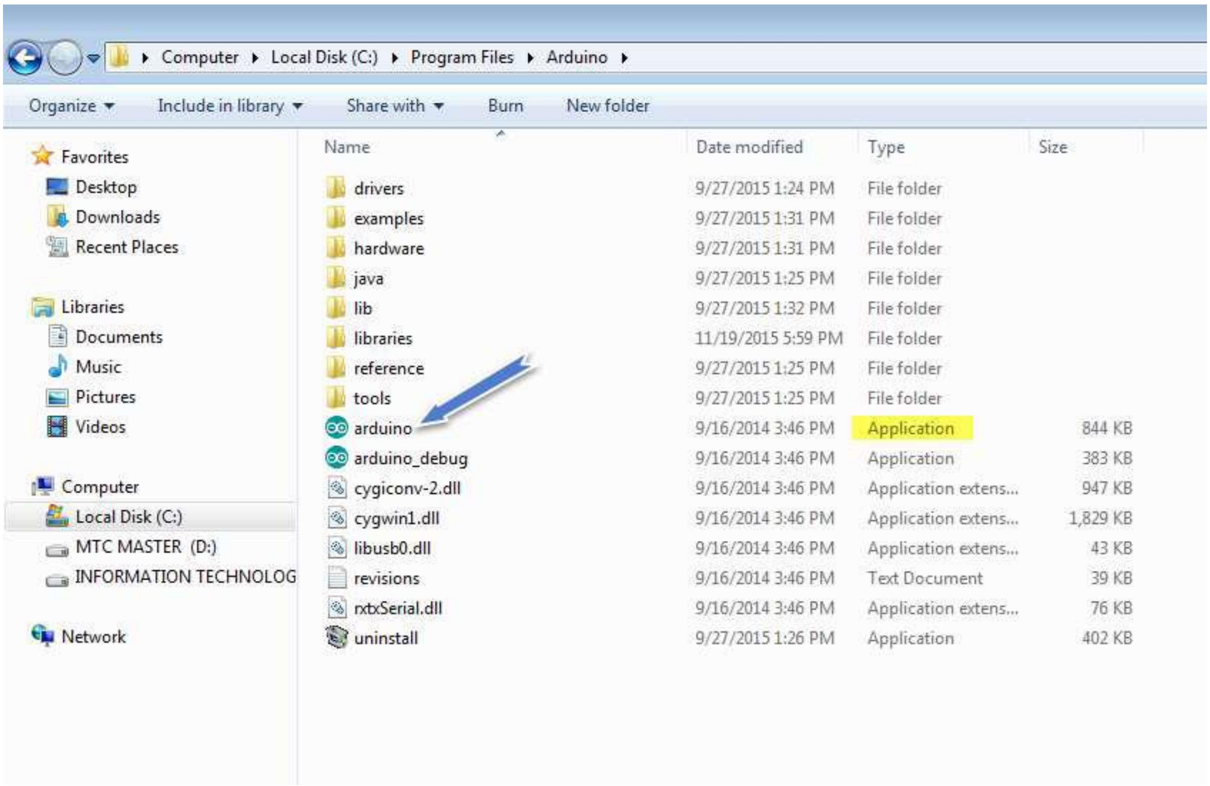
Step 3: Power up your board.

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port.

Connect the Arduino board to your computer using the USB cable. The green power LED (labeled PWR) should glow.

Step 4: Launch Arduino IDE.

After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Doubleclick the icon to start the IDE.



Name	Date modified	Type	Size
drivers	9/27/2015 1:24 PM	File folder	
examples	9/27/2015 1:31 PM	File folder	
hardware	9/27/2015 1:31 PM	File folder	
java	9/27/2015 1:25 PM	File folder	
lib	9/27/2015 1:32 PM	File folder	
libraries	11/19/2015 5:59 PM	File folder	
reference	9/27/2015 1:25 PM	File folder	
tools	9/27/2015 1:25 PM	File folder	
arduino	9/16/2014 3:46 PM	Application	844 KB
arduino_debug	9/16/2014 3:46 PM	Application	383 KB
cygiconv-2.dll	9/16/2014 3:46 PM	Application extens...	947 KB
cygwin1.dll	9/16/2014 3:46 PM	Application extens...	1,829 KB
libusb0.dll	9/16/2014 3:46 PM	Application extens...	43 KB
revisions	9/16/2014 3:46 PM	Text Document	39 KB
rxtxSerial.dll	9/16/2014 3:46 PM	Application extens...	76 KB
uninstall	9/27/2015 1:26 PM	Application	402 KB

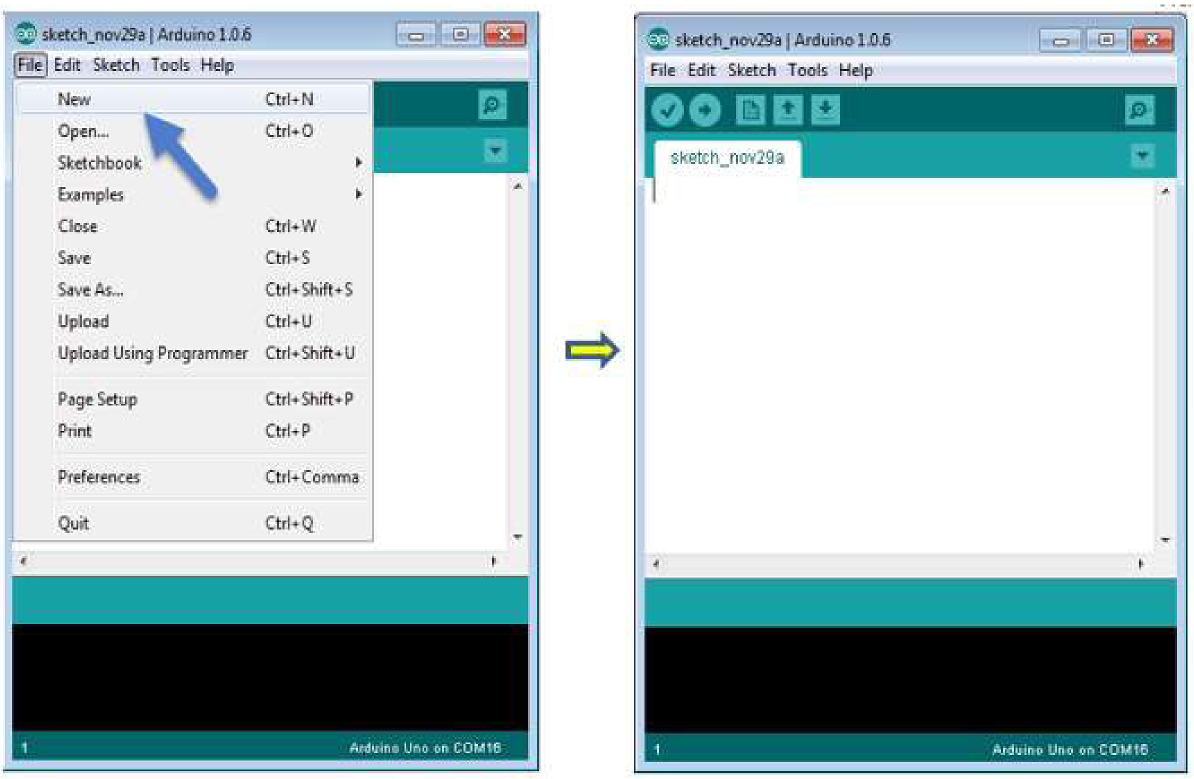
Step 5: Open your first project.

Once the software starts, you have two options:

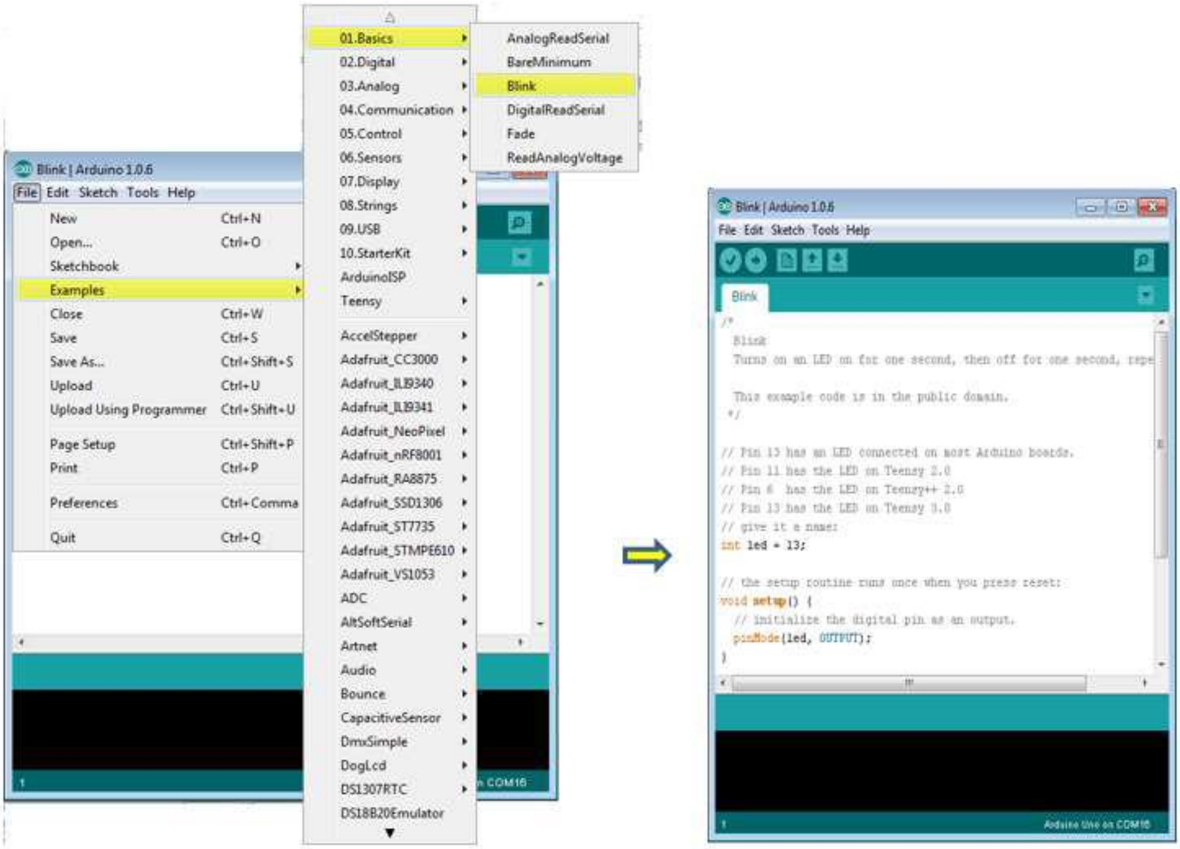
- Create a new project.

- Open an existing project example.

To create a new project, select File **--> New.**



To open an existing project example, select File -> Example -> Basics -> Blink.



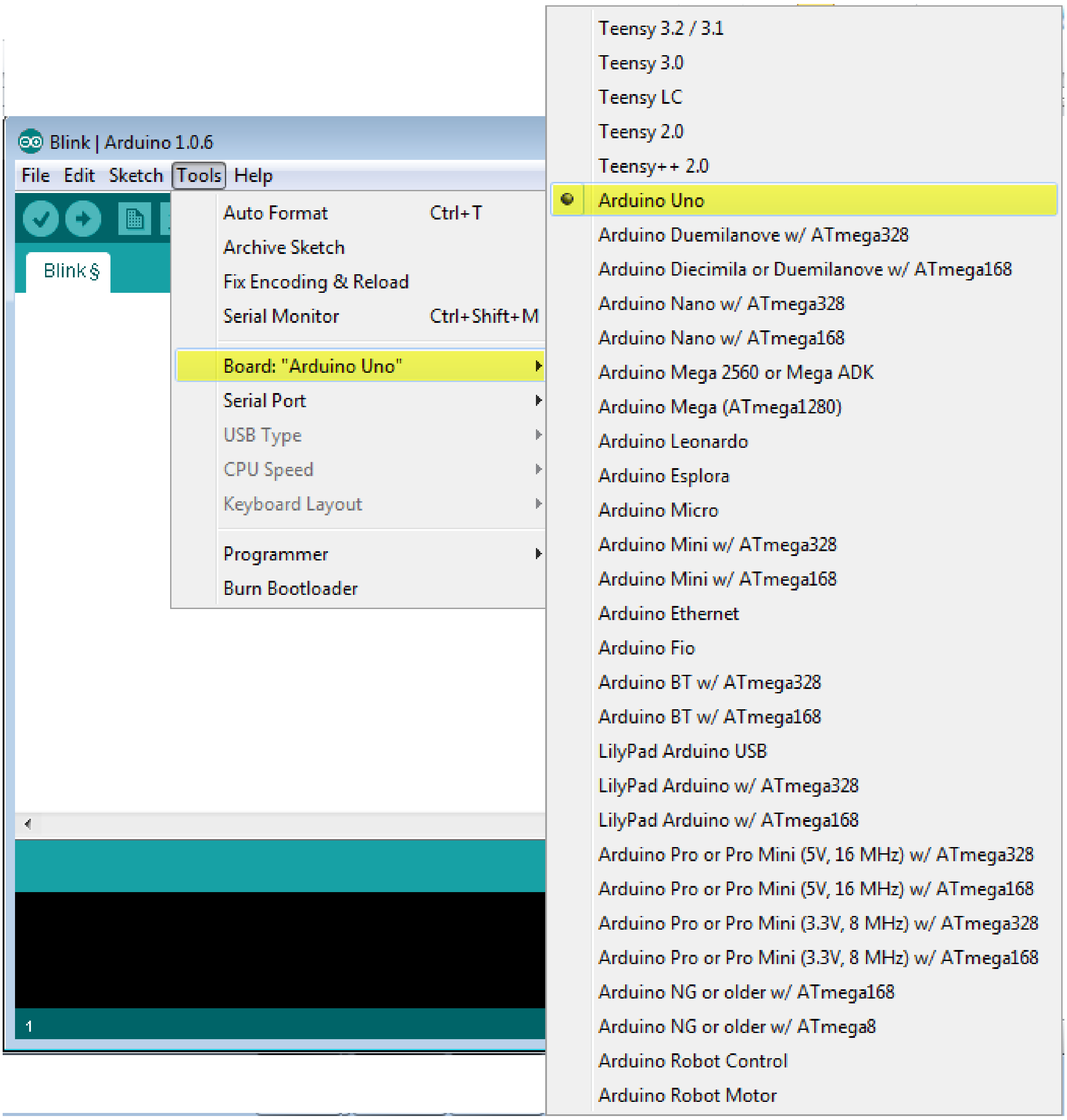
```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// Pin 11 has the LED on Teensy 2.0
// Pin 6  has the LED on Teensy++ 2.0
// Pin 13 has the LED on Teensy 3.0
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
```

Here, we are selecting just one of the examples with the name Blink. It turns the LED on and off with some time delay. You can select any other example from the list.

Step 6: Select your Arduino board.

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

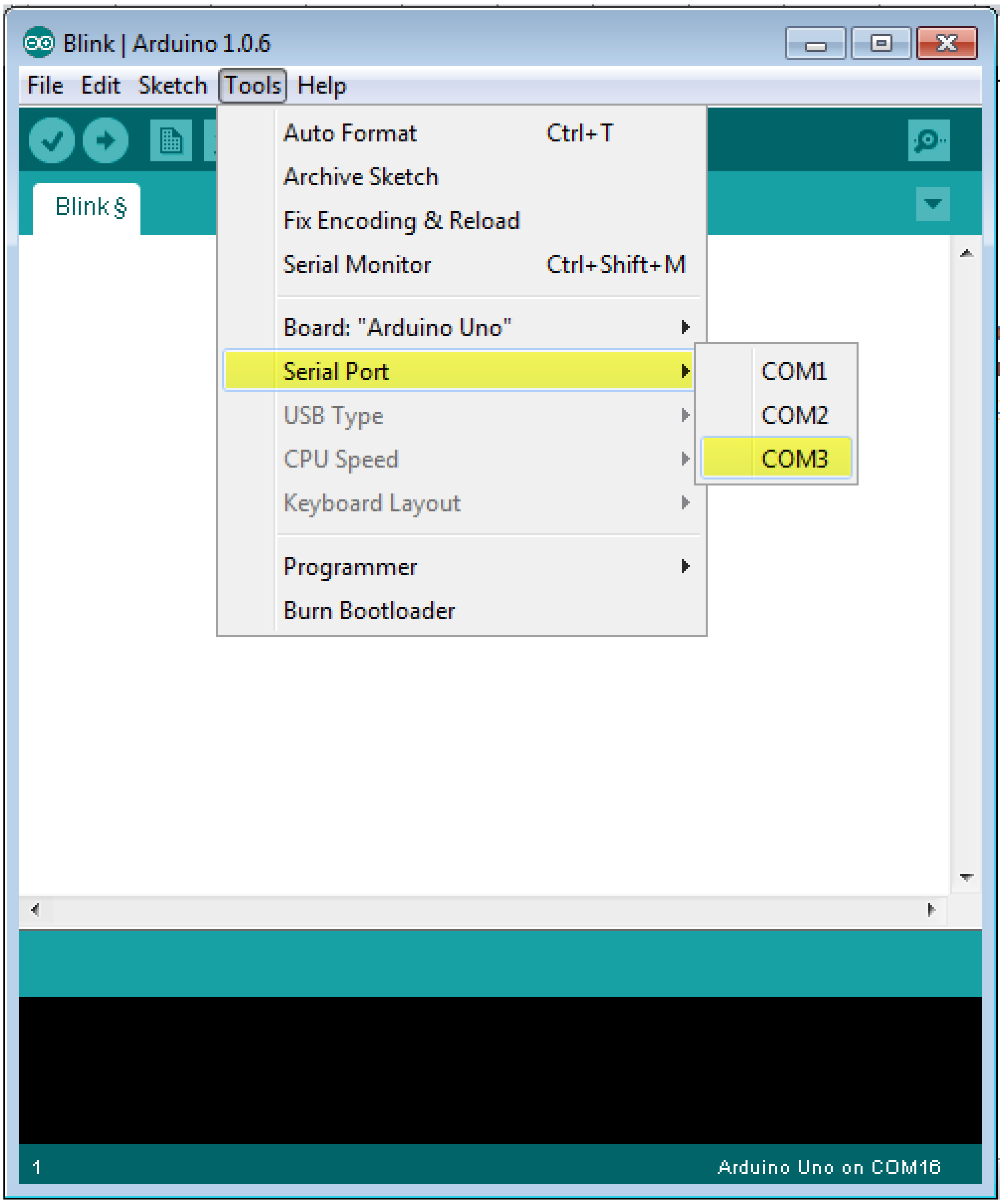
Go to Tools -> Board and select your board.


Blink | Arduino 1.0.6
File Edit Sketch Tools Help
Blink §
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board: "Arduino Uno"
Serial Port
USB Type
CPU Speed
Keyboard Layout
Programmer
Burn Bootloader
1
Teensy 3.2 / 3.1
Teensy 3.0
Teensy LC
Teensy 2.0
Teensy++ 2.0
● Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Leonardo
Arduino Esplora
Arduino Micro
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino USB
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
Arduino Robot Control
Arduino Robot Motor

Here, we have selected Arduino Uno board according to our tutorial, but you must select the name matching the board that you are using.

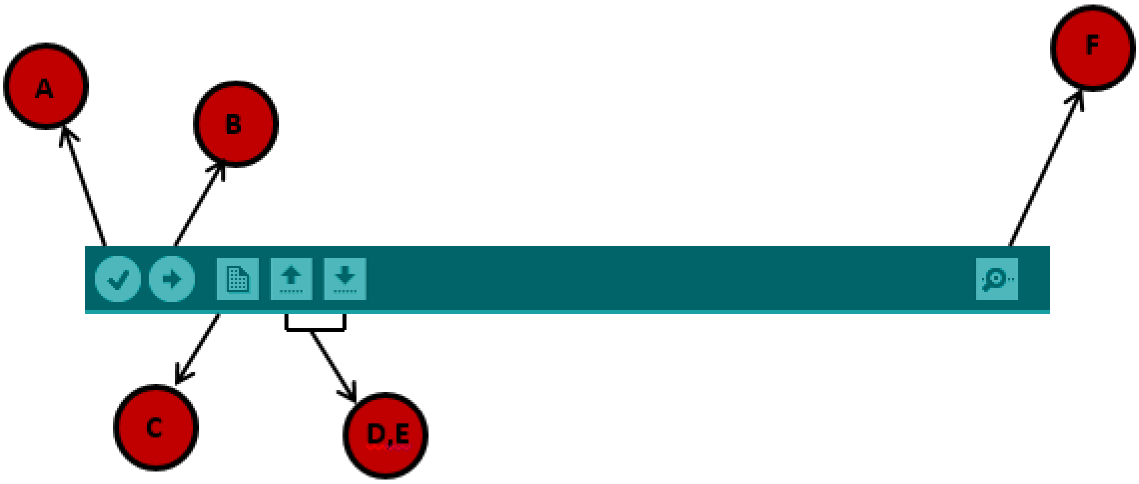
Step 7: Select your serial port.

Select the serial device of the Arduino board. Go to Tools -> Serial Port menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.



Step 8: Upload the program to your board.

Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.



A- Used to check if there is any compilation error.

B- Used to upload a program to the Arduino board.

C- Shortcut used to create a new sketch.

D- Used to directly open one of the example sketch.

E- Used to save your sketch.

F- Serial monitor used to receive serial data from the board and send the serial data to the board.

Now, simply click the "Upload" button in the environment. Wait a few seconds; you will see the RX and TX LEDs on the board, flashing. If the upload is successful, the message "Done uploading" will appear in the status bar.

Note: If you have an Arduino Mini, NG, or other board, you need to press the reset button physically on the board, immediately before clicking the upload button on the Arduino Software.

These instructions mostly show Windows software. Except when indicated, the software (should be) identical on all platforms. Linux will be added once I figure out how to get it working (yay)

Not much is needed for this lesson, just a USB cable and an Arduino. If you have an older Arduino you may also need an LED. Any LED is fine as long as it looks sorta like the photo, with a plastic bulb and two legs.

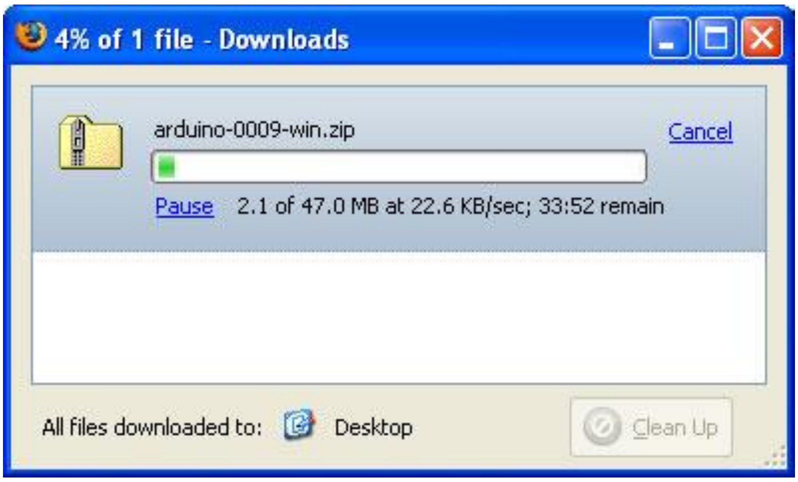
	Assembled Arduino board, preferrably a Diecimila (or whatever the latest version is)	Adafruit	**$35**
	USB Cable. Standard A-B cable is required. Any length is OK.	Adafruit Or any computer supply store	**$5**
	LED **-** Optional **Nearly any LED is OK, as long as it has two wire legs. This part is only required for** NG rev c **Arduinos (and maybe other older ones). Diecimila Arduino's**	**Any electronics supply store**	**$1**

	have this part 'built-in'		

Download the Software

The first thing to do is download the Arduino software.

Go to the Arduino Software Download page and grab the right file for your OS. As of Sept 2007 the version is **009** but you should use whatever is most recent.
The packages are quite large, 30-50 MB so it may take a while to finish


4% of 1 file - Downloads

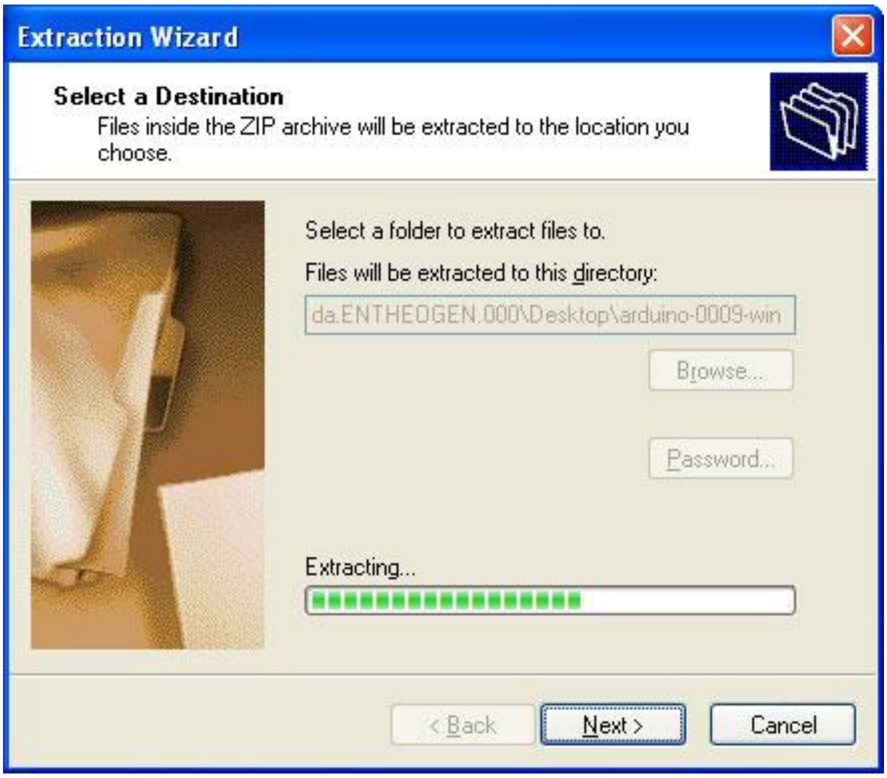
arduino-0009-win.zip

2.1 of 47.0 MB at 22.6 KB/sec; 33:52 remain

All files downloaded to: Desktop

Unpack and Install

Extract the package onto the Desktop


Extraction Wizard

Select a Destination

Files inside the ZIP archive will be extracted to the location you choose.

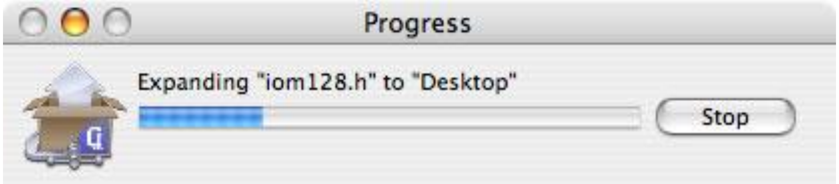
Select a folder to extract files to.

Files will be extracted to this directory:

da.ENTHEOGEN.000\Desktop\arduino-0009-win

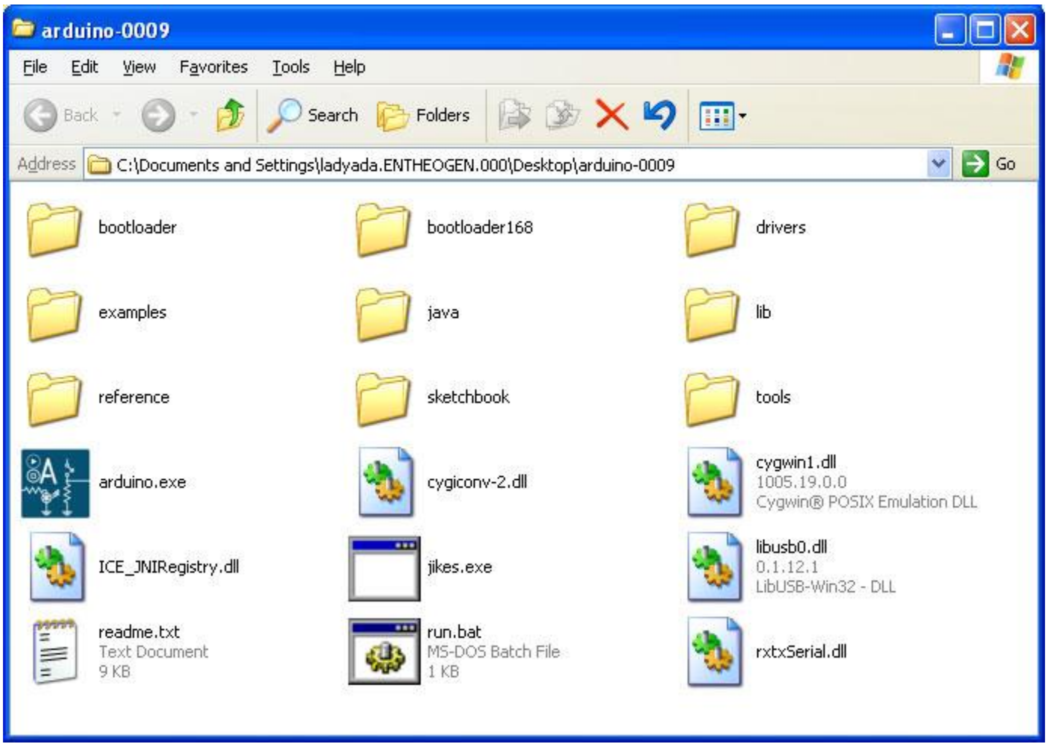
Extracting...

Windows



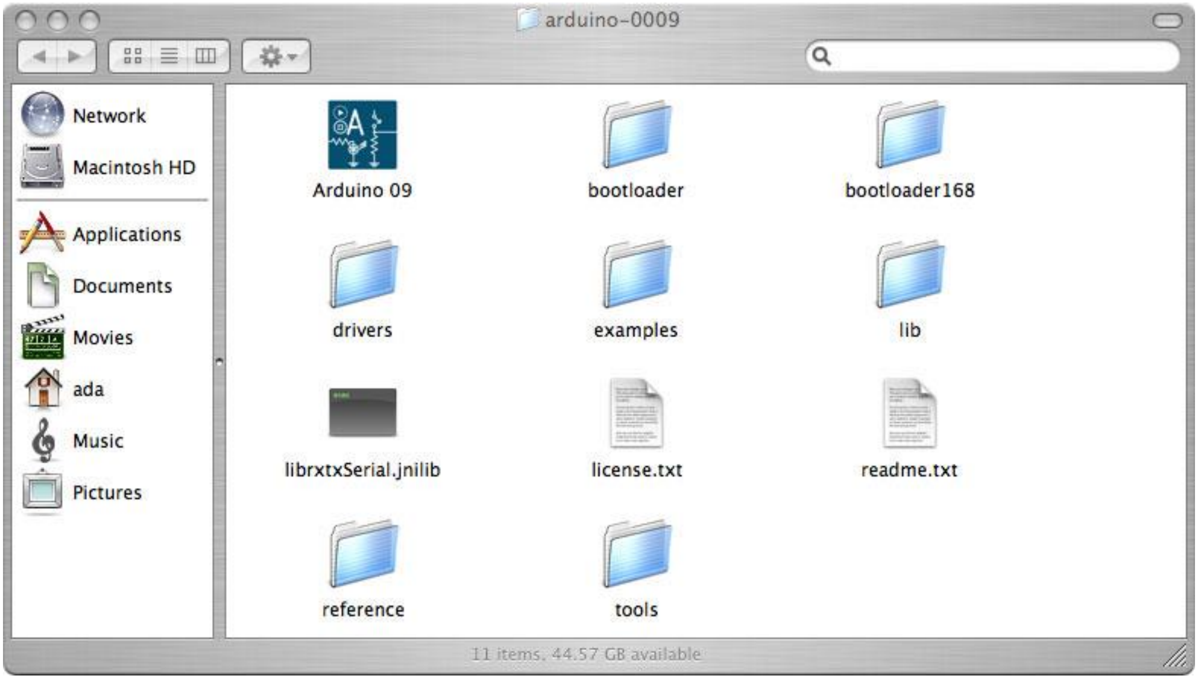
Expanding "iom128.h" to "Desktop"

Mac OS X



C:\Documents and Settings\ladyada.ENTHEOGEN.000\Desktop\arduino-0009

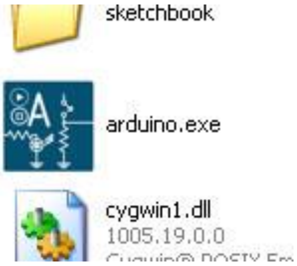
Windows



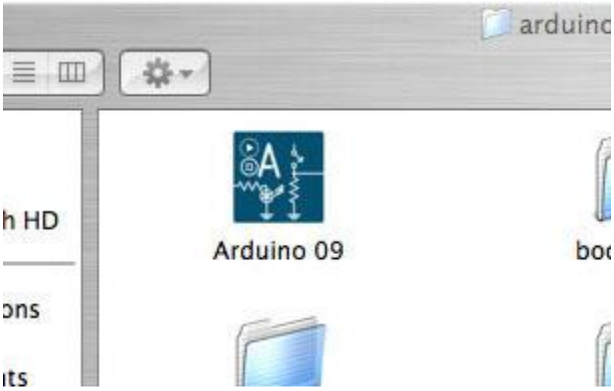
Mac OS X

Startup!

Double click the Arduino software icon

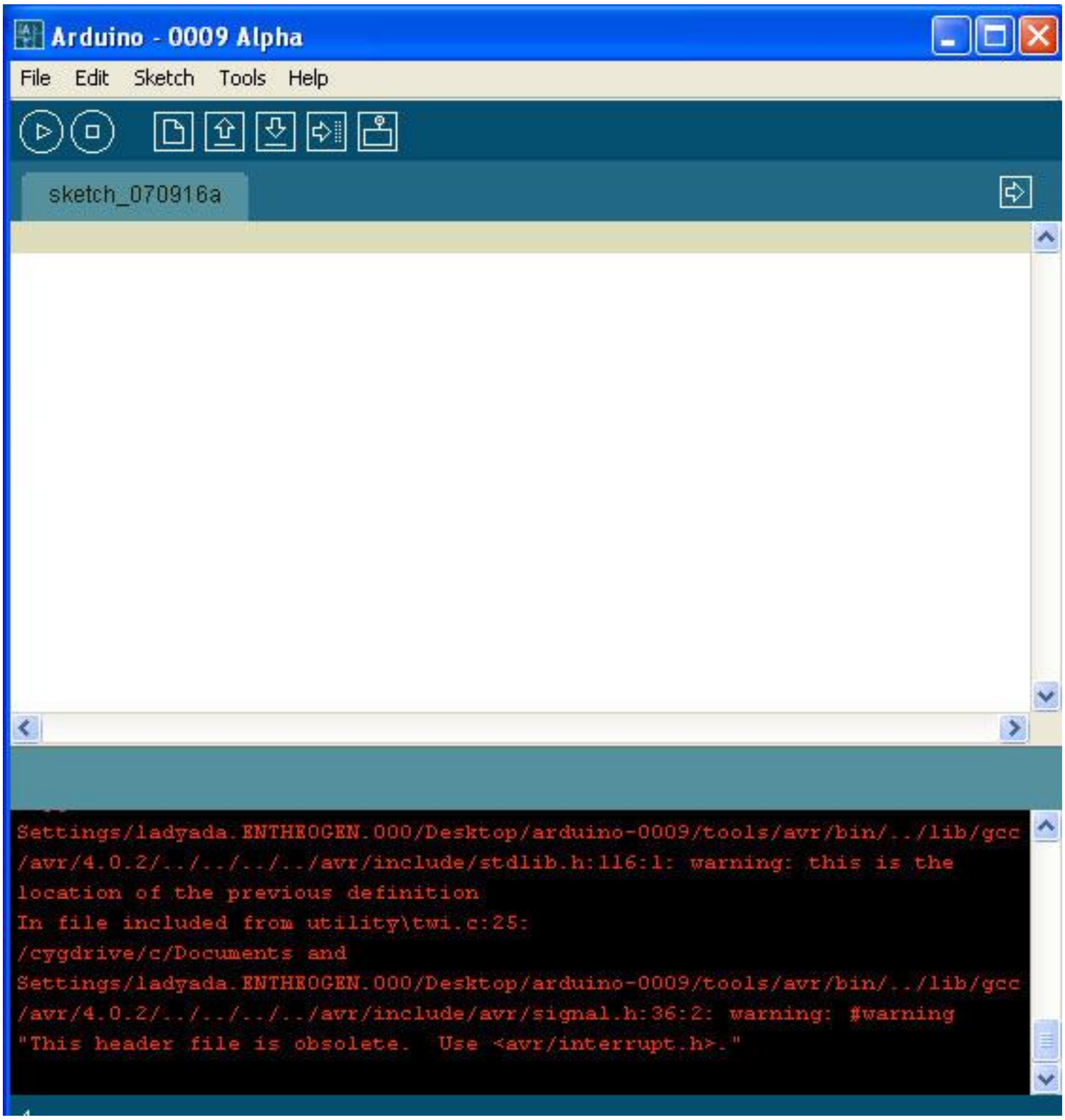


Windows



Mac OS X

To open up the workspace

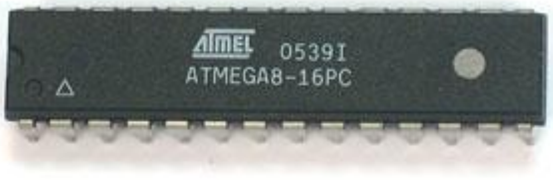


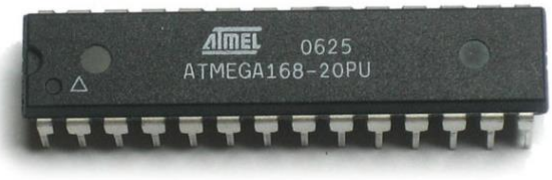
```
Settings/ladyada.ENTHEOGEN.000/Desktop/arduino-0009/tools/avr/bin/../lib/gcc
/avr/4.0.2/../../../../avr/include/stdlib.h:116:1: warning: this is the
location of the previous definition
In file included from utility\twi.c:25:
/cygdrive/c/Documents and
Settings/ladyada.ENTHEOGEN.000/Desktop/arduino-0009/tools/avr/bin/../lib/gcc
/avr/4.0.2/../../../../avr/include/avr/signal.h:36:2: warning: #warning
"This header file is obsolete.  Use <avr/interrupt.h>."
```

I think I get the red error text shown because I already have Arduino installed. Either way, it isn't a problem if you do or don't see it.

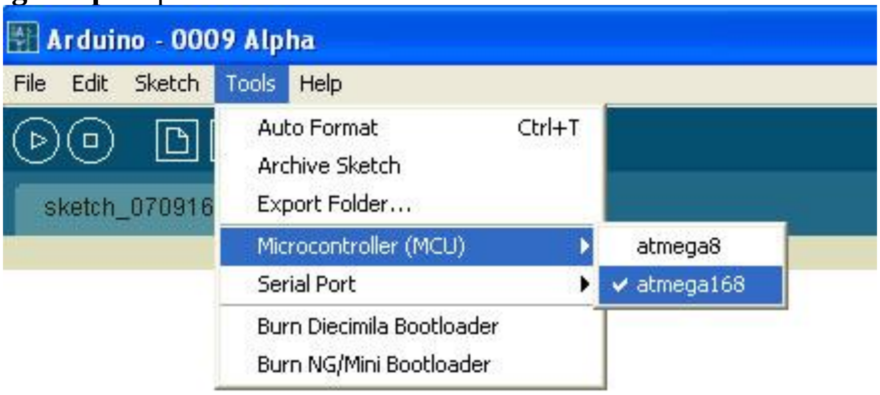
Select chip

The first step is to configure the Arduino software for the correct chip. Almost all Arduinos use the ATmega168, but there's a chance you have an ATmega8. Look for the chip on the Arduino that looks like this:





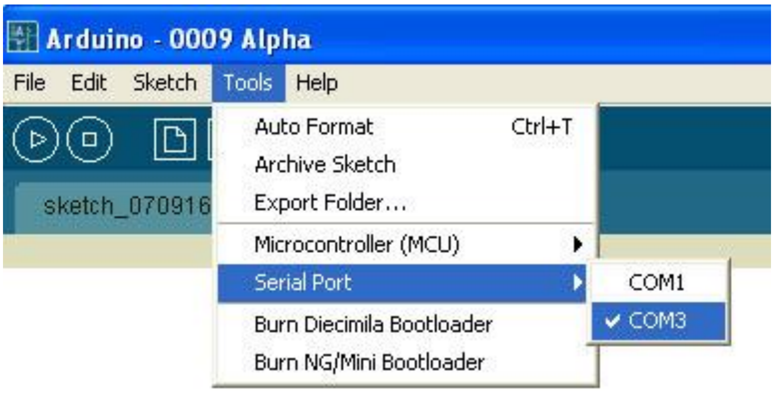
If the text says ATMEGA8-16P then you have an **atmega8** chip. If the text says ATMEGA168-20P then you have an**atmega168** chip. If it says "ATMEGA328P-20P" you have an **atmega328p** chip



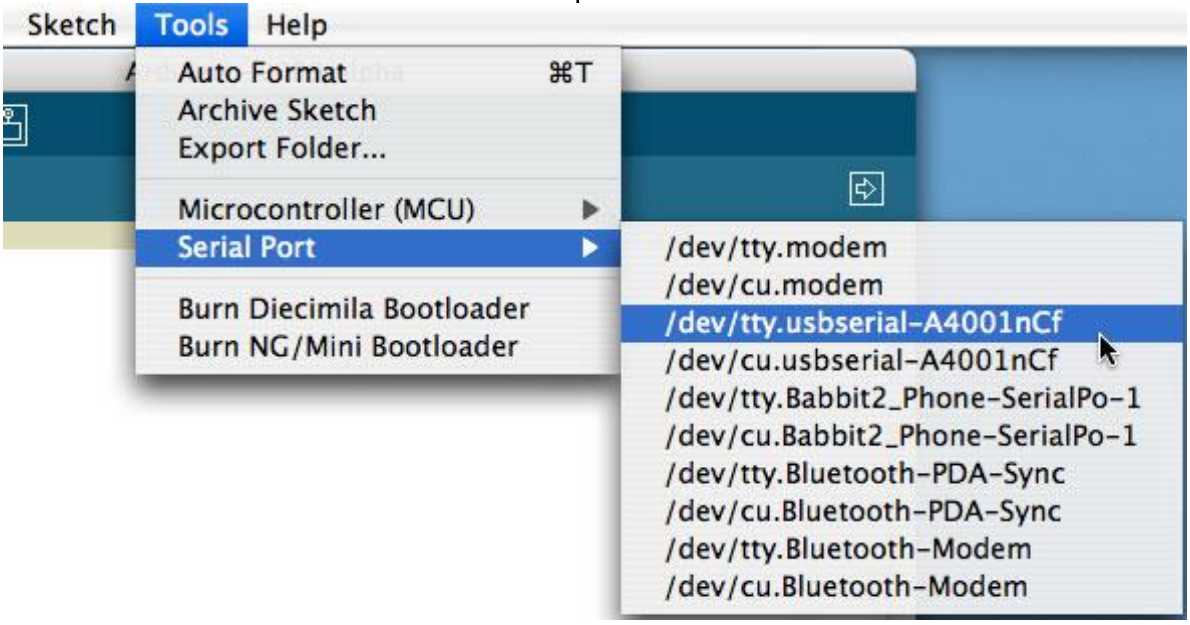
Make sure the correct chip is selected (this picture is really old, will be fixed soon). This preference is saved so you only have to set it once, the program will remember next time it's run.

Select port

Next, its time to configure the Serial Port (also known as the COM Port). Go back to lesson 0 to remind yourself of which port it is. On a PC it will probably be something like **COM3** or **COM4**. On a Mac it will be something like**tty.usbserial-xxxxx**



Windows port selection


Sketch
Tools
Help
Auto Format
⌘T
Archive Sketch
Export Folder...
Microcontroller (MCU)
Serial Port
Burn Diecimila Bootloader
Burn NG/Mini Bootloader
/dev/tty.modem
/dev/cu.modem
/dev/tty.usbserial-A4001nCf
/dev/cu.usbserial-A4001nCf
/dev/tty.Babbit2_Phone-SerialPo-1
/dev/cu.Babbit2_Phone-SerialPo-1
/dev/tty.Bluetooth-PDA-Sync
/dev/cu.Bluetooth-PDA-Sync
/dev/tty.Bluetooth-Modem
/dev/cu.Bluetooth-Modem

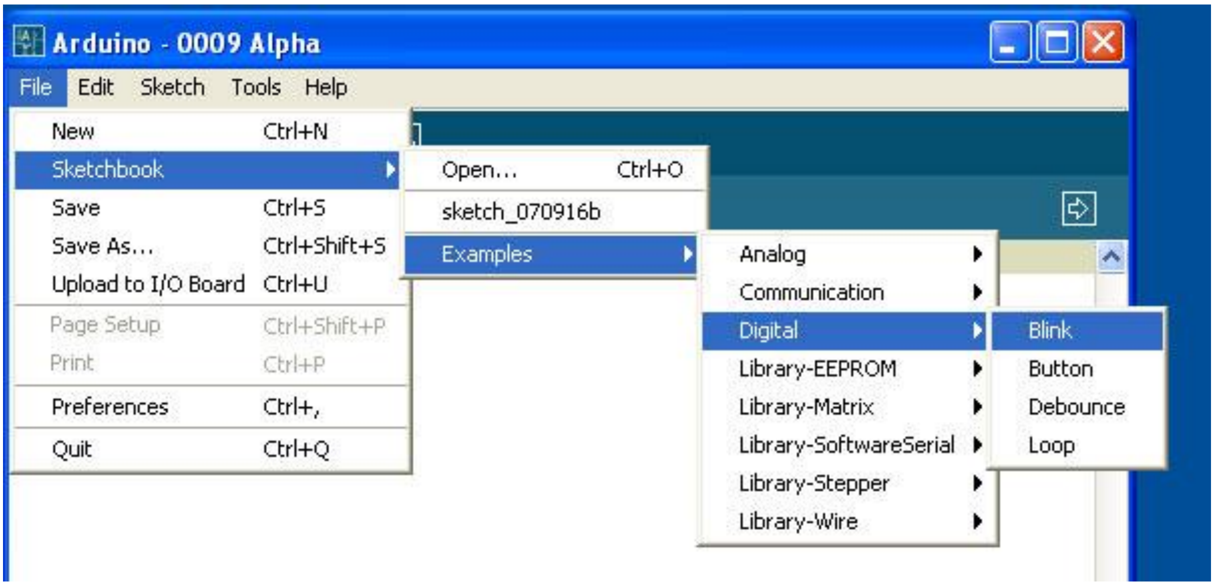
Mac port selection

This preference is saved so you only have to set it once, the program will remember next time it's run.

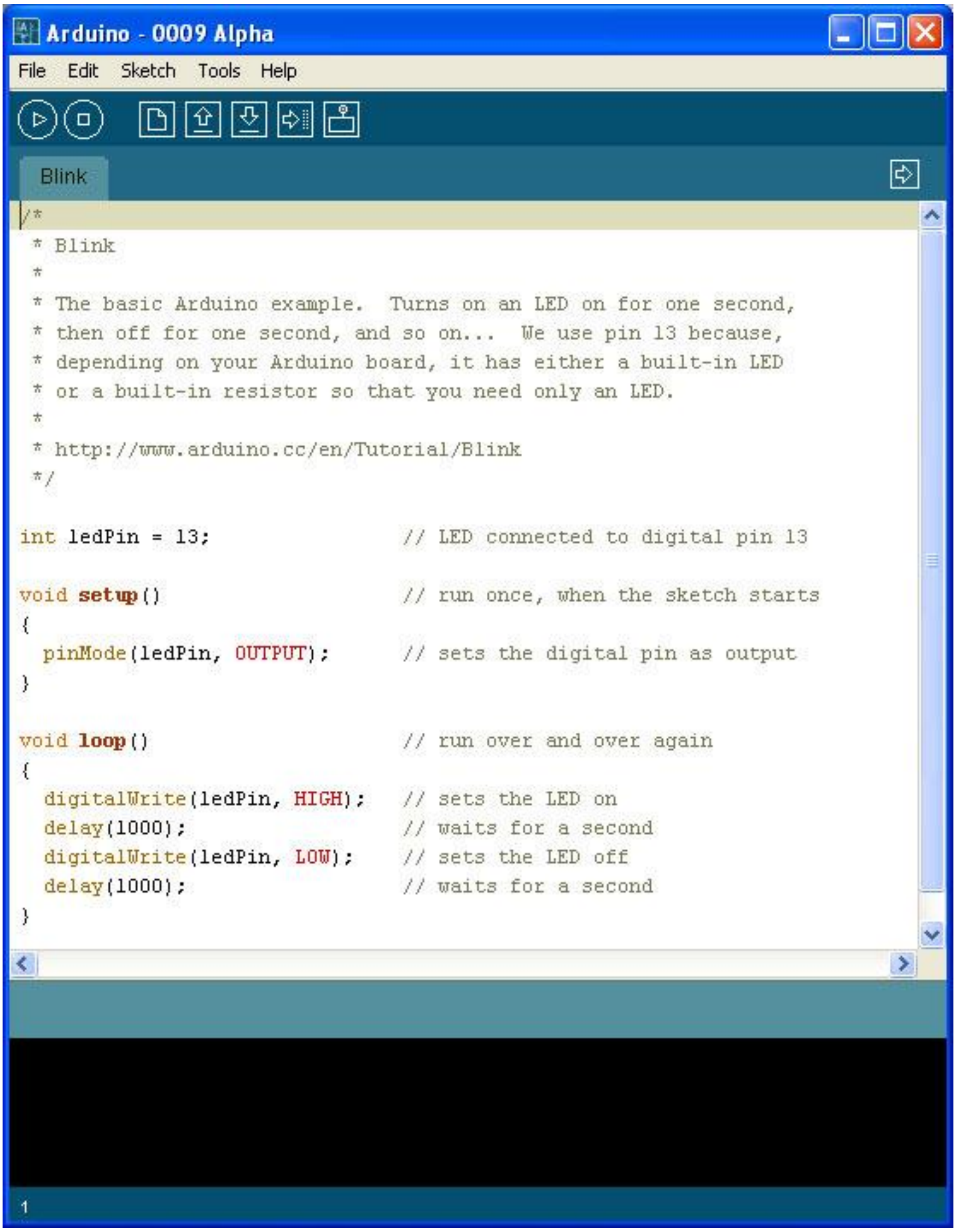
However, if you have multiple Arduino's, they may be assigned difference COM ports. So every time you plug in a new Arduino, double check that the correct port is selected.

Open blink sketch

Sketches are little scripts that you can send to the Arduino to tell it how to act. Let's open up an **Example Sketch**. Go to the **File menu -> Sketchbook -> Examples -> Digital -> Blink**



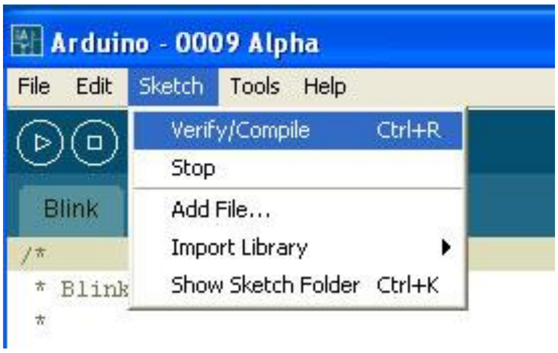
The window should now look like this, with a bunch of text in the formerly empty white space and the tab **Blink** above it



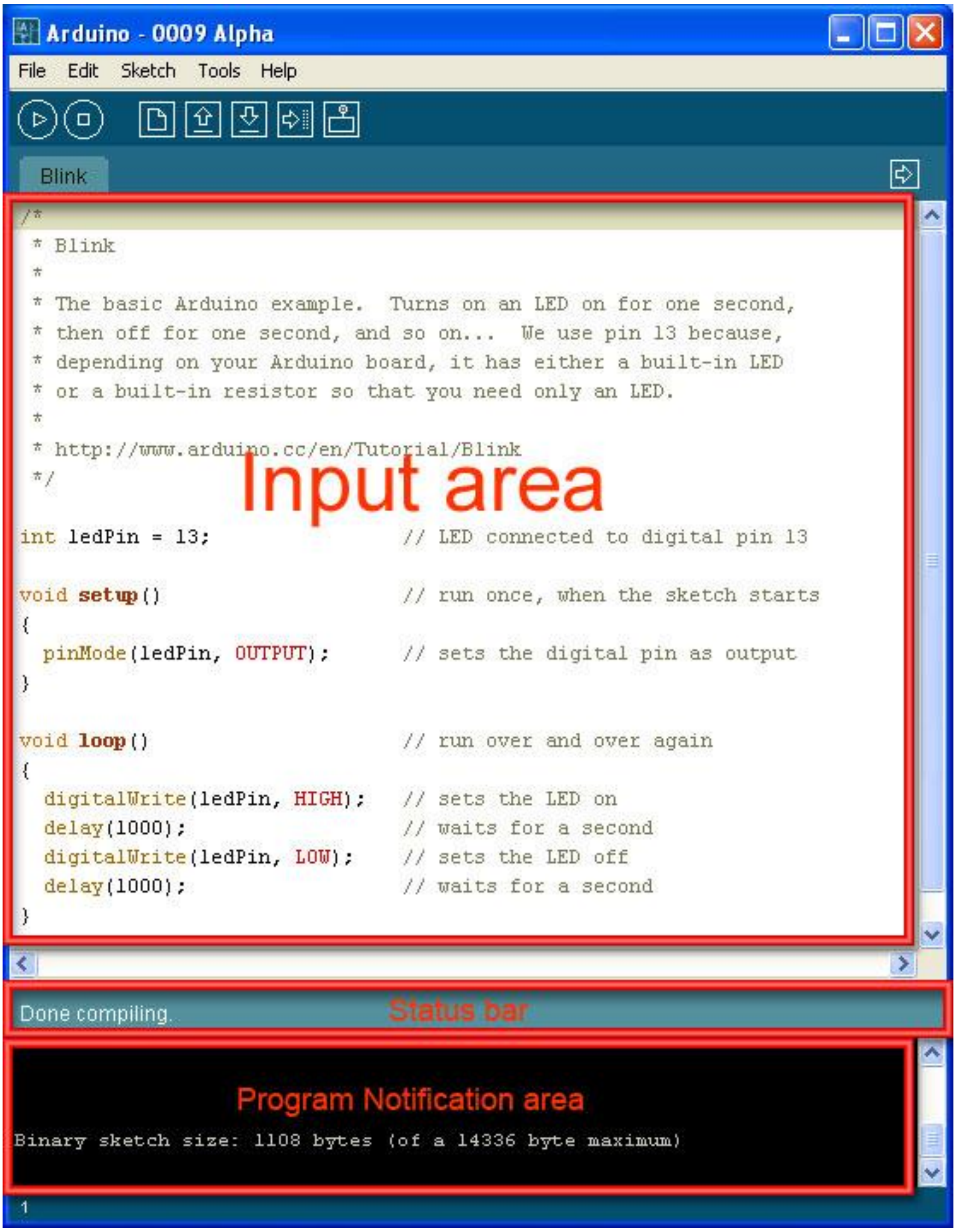
```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink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

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
}
```

Verify / Compile

The first step to getting a **Sketch** ready for transfer over to the Arduino is to **Verify/Compile** it. That means check it over for mistakes (sort of like editing) and then translate it into an application that is compatible with the Arduino hardware.



After a few seconds, you should see the message **Done compiling.** in the **Status Bar** and **Binary Sketch Size:** in the **Notification area.** This means the sketch was well-written and is ready for uploading to the Arduino board!



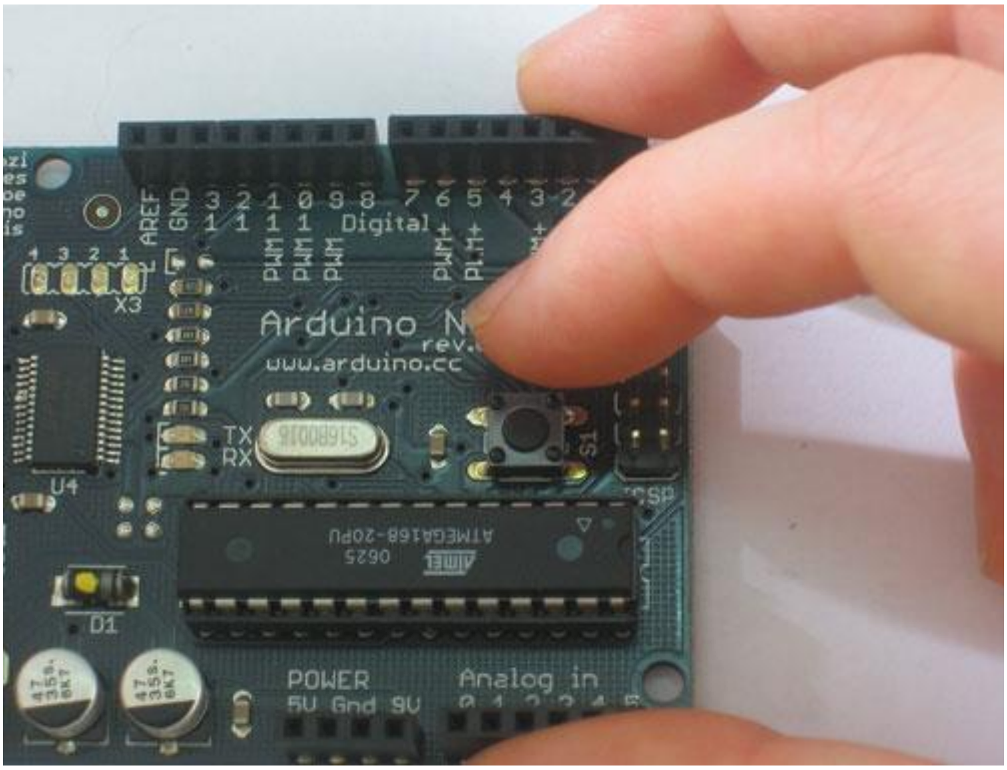
```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink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

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
}
```

Done compiling.

```
Binary sketch size: 1108 bytes (of a 14336 byte maximum)
```

Reset (NG only)

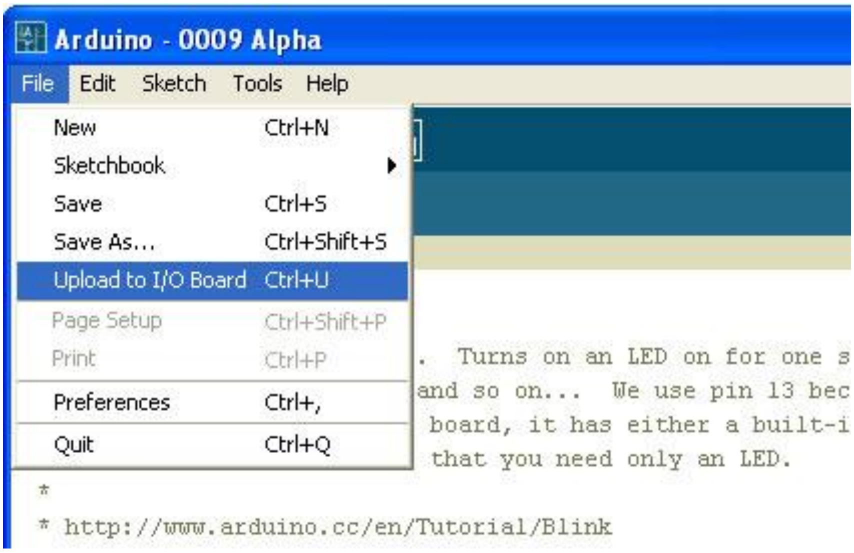
To tell the Arduino that it should prepare itself for a new Sketch upload, you must reset the board. Diecimila Arduino's have built-in auto-reset capability, so you don't need to do anything. Older Arduinos, such as NG, must be manually reset before uploading a sketch. To do that simply press the black button on the right hand side of the board, shown here.



Upload

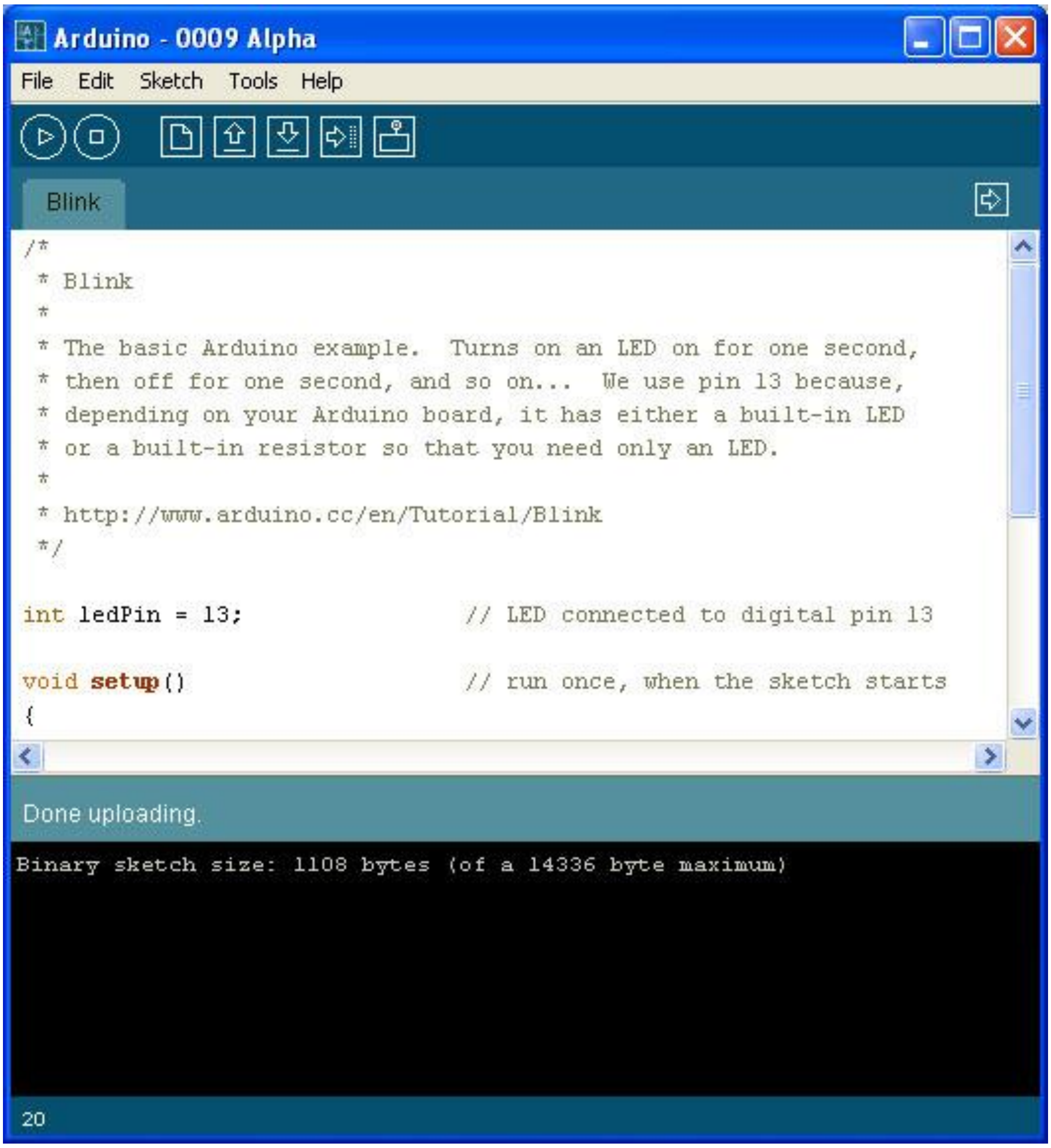
Now it's time to upload. Make sure the Arduino is plugged in, the green light is on and the correct Serial Port is selected.

If you have an NG Arduino, press the **Reset Button** now, just before you select the **Upload** menu item.
Select **Upload to I/O Board** from the **File menu**



```
.  Turns on an LED on for one s
and so on...  We use pin 13 bec
 board, it has either a built-i
 that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
```

After a few seconds you should get this screen, with the message **Done uploading.** in the status bar.

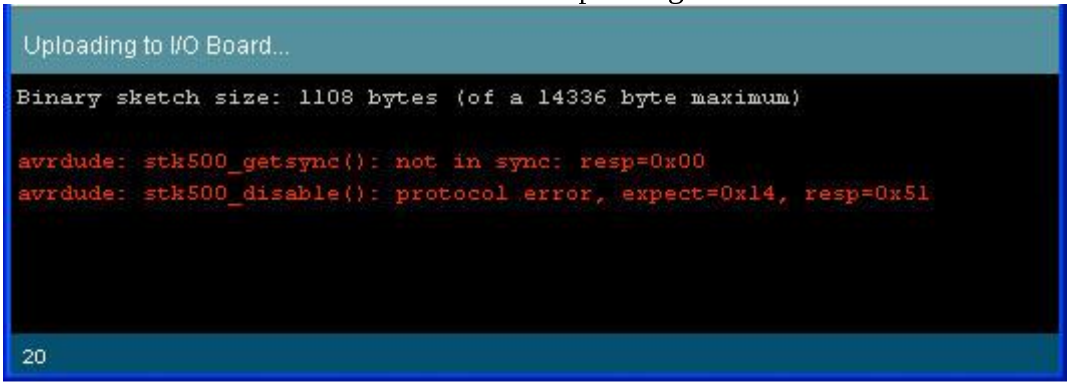


```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink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

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

void setup()                    // run once, when the sketch starts
{
```

Done uploading.

```
Binary sketch size: 1108 bytes (of a 14336 byte maximum)
```

If you get the following error message "**avrdude: stk500_getsync(): not in sync: resp=0x00"** that means that the Arduino is not responding



Uploading to I/O Board...

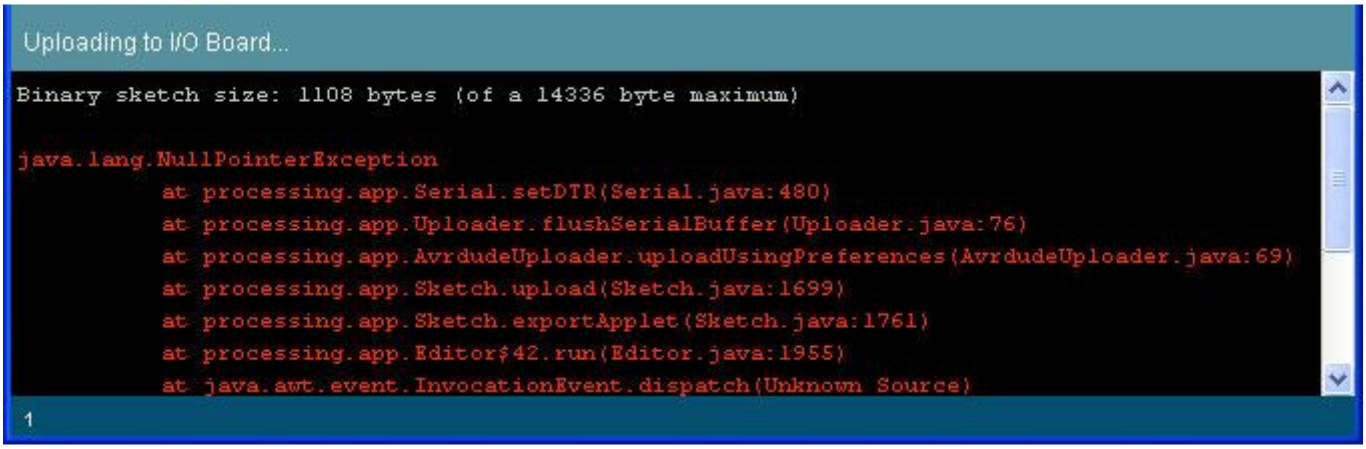
```
Binary sketch size: 1108 bytes (of a 14336 byte maximum)

avrdude: stk500_getsync(): not in sync: resp=0x00
avrdude: stk500_disable(): protocol error, expect=0x14, resp=0x51
```

Then check the following:

- **If you have a NG Arduino, did you press reset just before selecting** Upload **menu item?**
- **Is the correct Serial Port selected?**

- **Is the correct driver installed?**
- **Is the chip inserted into the Arduino properly? (If you built your own arduino or have burned the bootloader on yourself)**
- **Does the chip have the correct bootloader on it? (If you built your own arduino or have burned the bootloader on yourself)**

If you get the following error message:



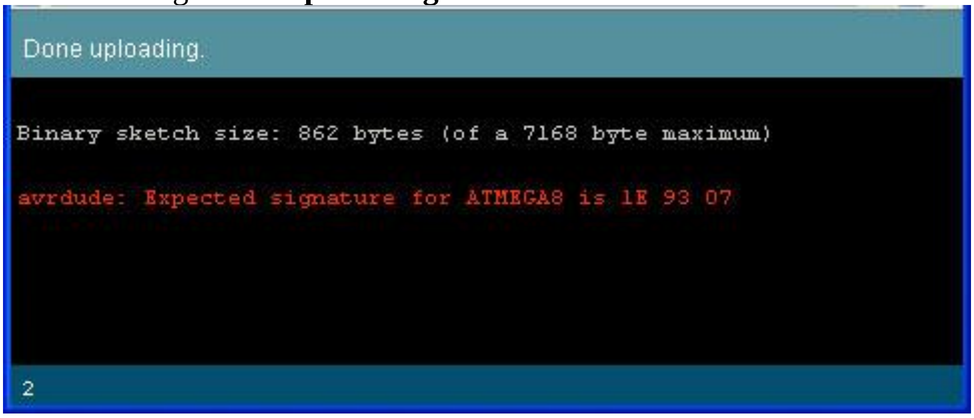
Uploading to I/O Board...

```
Binary sketch size: 1108 bytes (of a 14336 byte maximum)

java.lang.NullPointerException
        at processing.app.Serial.setDTR(Serial.java:480)
        at processing.app.Uploader.flushSerialBuffer(Uploader.java:76)
        at processing.app.AvrdudeUploader.uploadUsingPreferences(AvrdudeUploader.java:69)
        at processing.app.Sketch.upload(Sketch.java:1699)
        at processing.app.Sketch.exportApplet(Sketch.java:1761)
        at processing.app.Editor$42.run(Editor.java:1955)
        at java.awt.event.InvocationEvent.dispatch(Unknown Source)
```

It means you don’t cm have a serial port selected, go back and verify that the correct driver is installed (lesson 0)

and that you have the correct serial port selected in the menu.

If you get the following error **Expected signature for ATMEGA**

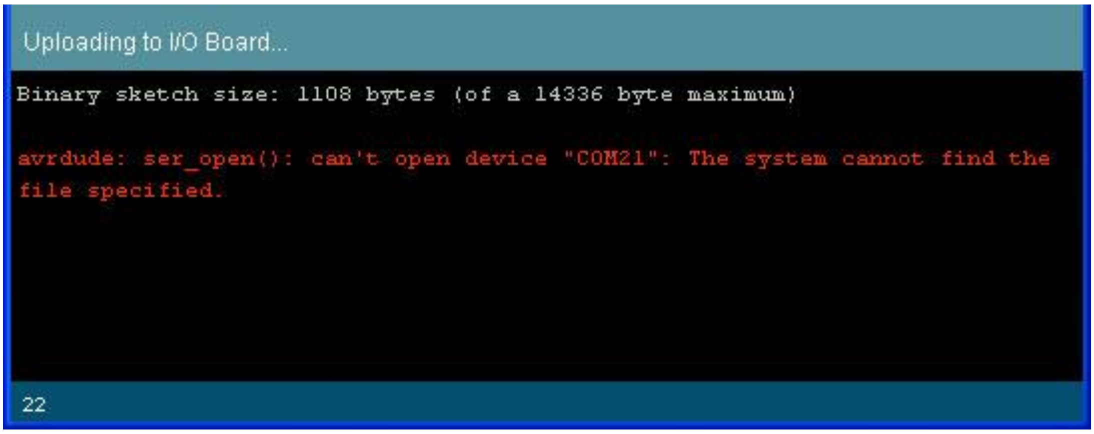


Done uploading.

```
Binary sketch size: 862 bytes (of a 7168 byte maximum)

avrdude: Expected signature for ATMEGA8 is 1E 93 07
```

Then you have either the incorrect chip selected in the **Tools** menu or the wrong bootloader burned onto the chip

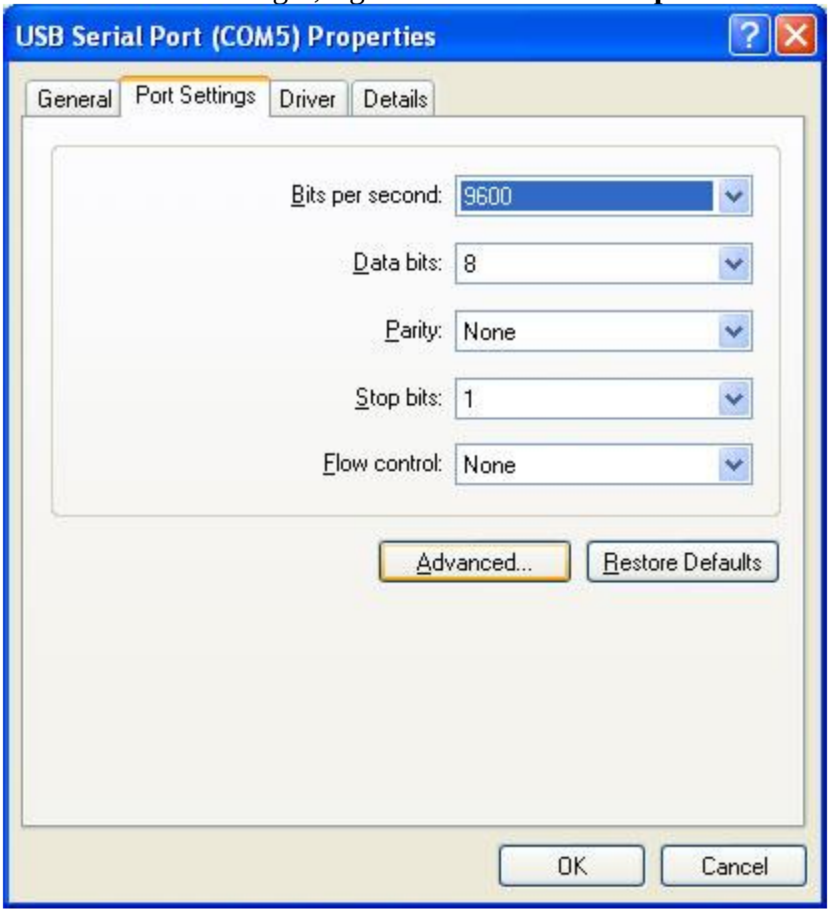
If you get the following error: **can't open device "COM10": The system cannot find the file specified** (under Windows, COM port value may vary)



```
Uploading to I/O Board...

Binary sketch size: 1108 bytes (of a 14336 byte maximum)

avrdude: ser_open(): can't open device "COM21": The system cannot find the
file specified.

22
```

It means that you have too many COM ports (maybe you've got 9 Arduinos?) You should make sure that the port is numbered as low as possible. You can use a program like FTClean to clear out old COM ports you aren't using anymore. Once you've cleaned out the ports, you'll have to reinstall the driver again (see lesson 0).

Alternately, if you're sure that the ports are not used for something else but are left over from other USB devices, you can simply change the COM port using the **Device Manager**. Select the USB device in the Device Manager, right click and select **Properties**



USB Serial Port (COM5) Properties

Bits per second: 9600

Data bits: 8

Parity: None

Stop bits: 1

Flow control: None

Then click **Advanced**... and in the next window change the COM port to something like **COM4** or **COM5**. Don't forget to select the new port name in the Arduino software. The lower port names may say **(in use**) but as long as the other USB devices aren't plugged in, it shouldn't be a problem. This is a little riskier than just using FTClean...

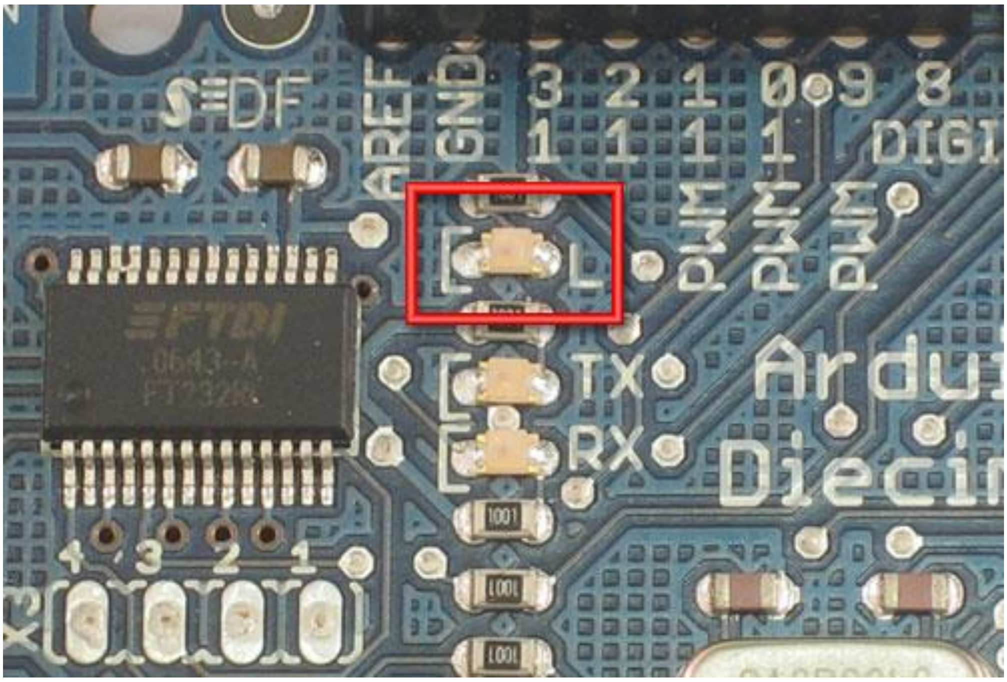
Video of all steps

Here is a video showing the timing of the steps described so far.

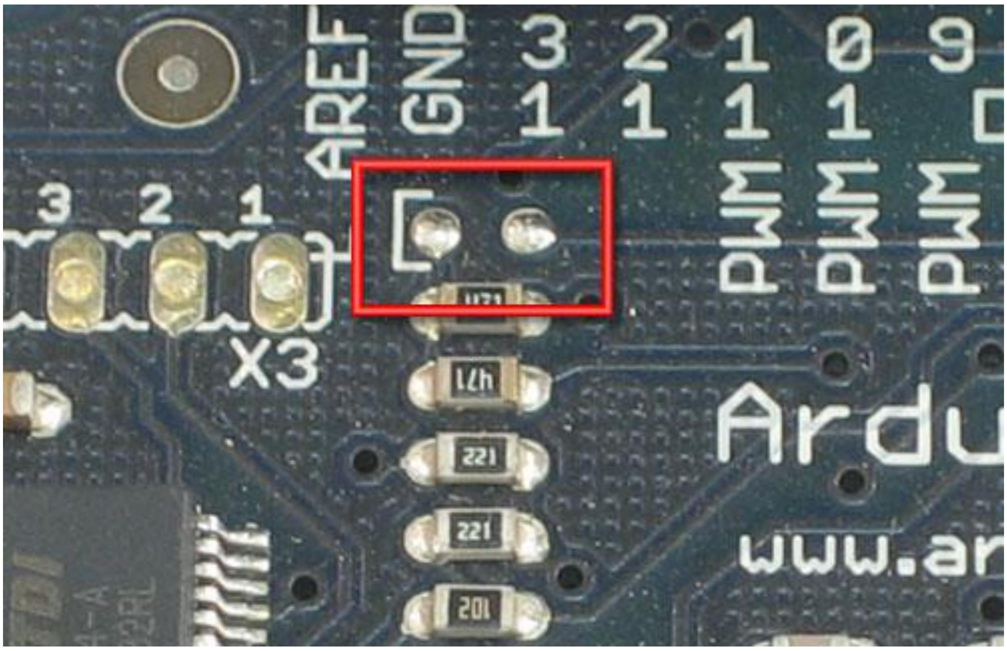
Insert LED (NG Arduinos)

Some older Arduinos don't have a built in LED, its easy to tell if yours does or not

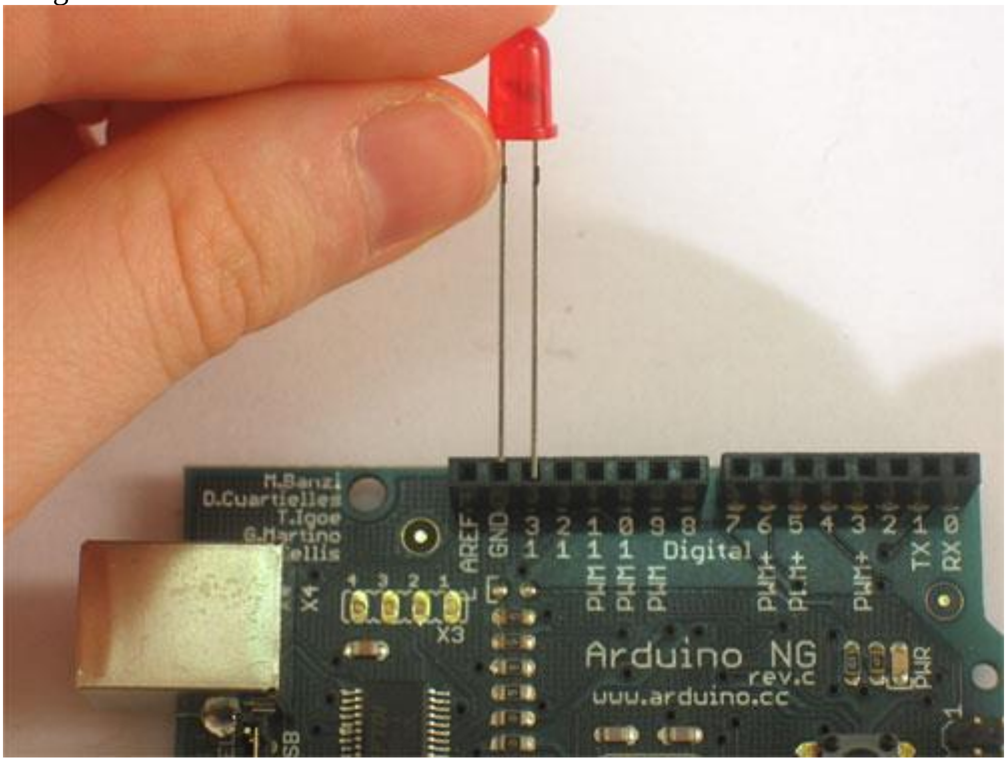
If you have a Diecimila or other Arduino with a built in LED you will see a translucent part as shown



If you have an NG rev C or other Arduino without an LED, the translucent part will not be there, and instead you will see two silver dots



If you don't have an LED, you'll need to add your own. Any LED will do, as long as it has two legs and kind looks like the one shown here. LEDs are **directional** components. That means if you put it in backwards it will not work! To help you put the LED in right, the LED factory cuts the legs at different lengths. The longer leg goes in the hole marked **13**and the shorter one goes in the hole marked **GND**



CONCLUSION

The project **“IOT BASED HOME AUTOMATION”** has been successfully designed and tested. It has been developed by integrating features of all the hardware components used. Presence of every module has been reasoned out and placed carefully thus contributing to the best working of the unit.

Secondly, using highly advanced IC’s and with the help of growing technology the project has been successfully implemented.