**NON-INVASIVE DIABETES MONITORING SYSTEM**

**Abstract:**

Diabetes is a common chronic disease in mostly all countries worldwide. Monitoring of glucose level of blood is important to avoid complications of diabetic and damage to organs. Since invasive method of glucose level measurement is painful and causes damage to nerves, non-invasive method is used as an alternative.

The variation in the intensity of glucose is estimated by the light received from the photo detector is used to determine the glucose level of blood. The measured glucose level is displayed in LCD display and also transmitted to the cloud server which is created in the mobile phone to display and store data via GSM.

**INTRODUCTION**

Diabetes Mellitus is one of the common life-threatening diseases in the world. Diabetes in general is known to increase blood glucose concentration which further introduces variations in the individual’s metabolic pathways. The change in metabolism affects directly or indirectly the electrochemistry of various body fluids such as saliva, urine and tears. According to the recent report by Indian Diabetic Federation, 382 million people were found to be diabetic.

In this project, the suitable wavelength of light for BGM is used by determining the transmittance and absorbance of various wavelengths when passed through water and the human finger. On the basis of this suitable wavelength, a simple, non-invasive, cost-effective blood glucose level detection technique and device based on the variations in the refractive index of red light is presented. Identification of glucose concentration increases the refractive index which consequently steps up the output voltage at the photo-sensor. The variations in output voltage are converted into equivalent glucose concentrations level.

**Embedded system implementation**

**Introduction:**

An embedded system is one kind of a computer system mainly designed to perform several tasks like to access, process, and store and also control the data in various electronics-based systems. Embedded systems are a combination of hardware and software where software is usually known as firmware that is embedded into the hardware. One of its most important characteristics of these systems is, it gives the o/p within the time limits. Embedded systems support to make the work more perfect and convenient. So, we frequently use embedded systems in simple and complex devices too. The applications of embedded systems mainly involve in our real life for several devices like microwave, calculators, TV remote control, home security and neighborhood traffic control systems, etc.

User interface

Embedded system

Hardware

Software

Inputs

Output

Link to other systems

Fig: Overview of embedded system

**Embedded system:**

Embedded system includes mainly two sections, they are

1. Hardware

2. Software

Serial communication ports

Application specific circuits

Power supply and oscillator circuits

Timers

Processor

Interrupt controller

Parallel ports

Memory

Output devices interfacing

Input devices interfacing

and driver circuits

Block diagram of embedded system

**Embedded System Hardware:**

As with any electronic system, an embedded system requires a hardware platform on which it performs the operation. Embedded system hardware is built with a microprocessor or microcontroller. The embedded system hardware has elements like input output (I/O) interfaces, user interface, memory and the display. Usually, an embedded system consists of:

* Power Supply
* Processor
* Memory
* Timers
* Serial communication ports
* Output/Output circuits
* System application specific circuits

Embedded systems use different processors for its desired operation. Some of the processors used are

1. Microprocessor

2. Microcontroller

3. Digital signal processor

**Microprocessor vs. Microcontroller**

**Microprocessor**

* **CPU** on a chip.
* We can attach required amount of ROM, RAM and I/O ports.
* Expensive due to external peripherals.
* Large in size
* general-purpose

**Microcontroller**

* **Computer** on a chip
* fixed amount of on-chip ROM, RAM, I/O ports
* Low cost.
* Compact in size.
* Specific –purpose

**Embedded System Software:**

The embedded system software is written to perform a specific function. It is typically written in a high level format and then compiled down to provide code that can be lodged within a non-volatile memory within the hardware. An embedded system software is designed to keep in view of the three limits:

* Availability of system memory
* Availability of processor’s speed
* When the system runs continuously, there is a need to limit power dissipation for events like stop, run and wake up.

**Bringing software and hardware together for embedded system:**

To make software to work with embedded systems we need to bring software and hardware together .for this purpose we need to burn our source code into microprocessor or microcontroller which is a hardware component and which takes care of all operations to be done by embedded system according to our code.

Generally we write source codes for embedded systems in assembly language, but the processors run only executable files. The process of converting the source code representation of your embedded software into an executable binary image involves three distinct steps:

1. Each of the source files must be compiled or assembled into an object file.
2. All of the object files that result from the first step must be linked together to produce a single object file, called the re-locatable program.
3. Physical memory addresses must be assigned to the relative offsets within the re-locatable program in a process called relocation.

The result of the final step is a file containing an executable binary image that is ready to run on the embedded system.

Source code

Linker

Locator

Assembler

Executable file

Processor

Flow of burning source code to processor

**Applications:**

Embedded systems have different applications. A few select [applications of embedded systems](https://www.elprocus.com/embedded-systems-real-time-applications/) are smart cards, telecommunications, satellites, missiles, digital consumer electronics, computer networking, etc.

[Embedded Systems in Automobiles](http://www.edgefx.in/importance-of-embedded-systems-in-automobiles-with-applications/)

* Motor Control System
* Engine or Body Safety
* [Robotics](http://www.edgefx.in/top-list-robotics-projects-for-engineering-beginners/) in Assembly Line
* Mobile and E-Com Access

Embedded systems in Telecommunications

* Mobile computing
* Networking
* [Wireless Communications](http://www.edgefx.in/multiple-input-and-multiple-output-mimo-wireless-communications/)

Embedded Systems in Smart Cards

* Banking
* Telephone
* [Security Systems](http://www.edgefx.in/microcontroller-based-projects-on-car-security-systems-using-gsm/)

**Implementation flow:**

**Stage 1:**

Considering the problems of existing methods and giving solution to that problem by considering the basic requirements for our proposed system

**Stage 2:**

Considering the hardware requirement for the proposed system

For this we need to select the below components:

1. Microcontroller

2. Inputs for the proposed system (ex: sensors, drivers etc..,)

3. Outputs (ex: relays, loads)

**Stage 3:**

After considering hardware requirements, now we need to check out the software requirements. Based on the microcontroller we select there exists different software for coding, compiling, debugging. we need to write source code for that proposed system based on our requirements and compile, debug the code in that software .

After completing all the requirements of software and hardware we need to bring both together to work our system. For this we need to burn our source code into microcontroller, after burning our source code to microcontroller then connect all input and output modules as per our requirement.

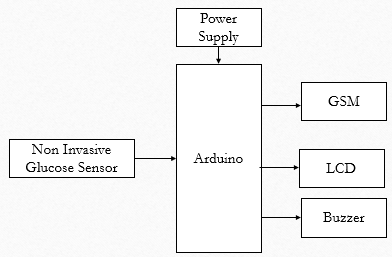
**EXISTING SYSTEM**

Glucometer is working on the principle of electrochemical detection. The major drawback associated with this kind of disease is blood dependency, which makes it an invasive approach and also increases the risk of infection for the patient.

**PROPOSED SYSTEM**

In the proposed system in order to find the glucose level we are using Red Led and a photo transistor. Based on light falling on the photo transistor from the LED. The glucose level is monitored and is uploaded to cloud server using GSM.

**Block Diagram:**



**HARDWARE REQUIRMENTS**

**ARDUINO**

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

This is what the Arduino board looks like.



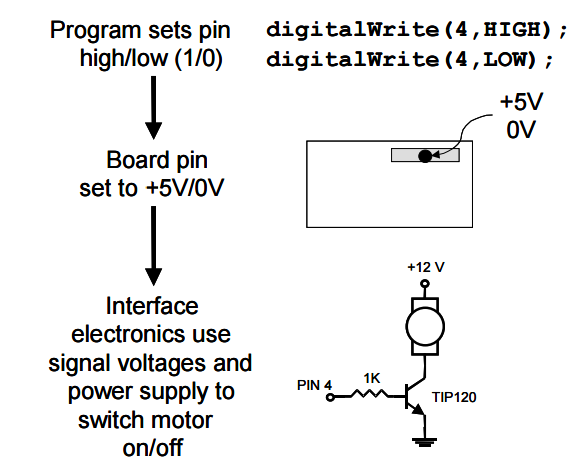
The Arduino programming language is a simplified version of C/C++. If you know C, programming the Arduino will be familiar. If you do not know C, no need to worry as only a few commands are needed to perform useful functions.

**Arduino Hardware**

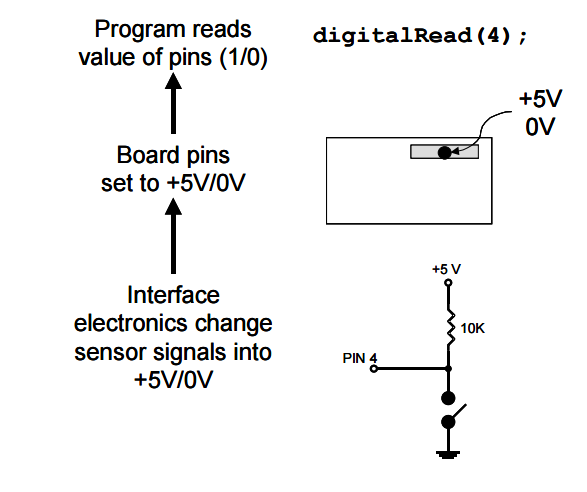
The power of the Arduino is not its ability to crunch code, but rather its ability to interact with the outside world through its input-output (I/O) pins. The Arduino has 14 digital I/O pins labeled 0 to 13 that can be used to turn motors and lights on and off and read the state of switches.

Each digital pin can sink or source about 40 mA of current. This is more than adequate for interfacing to most devices, but does mean that interface circuits are needed to control devices other than simple LED's. In other words, you cannot run a motor directly using the current available from an Arduino pin, but rather must have the pin drive an interface circuit that in turn drives the motor. A later section of this document shows how to interface to a small motor.

To interact with the outside world, the program sets digital pins to a high or low value using C code instructions, which corresponds to +5 V or 0 V at the pin. The pin is connected to external interface electronics and then to the device being switched on and off. The sequence of events is shown in this figure.



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



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

**Atmega328p features:**

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

– 131 Powerful Instructions

– Most Single Clock Cycle Execution

– 32 x 8 General Purpose Working Registers

– Fully Static Operation

– Up to 20 MIPS Throughput at 20 MHz

– On-chip 2-cycle Multiplier

* High Endurance Non-volatile Memory Segments

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

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

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

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

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

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

– Programming Lock for Software Security

* Peripheral Features

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

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

– Real Time Counter with Separate Oscillator

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

– Programmable Serial USART

– Master/Slave SPI Serial Interface

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

– Programmable Watchdog Timer with Separate On-chip Oscillator

– On-chip Analog Comparator

– Interrupt and Wake-up on Pin Change

* Special Microcontroller Features

– Power-on Reset and Programmable Brown-out Detection

– Internal Calibrated Oscillator

– External and Internal Interrupt Sources

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

* I/O and Packages

– 23 Programmable I/O Lines

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

* Operating Voltage:

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

* Temperature Range:

– -40°C to 85°C

* Speed Grade:

– 0 - 20 MHz @ 1.8 - 5.5V

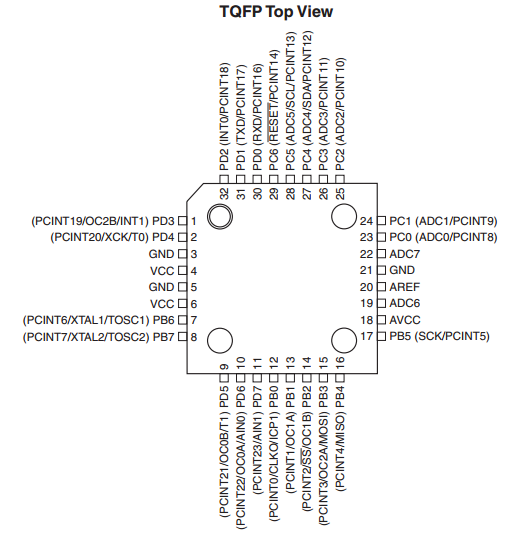
* Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:

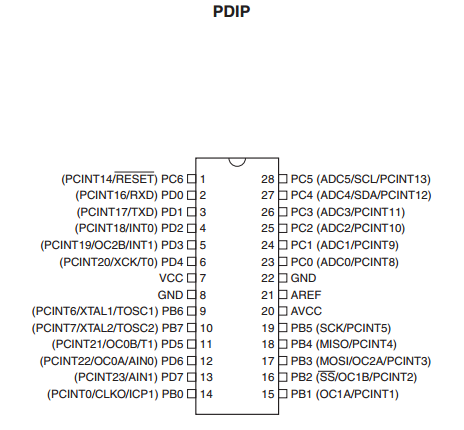
– Active Mode: 0.2 mA

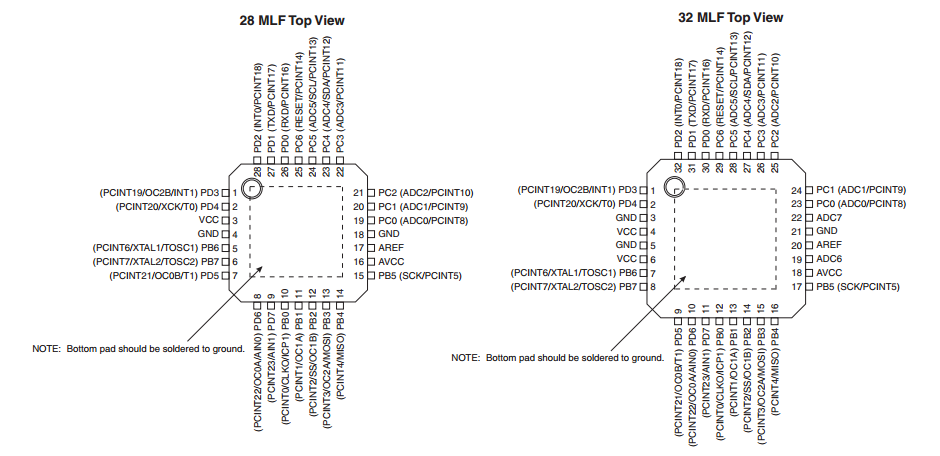
– Power-down Mode: 0.1 µA

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

**PIN CONFIGURATIONS**







**Pin Descriptions**

**VCC:** Digital supply voltage.

**GND:** Ground.

**Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2:** Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running. Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit. Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier. If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set. The various special features of Port B are elaborated in ”Alternate Functions of Port B” on page 76 and ”System Clock and Clock Options” on page 26.

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

**PC6/RESET:** If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in Table 28-3 on page 308. Shorter pulses are not guaranteed to generate a Reset. The various special features of Port C are elaborated in ”Alternate Functions of Port C” on page 79.

**Port D (PD7:0):** Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running. The various special features of Port D are elaborated in “Alternate Functions of Port D” on page 82.

**AVCC:** AVCC is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter. Note that PC6..4 use digital supply voltage, VCC.

**AREF:** AREF is the analog reference pin for the A/D Converter

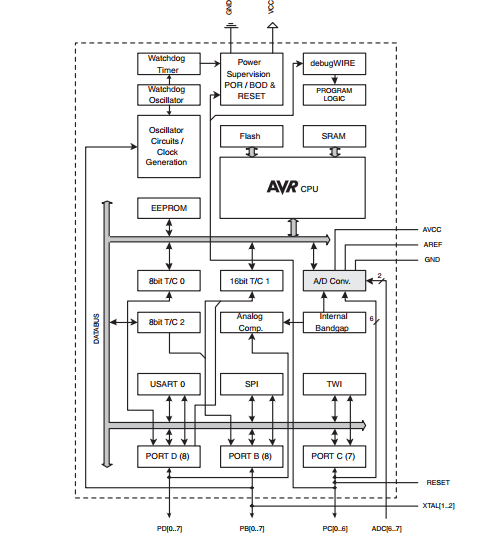
**ADC7:6 (TQFP and QFN/MLF Package Only):** In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

**OVERVIEW**

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

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

**BLOCK DIAGRAM**

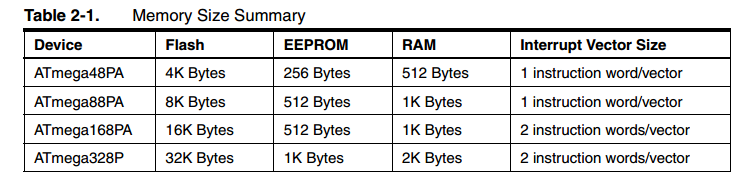
****

The ATmega48PA/88PA/168PA/328P provides the following features: 4/8/16/32K 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.

**Comparison Between ATmega48PA, ATmega88PA, ATmega168PA and ATmega328P**

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

****

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.

**POWER:**

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

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

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

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

**• GND.** Ground pins.

**Memory:**

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

**Input and Output**

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

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

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

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

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

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

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

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

There are a couple of other pins on the board:

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

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

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

**Communication**

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

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

**Programming**

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the Tools > Board menu (according to the microcontroller on your board). For details, see the reference and tutorials. The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files). You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details. The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

• On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.

• On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

**Automatic (Software) Reset**

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment.

This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following halfsecond or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened.

If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data. The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

**USB Overcurrent Protection**

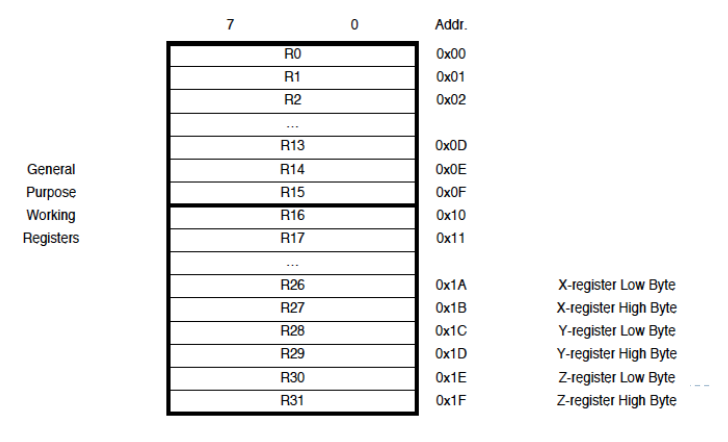
The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break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

**Physical Characteristics**

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

**Register File**

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

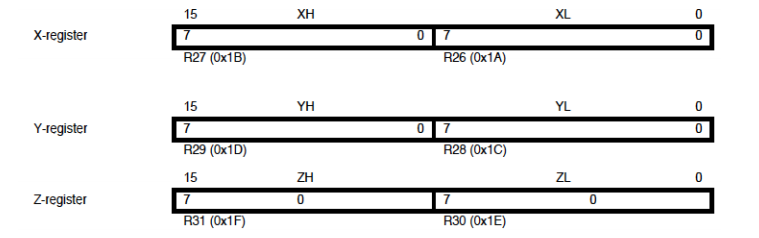
****

**Special Addressing Registers**

* X, Y and Z registers

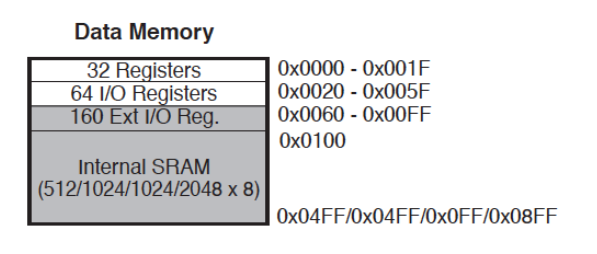
16-bit registers made using registers 26 – 31

* Support indirect addressing

****

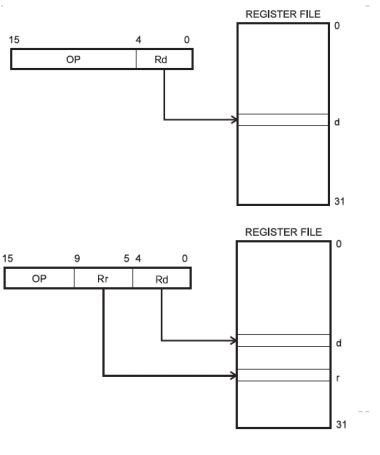
**AVR Memory**

* Program memory – Flash
* Data memory – SRAM

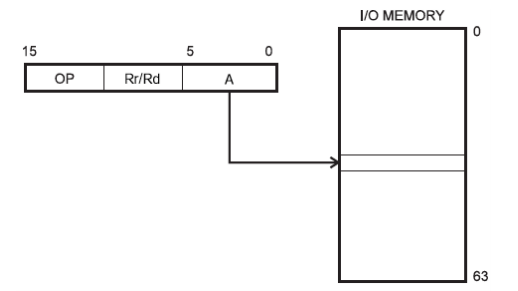


**Addressing Modes**

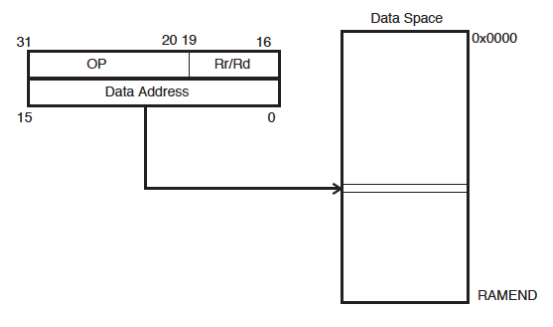
* Direct register addressing

****

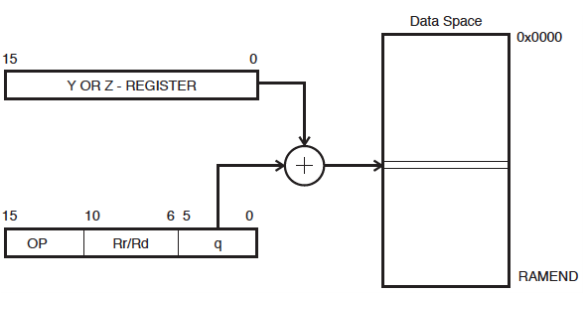
* Direct I/O addressing

****

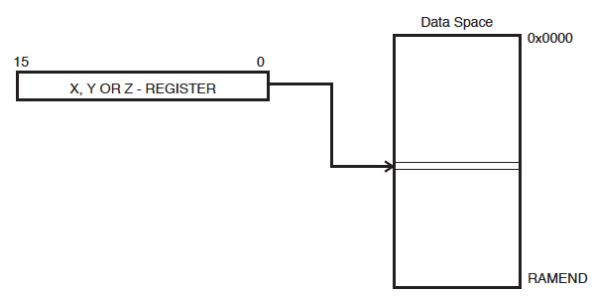
Direct data memory addressing

****

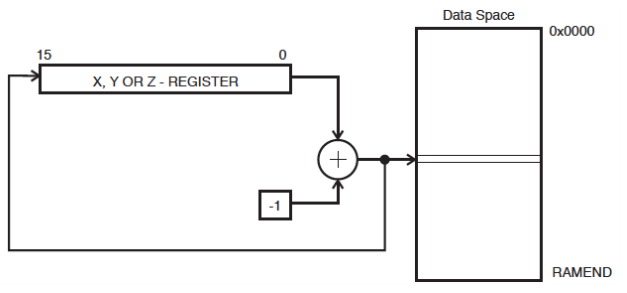
Direct data memory with displacement addressing

****

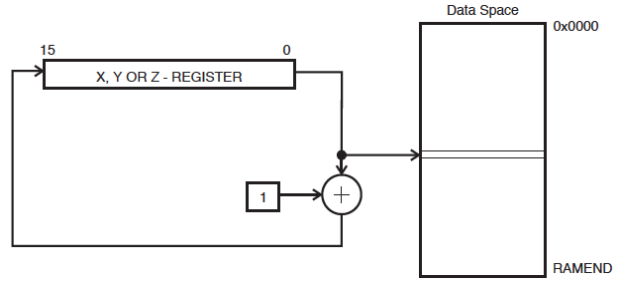
Indirect data memory addressing

****

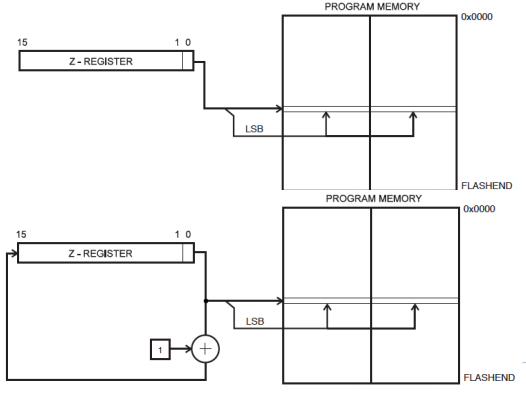
Indirect data memory addressing with pre-decrement

****

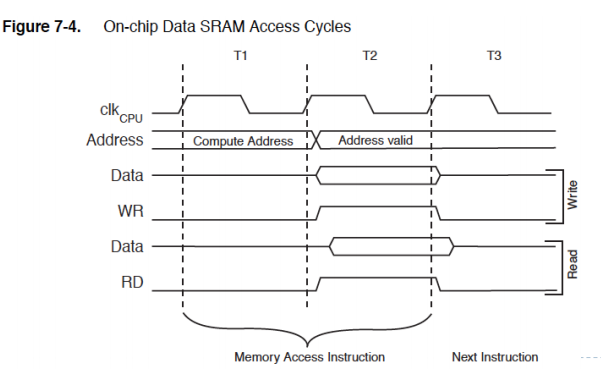
Indirect data memory addressing with post-increment

****

Program memory addressing (constant data)

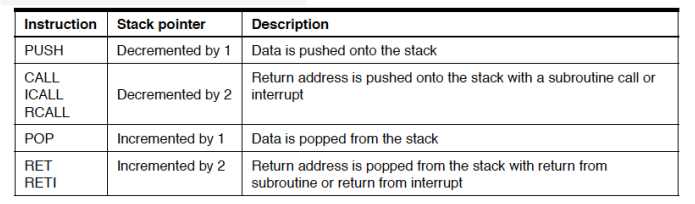
****

SRAM Read/Write Timing

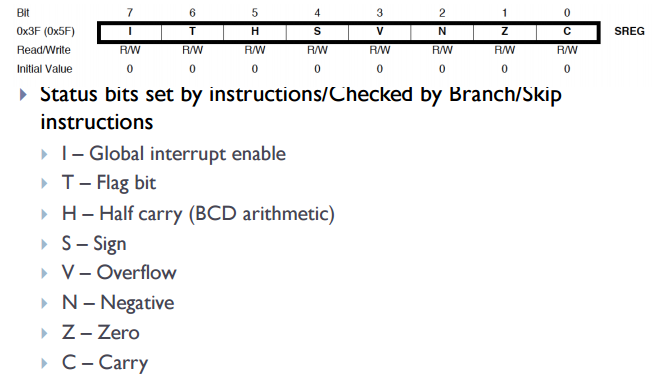
****

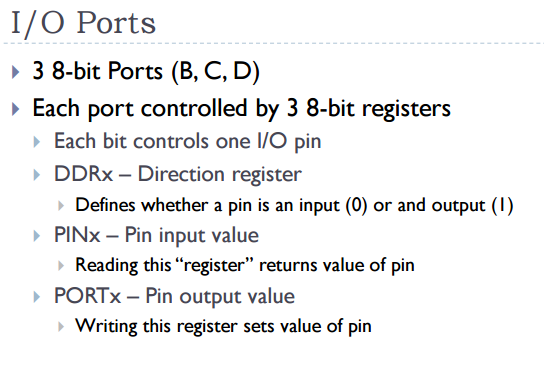
**Stack Pointer Register**

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

****

**Program Status Register (PSR)**

****

****

**LCD:**

LCD (Liquid Crystal Display) is the innovation utilized in scratch pad shows and other littler PCs. Like innovation for light-producing diode (LED) and gas-plasma, LCDs permit presentations to be a lot more slender than innovation for cathode beam tube (CRT). LCDs expend considerably less power than LED shows and gas shows since they work as opposed to emanating it on the guideline of blocking light.

A LCD is either made with a uninvolved lattice or a showcase network for dynamic framework show. Likewise alluded to as a meager film transistor (TFT) show is the dynamic framework LCD. The uninvolved LCD lattice has a matrix of conductors at every crossing point of the network with pixels. Two conductors on the lattice send a current to control the light for any pixel. A functioning framework has a transistor situated at every pixel crossing point, requiring less current to control the luminance of a pixel.

Some aloof network LCD's have double filtering, which implies they examine the matrix twice with current in the meantime as the first innovation took one sweep. Dynamic lattice, be that as it may, is as yet a higher innovation.

A 16x2 LCD show is an essential module that is generally utilized in various gadgets and circuits. These modules more than seven sections and other multi fragment LEDs are liked. The reasons being: LCDs are affordable; effectively programmable; have no restriction of showing exceptional and even custom characters (not at all like in seven fragments), movements, etc.

A 16x2 LCD implies 16 characters can be shown per line and 2 such lines exist. Each character is shown in a lattice of 5x7 pixels in this LCD. There are two registers in this LCD, in particular Command and Data.

The directions given to the LCD are put away by the order register. An order is a direction given to LCD to play out a predefined assignment, for example, introducing it, clearing its screen, setting the situation of the cursor, controlling presentation, and so forth. The information register will store the information that will be shown on the LCD. The information is the character's ASCII incentive to show on the LCD.

**Data/Signals/Execution of LCD**

Now that was all about the signals and the hardware. Let us come to data, signals and execution.

Two types of signals are accepted by LCD, one is data and one is control. The LCD module recognizes these signals from the RS pin status. By pulling the R / W pin high, data can now also be read from the LCD display. Once the E pin has been pulsed, the LCD display reads and executes data at the falling edge of the pulse, the same for the transmission case.

It takes 39-43μS for the LCD display to place a character or execute a command. It takes 1.53ms to 1.64ms except for clearing display and searching for cursor to the home position.

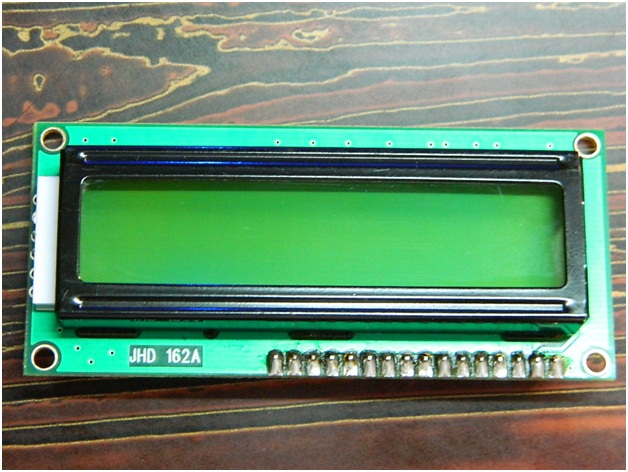
Any attempt to send data before this interval may result in failure in some devices to read data or execute the current data. Some devices compensate for the speed by storing some temporary registers with incoming data.

There are two RAMs for LCD displays, namely DDRAM and CGRAM. DDRAM registers the position in which the character would be displayed in the ASCII chart. Each DDRAM byte represents every single position on the display of the LCD.

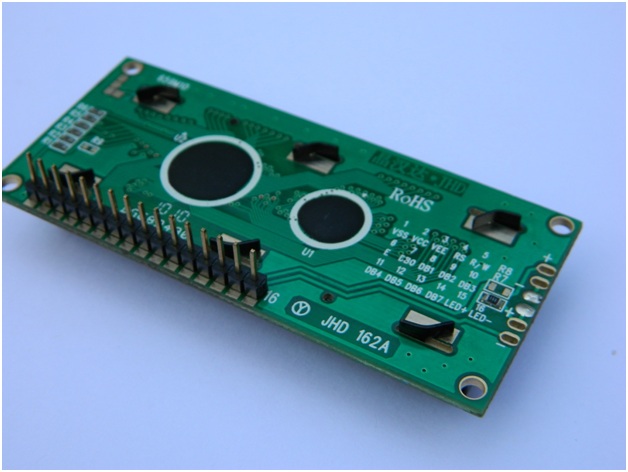
The DDRAM information is read by the LCD controller and displayed on the LCD screen. CGRAM enables users to define their personalized characters. Address space is reserved for users for the first 16 ASCII characters.

Users can easily display their custom characters on the LCD screen after CGRAM has been set up to display characters.

**Images of LCD Display:-**

[](http://www.circuitstoday.com/wp-content/uploads/2012/02/LCD-Display-Front-Side.jpg)

**LCD – Front View**

[](http://www.circuitstoday.com/wp-content/uploads/2012/02/lcd-display-back-side.jpg)

**LCD – Back View**

**Pin Diagram:**



**Pin Description:**

|  |  |  |
| --- | --- | --- |
| Pin No | Function | Name |
| 1 | Ground (0V) | Ground |
| 2 | Supply voltage; 5V (4.7V – 5.3V) | Vcc |
| 3 | Contrast adjustment; through a variable resistor | VEE |
| 4 | Selects command register when low; and data register when high | Register Select |
| 5 | Low to write to the register; High to read from the register | Read/write |
| 6 | Sends data to data pins when a high to low pulse is given | Enable |
| 7 | 8-bit data pins | DB0 |
| 8 | DB1 |
| 9 | DB2 |
| 10 | DB3 |
| 11 | DB4 |
| 12 | DB5 |
| 13 | DB6 |
| 14 | DB7 |
| 15 | Backlight VCC (5V) | Led+ |
| 16 | Backlight Ground (0V) | Led- |

**RS (Register select)**

A 16X2 LCD has two order and information registers. The determination of the register is utilized to change starting with one register then onto the next. RS=0 for the register of directions, while RS=1 for the register of information.

**Command Register**

The guidelines given to the LCD are put away by the direction register. An order is a direction given to LCD to play out a predefined assignment, for example, instating it, clearing its screen, setting the situation of the cursor, controlling showcase, and so on. Order preparing happens in the direction register.

**Data Register:**

The information register will store the information that will be shown on the LCD. The information is the character's ASCII incentive to show on the LCD. It goes to the information register and is prepared there when we send information to the LCD. While choosing RS=1, the information register.

Read and Write Mode of LCD:

As stated, the LCD itself comprises of an interface IC. This interface IC can be perused or composed by the MCU. A large portion of the occasions we're simply going to keep in touch with the IC since perusing will make it increasingly perplexing and situations like that are exceptionally uncommon.Information such as cursor position, status completion interrupts, etc. can be read if necessary.

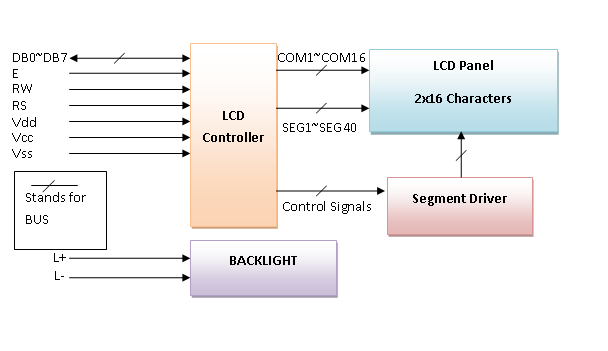
**LCD Commands:**

There are some preset commands in the LCD that we need to send to the LCD via some microcontroller. The following are some important command instructions:

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Hex Code** | **Command to LCD instruction Register** |
| 1 | 01 | Clear display screen |
| 2 | 02 | Return home |
| 3 | 04 | Decrement cursor (shift cursor to left) |
| 4 | 06 | Increment cursor (shift cursor to right) |
| 5 | 05 | Shift display right |
| 6 | 07 | Shift display left |
| 7 | 08 | Display off, cursor off |
| 8 | 0A | Display off, cursor on |
| 9 | 0C | Display on, cursor off |
| 10 | 0E | Display on, cursor blinking |
| 11 | 0F | Display on, cursor blinking |
| 12 | 10 | Shift cursor position to left |
| 13 | 14 | Shift cursor position to right |
| 14 | 18 | Shift the entire display to the left |
| 15 | 1C | Shift the entire display to the right |
| 16 | 80 | Force cursor to beginning ( 1st line) |
| 17 | C0 | Force cursor to beginning ( 2nd line) |
| 18 | 38 | 2 lines and 5×7 matrix |

## Command codes for LCD

**Block Diagram of LCD Display:-**

**[](http://www.circuitstoday.com/wp-content/uploads/2012/02/LCD-Display-Block-Diagram.png)**

**Control and display commands**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Instruction Code** | | | | | | | | | | **Instruction Code Description** | **Execution time** |
| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
| Read Data From RAM | 1 | 1 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Read data from internal RAM | 1.53-1.64ms |
| Write data to RAM | 1 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Write data into internal RAM (DDRAM/CGRAM) | 1.53-1.64ms |
| Busy flag & Address | 0 | 1 | BF | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Busy flag (BF: 1→ LCD Busy) and contents of address counter in bits AC6-AC0. | 39 µs |
| Set DDRAM Address | 0 | 0 | 1 | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Set DDRAM address in address counter. | 39 µs |
| Set CGRAM Address | 0 | 0 | 0 | 1 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Set CGRAM Address in address counter. | 39 µs |
| Function Set | 0 | 0 | 0 | 0 | 1 | DL | N | F | X | X | Set interface data length (DL: 4bit/8bit), Numbers of display line (N: 1-line/2-line) display font type (F:0→ 5×8 dots, F:1→ 5×11 dots) | 39 µs |
| Cursor or Display Shift | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | X | X | Set cursor moving and display shift control bit, and the direction without changing DDRAM data | 39 µs |
| Display & Cursor On/Off | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | Set Display(D),Cursor(C) and cursor blink(b) on/off control | 39 µs |
| Entry Mode Set | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | SH | Assign cursor moving direction and enable shift entire display. | 0µs |
| Return Home | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | X | Set DDRAM Address to “00H” from AC and return cursor to its original position if shifted. | 43µs |
| Clear Display | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Write “20H” to DDRAM and set DDRAM Address to “00H” from AC | 43µs |

4-bit and 8-bit Mode of LCD:

The LCD can work in two striking modes, the 4-bit mode and the 8-bit mode. We send the information snack through snack in 4 bit mode, first upper chomp, by then lower snack. For those of you who don't have the foggiest idea what a goody is: a chomp is a four-piece gathering, so a byte's lower four bits (D0-D3) are the lower snack, while a byte's upper four bits (D4-D7) are the higher snack. This enables us to send 8 bit data. This connects with us to send 8 bit data. Whereas in 8 bit mode we can send the 8-bit information truly in one stroke since we utilize all the 8 information lines. You need to get it now; yes 8-bit mode is quicker and immaculate than 4-bit mode. In any case, the fundamental shortcoming is that it needs 8 microcontroller-related information lines. This will result in our MCU coming up short on I/O pins, so 4-bit mode is extensively utilized. To set these modes, no control pins are used.

**GSM**

GSM is a mobile communication modem; it is stands for global system for mobile communication (GSM). The idea of GSM was developed at Bell Laboratories in 1970.  It is widely used mobile communication system in the world. GSM is an open and digital cellular technology used for transmitting mobile voice and data services operates at the 850MHz, 900MHz, 1800MHz and 1900MHz frequency bands.

GSM system was developed as a digital system using time division multiple access (TDMA) technique for communication purpose. A GSM digitizes and reduces the data, then sends it down through a channel with two different streams of client data, each in its own particular time slot. The digital system has an ability to carry 64 kbps to 120 Mbps of data rates.

There are various cell sizes in a GSM system such as macro, micro, pico and umbrella cells. Each cell varies as per the implementation domain. There are five different cell sizes in a GSM network macro, micro, pico and umbrella cells. The coverage area of each cell varies according to the implementation environment.

### Time Division Multiple Access

TDMA technique relies on assigning different time slots to each user on the same frequency. It can easily adapt to data transmission and voice communication and can carry 64kbps to 120Mbps of data rate.

### GSM Architecture

A GSM network consists of the following components:

* **A Mobile Station:**  It is the mobile phone which consists of the transceiver, the display and the processor and is controlled by a SIM card operating over the network.
* **Base Station Subsystem:** It acts as an interface between the mobile station and the network subsystem. It consists of the Base Transceiver Station which contains the radio transceivers and handles the protocols for communication with mobiles. It also consists of the Base Station Controller which controls the Base Transceiver station and acts as a interface between the mobile station and mobile switching centre.
* **Network Subsystem:** It provides the basic network connection to the mobile stations. The basic part of the Network Subsystem is the Mobile Service Switching Centre which provides access to different networks like ISDN, PSTN etc. It also consists of the Home Location Register and the Visitor Location Register which provides the call routing and roaming capabilities of GSM. It also contains the Equipment Identity Register which maintains an account of all the mobile equipments wherein each mobile is identified by its own IMEI number. IMEI stands for International Mobile Equipment Identity.

### Features of GSM Module:

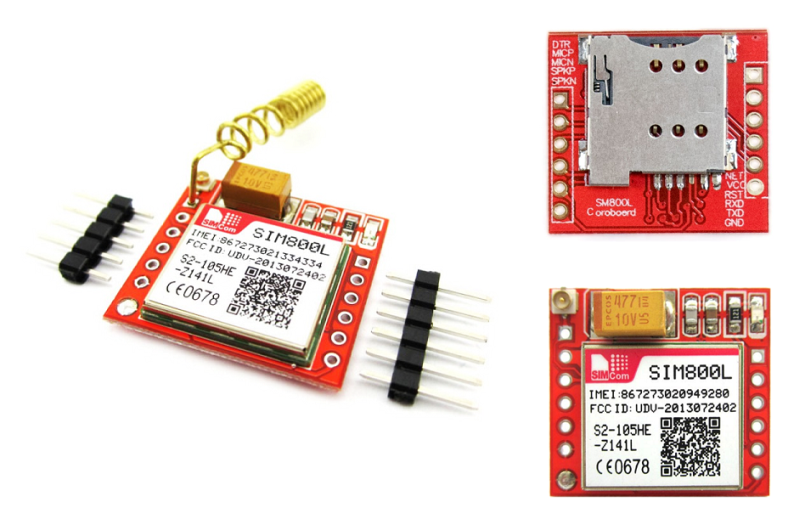
* Improved spectrum efficiency
* International roaming
* Compatibility with integrated services digital network (ISDN)
* Support for new services.
* SIM phonebook management
* Fixed dialing number (FDN)
* Real time clock with alarm management
* High-quality speech
* Uses encryption to make phone calls more secure
* Short message service (SMS)

The security strategies standardized for the GSM system make it the most secure telecommunications standard currently accessible. Although the confidentiality of a call and secrecy of the GSM subscriber is just ensured on the radio channel, this is a major step in achieving end-to- end security.

### GSM Modem

A GSM modem is a device which can be either a mobile phone or a modem device which can be used to make a computer or any other processor communicate over a network. A GSM modem requires a SIM card to be operated and operates over a network range subscribed by the network operator.  It can be connected to a computer through serial, USB or Bluetooth connection.

A GSM modem can also be a standard GSM mobile phone with the appropriate cable and software driver to connect to a serial port or USB port on your computer. GSM modem is usually preferable to a GSM mobile phone. The GSM modem has wide range of applications in transaction terminals, supply chain management, security applications, weather stations and GPRS mode remote data logging.

****

It requires a **SIM (Subscriber Identity Module)** card just like mobile phones to activate communication with the network. Also they have **IMEI** (International Mobile Equipment Identity) number similar to mobile phones for their identification. A GSM/GPRS MODEM can perform the following operations:

1.      Receive, send or delete SMS messages in a SIM.

2.      Read, add, search phonebook entries of the SIM.

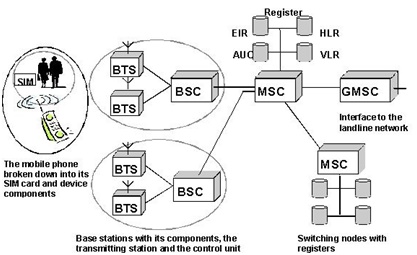
3.      Make, Receive, or reject a voice call.

The MODEM needs **AT commands**, for interacting with processor or controller, which are communicated through serial communication. These commands are sent by the controller/processor. The MODEM sends back a result after it receives a command. Different AT commands supported by the MODEM can be sent by the processor/controller/computer to interact with the **GSM and GPRS cellular network**.

### **GSM Architecture**

The GSM architecture is divided into Radio Subsystem, Network and Switching Subsystem and the Operation Subsystem. The radio sub system consists of the Mobile Station and Base Station Subsystem.

The mobile station is generally the mobile phone which consists of a transceiver, display and a processor. Each handheld or portable mobile station consists of a unique identity stored in a module known as SIM (Subscriber Identity Chip). It is a small microchip which is inserted in the mobile phone and contains the database regarding the mobile station.



**Buzzer:**

A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric. Typical uses of buzzers and beepers include alarm devices, timers and confirmation of user input such as a mouse click or keystroke. Buzzer is an integrated structure of electronic transducers, DC power supply, widely used in computers, printers, copiers, alarms, electronic toys, automotive electronic equipment, telephones, timers and other electronic products for sound devices. Active buzzer 5V Rated power can be directly connected to a continuous sound, this section dedicated sensor expansion module and the board in combination, can complete a simple circuit design, to "plug and play.



**Buzzer Pin Configuration**

|  |  |  |
| --- | --- | --- |
| **Pin Number** | **Pin Name** | **Description** |
| 1 | Positive | Identified by (+) symbol or longer terminal lead. Can be powered by 5V DC |
| 2 | Negative | Identified by short terminal lead. Typically connected to the ground of the circuit |

**Buzzer Features and Specifications**

* Rated Voltage: 6V DC
* Operating Voltage: 4-8V DC
* Rated current: <30mA
* Sound Type: Continuous Beep
* Resonant Frequency: ~2300 Hz
* Small and neat sealed package
* Breadboard and Perf board friendly

**How to use a Buzzer**

A **buzzer**is a small yet efficient component to add sound features to our project/system. It is very small and compact 2-pin structure hence can be easily used on [breadboard](https://components101.com/misc/breadboard-connections-uses-guide), Perf Board and even on PCBs which makes this a widely used component in most electronic applications.

There are two types are buzzers that are commonly available. The one shown here is a simple buzzer which when powered will make a Continuous Beeeeeeppp.... sound, the other type is called a readymade buzzer which will look bulkier than this and will produce a Beep. Beep. Beep. Sound due to the internal oscillating circuit present inside it. But, the one shown here is most widely used because it can be customized with help of other circuits to fit easily in our application.

This buzzer can be used by simply powering it using a DC power supply ranging from 4V to 9V. A simple 9V battery can also be used, but it is recommended to use a regulated +5V or +6V DC supply. The buzzer is normally associated with a switching circuit to turn ON or turn OFF the buzzer at required time and require interval.

**Applications of Buzzer**

* Alarming Circuits, where the user has to be alarmed about something
* Communication equipment’s
* Automobile electronics
* Portable equipment’s, due to its compact size

**Non-Invasive Glucose Sensor:**

Non-invasive glucose monitoring refers to the measurement of [blood glucose](https://en.wikipedia.org/wiki/Blood_glucose) levels (required by people with [diabetes](https://en.wikipedia.org/wiki/Diabetes) to prevent both chronic and acute complications from the disease) without drawing blood, puncturing the skin, or causing pain or trauma.

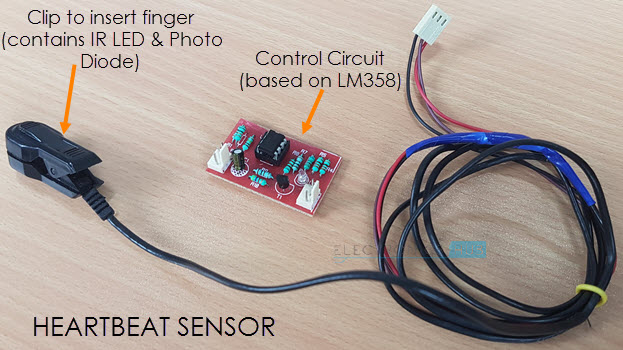
In order to measure the body temperature, we use thermometers and a sphygmomanometer to monitor the Arterial Pressure or Blood Pressure.

Heart Rate can be monitored in two ways: one way is to manually check the pulse either at wrists or neck and the other way is to use a Heartbeat Sensor.

Monitoring heart rate is very important for athletes, patients as it determines the condition of the heart (just heart rate). There are many ways to measure heart rate and the most precise one is using an Electrocardiography

But the more easy way to monitor the heart rate is to use a Heartbeat Sensor. It comes in different shapes and sizes and allows an instant way to measure the heartbeat.

Heartbeat Sensors are available in Wrist Watches (Smart Watches), Smart Phones, chest straps, etc. The heartbeat is measured in beats per minute or bpm, which indicates the number of times the heart is contracting or expanding in a minute.



#### Principle:

The principle behind the working of the Sensor is Photoplethysmograph. According to this principle, the changes in the volume of blood in an organ is measured by the changes in the intensity of the light passing through that organ.

Usually, the source of light in a heartbeat sensor would be an IR LED and the detector would be any Photo Detector like a Photo Diode, an LDR (Light Dependent Resistor) or a Photo Transistor.

With these two i.e. a light source and a detector, we can arrange them in two ways: A Transmissive Sensor and a Reflective Sensor.

In a Transmissive Sensor, the light source and the detector are place facing each other and the finger of the person must be placed in between the transmitter and receiver.

Reflective Sensor, on the other hand, has the light source and the detector adjacent to each other and the finger of the person must be placed in front of the sensor.

**SOFTWARE REQUIREMENTS**

**Arduino IDE:**

**Arduino IDE**where IDE stands for Integrated Development Environment – An official software introduced by Arduino.cc, that is mainly used for writing, compiling and uploading the code in the Arduino Device. Almost all Arduino modules are compatible with this software that is an open source and is readily available to install and start compiling the code on the go.

**Introduction to Arduino IDE:**

* Arduino IDE is an open source software that is mainly used for writing and compiling the code into the Arduino Module.
* It is an official Arduino software, making code compilation too easy that even a common person with no prior technical knowledge can get their feet wet with the learning process.
* It is easily available for operating systems like MAC, Windows, and Linux and runs on the Java Platform that comes with inbuilt functions and commands that play a vital role for debugging, editing and compiling the code in the environment.
* A range of Arduino modules available including Arduino Uno, Arduino Mega, Arduino Leonardo, [Arduino Micro](https://www.theengineeringprojects.com/2018/09/introduction-to-arduino-micro.html) and many more.
* Each of them contains a microcontroller on the board that is actually programmed and accepts the information in the form of code.
* The main code, also known as a sketch, created on the IDE platform will ultimately generate a Hex File which is then transferred and uploaded in the controller on the board.
* The IDE environment mainly contains two basic parts: Editor and Compiler where former is used for writing the required code and later is used for compiling and uploading the code into the given Arduino Module.
* This environment supports both C and C++ languages.

**How to install Arduino IDE:**

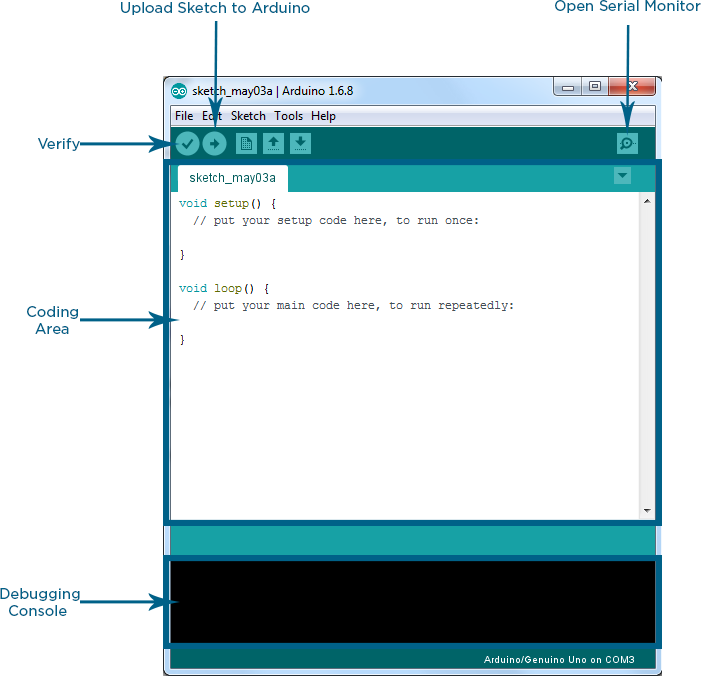
You can download the Software from [Arduino](https://www.arduino.cc/en/Main/Software) main website. As I said earlier, the software is available for common operating systems like Linux, Windows, and MAX, so make sure you are downloading the correct software version that is easily compatible with your operating system.

* If you aim to download Windows app version, make sure you have Windows 8.1 or Windows 10, as app version is not compatible with Windows 7 or older version of this operating system.

The IDE environment is mainly distributed into three sections

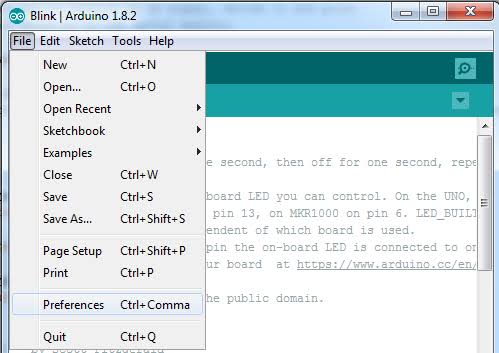
* **1. Menu Bar**
* **2. Text Editor**
* **3. Output Pane**

As you download and open the IDE software, it will appear like an image below.

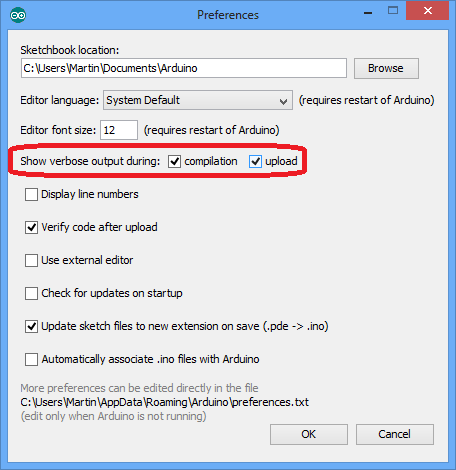


The bar appearing on the top is called **Menu Bar** that comes with five different options as follow

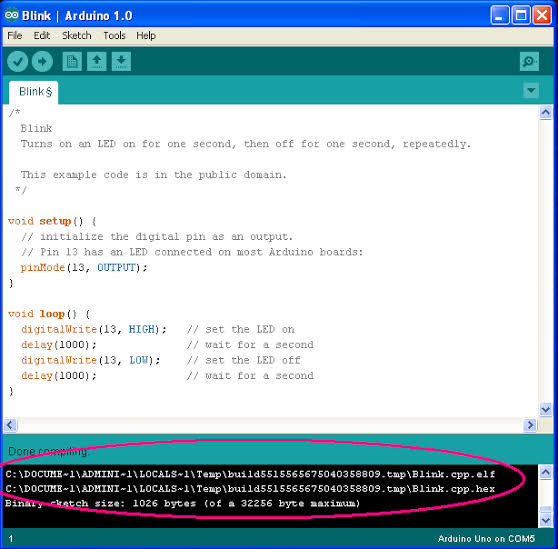
* **File** – You can open a new window for writing the code or open an existing one. Following table shows the number of further subdivisions the file option is categorized into.

****

As you go to the preference section and check the compilation section, the Output Pane will show the code compilation as you click the upload button.

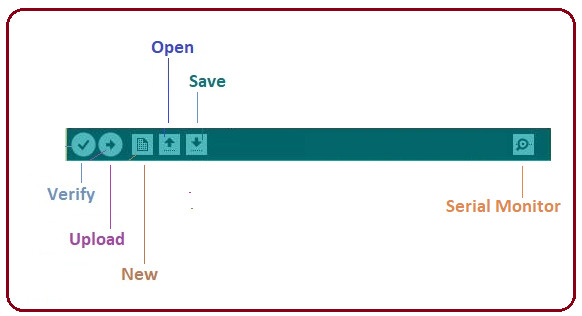


And at the end of compilation, it will show you the hex file it has generated for the recent sketch that will send to the Arduino Board for the specific task you aim to achieve.

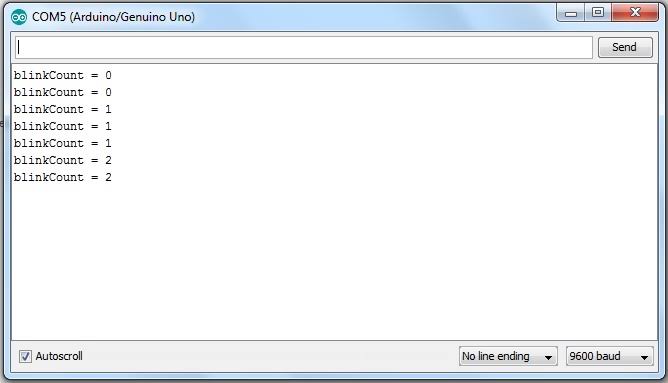
****

* **Edit** – Used for copying and pasting the code with further modification for font
* **Sketch** – For compiling and programming
* **Tools** – Mainly used for testing projects. The Programmer section in this panel is used for burning a bootloader to the new microcontroller.
* **Help** – In case you are feeling skeptical about software, complete help is available from getting started to troubleshooting.

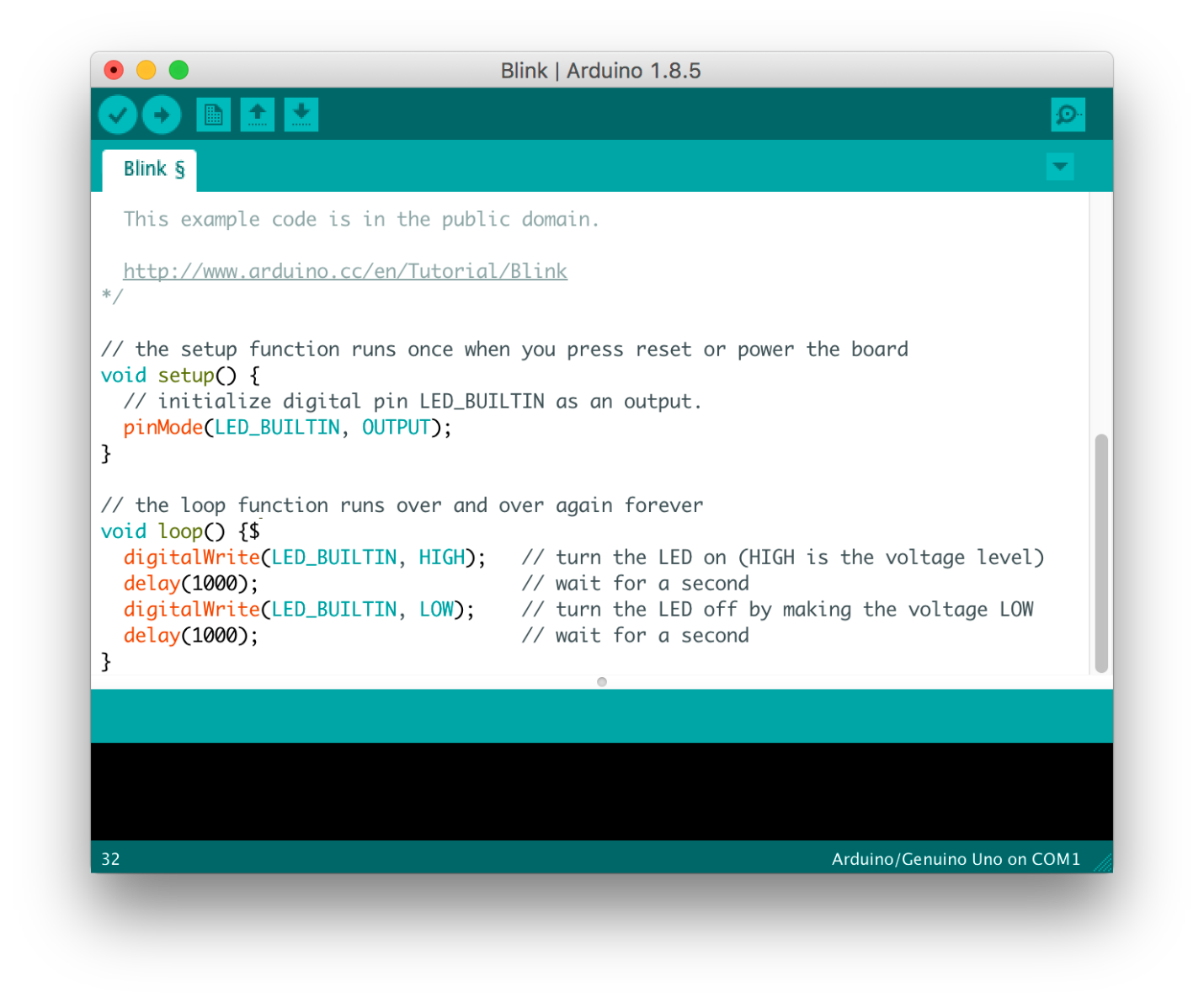
The **Six Buttons** appearing under the Menu tab are connected with the running program as follow.

****

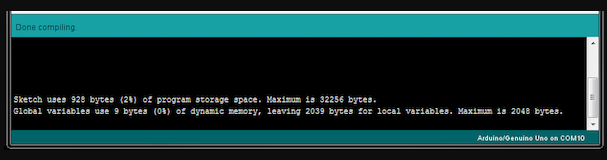
* The check mark appearing in the circular button is used to verify the code. Click this once you have written your code.
* The arrow key will upload and transfer the required code to the Arduino board.
* The dotted paper is used for creating a new file.
* The upward arrow is reserved for opening an existing Arduino project.
* The downward arrow is used to save the current running code.
* The button appearing on the top right corner is a **Serial Monitor** – A separate pop-up window that acts as an independent terminal and plays a vital role for sending and receiving the Serial Data. You can also go to the Tools panel and select Serial Monitor, or pressing Ctrl+Shift+M all at once will open it instantly. The Serial Monitor will actually help to debug the written Sketches where you can get a hold of how your program is operating. Your Arduino Module should be connected to your computer by USB cable in order to activate the Serial Monitor.
* You need to select the baud rate of the Arduino Board you are using right now. For my Arduino Uno Baud Rate is 9600, as you write the following code and click the Serial Monitor, the output will show as the image below.



The main screen below the Menu bard is known as a simple text editor used for writing the required code.



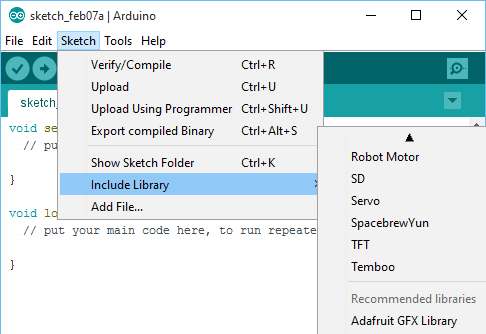
The bottom of the main screen is described as an Output Pane that mainly highlights the compilation status of the running code: the memory used by the code, and errors occurred in the program. You need to fix those errors before you intend to upload the hex file into your Arduino Module.



More or less, Arduino C language works similar to the regular C language used for any embedded system microcontroller, however, there are some dedicated libraries used for calling and executing specific functions on the board.

**Libraries:**

Libraries are very useful for adding the extra functionality into the Arduino Module. There is a list of libraries you can add by clicking the Sketch button in the menu bar and going to Include Library.



As you click the Include Library and Add the respective library it will on the top of the sketch with a #include sign. Suppose, I Include the EEPROM library, it will appear on the text editor as

#include <EEPROM.h>.

Most of the libraries are preinstalled and come with the Arduino software. However, you can also download them from the external sources.

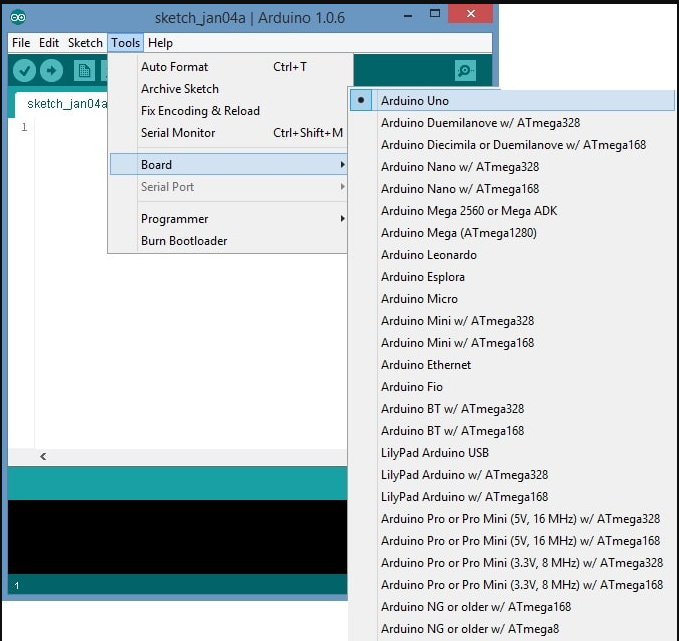
**Making pins Input and output:**

The digitalRead and [digitalWrite](https://www.theengineeringprojects.com/2018/09/how-to-use-digitalwrite-arduino-command.html" \t "_blank) commands are used for addressing and making the Arduino pins as an input and output respectively.

These commands are text sensitive i.e. you need to write them down the exact way they are given like digitalWrite starting with small “d” and write with capital “W”. Writing it down with Digitalwrite or digitalwrite won’t be calling or addressing any function.

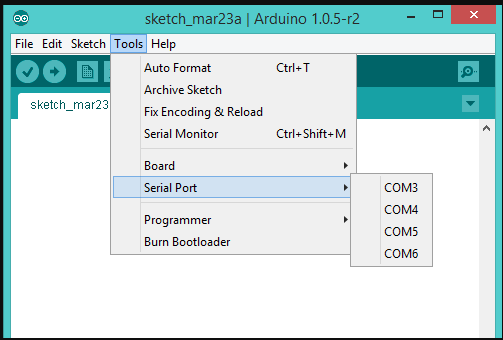
**How to select the board:**

In order to upload the sketch, you need to select the relevant board you are using and the ports for that operating system. As you click the Tools on the Menu, it will open like the figure below.



* Just go to the “Board” section and select the board you aim to work on. Similarly, COM1, COM2, COM4, COM5, COM7 or higher are reserved for the serial and USB board. You can look for the USB serial device in the ports section of the Windows Device Manager.

Following figure shows the COM4 that I have used for my project, indicating the Arduino Uno with COM4 port at the right bottom corner of the screen.



* After correct selection of both Board and Serial Port, click the verify and then upload button appearing in the upper left corner of the six button section or you can go to the Sketch section and press verify/compile and then upload.
* The sketch is written in the text editor and is then saved with the file extension .ino.

It is important to note that the recent Arduino Modules will reset automatically as you compile and press the upload button the IDE software, however, older version may require the physical reset on the board.

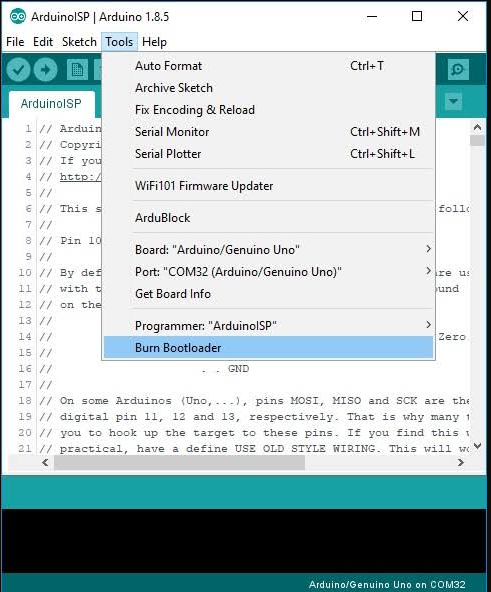
* Once you upload the code, TX and RX LEDs will blink on the board, indicating the desired program is running successfully.

**Note**: The port selection criteria mentioned above is dedicated for Windows operating system only, you can check this [Guide](https://www.arduino.cc/en/Guide/Environment) if you are using MAC or Linux.

* The amazing thing about this software is that no prior arrangement or bulk of mess is required to install this software, you will be writing your first program within 2 minutes after the installation of the IDE environment.

**BootLoader:**

As you go to the Tools section, you will find a bootloader at the end. It is very helpful to burn the code directly into the controller, setting you free from buying the external burner to burn the required code.



When you buy the new Arduino Module, the bootloader is already installed inside the controller. However, if you intend to buy a controller and put in the Arduino module, you need to burn the bootloader again inside the controller by going to the Tools section and selecting the burn bootloader.

**ADVANTAGES**

* Effective cost.
* Can avoid painful devices
* Can be safe from spreading diseases

**APPLICATIONS**

* Homes
* Hospitals

**CONCLUSION**

Invasive method of glucose measurement is painful, costly and discomfort. It also has a risk of infection and not used for continuous monitoring. In order to overcome the above disadvantages, a noninvasive method for blood glucose measurement using Sensor is proposed in this project. The glucose level in the blood which is obtained from the photodetector is displayed in both the LCD display. This portable noninvasive blood glucose monitor provides a very effective means for assisting the health care management of diabetic patients. This can be used for monitoring blood glucose level of the patients in the home as well as health care centers.

**REFERENCES**

[1] Prof.Mrs.A.A.Shinde, “Non Invasive Blood Glucose Measurement using NIR technique based on occlusion spectroscopy”, International Journal of Engineering Science and Technology (IJEST) ISSN: 0975-5462 Vol. 3 No. 12, pp. 8325-8333, December 2011.

[2] Chi-Fuk So, Kup-Sze Choi, Thomas KS Wong, Joanne WY Chung, “Recent advances in non invasive glucose monitoring”, Medical Devices: Evidence and Research. Vol.5, pp. 45-52, June 2012.

[3] JyotiYadav, Asha Rani, Vijender Singh “Near-Infrared LED based Non-invasive Blood Glucose Sensor”, International conference on Signal Processing and Integrated Network(SPIN), IEEE, pp. 591-594, Feb 2014.

[4] Christopher Dale Chua, Ian Mikhael Gonzales, Enrique Manzano, Maria Carla Manzano, “Design and Fabrication of a Non-Invasive Blood Glucometer Using Paired Photo-Emitter and Detector Near-Infrared LEDs”, DLSU Research Congress, De La Salle University, Manila, Philippines, March 2014.

[5] JyotiYadav, Asha Rani, Viajnder Singh, BhaskarMoahnMurari, “Comparative Study of Different Measurement Sites using NIR Based Non-invasive Glucose Measurement system”, Procedia Computer Science, Elsevier, Vol. 70, pp. 469 – 475, 2015.

[6] Parag Narkhede, Suraj Dhalwar, B. Karthikeyan. “NIR Based Non-Invasive Blood Glucose Measurement”, Indian Journal of Science and Technology, Vol. 9 No. 41, pp. 1-7, November 2016.