

PLAN FOR ARDUINO TRAINING

Part-1

Basics of Arduino

- Basic pin diagram
- Architecture
- Board circuit diagram

Installation of Arduino software

Communication mode

Input and Output(Programming included)

- Digital input
- Digital output
- Analog input (sensor input)

PWM (speed control)

Part-2

Controlling relays

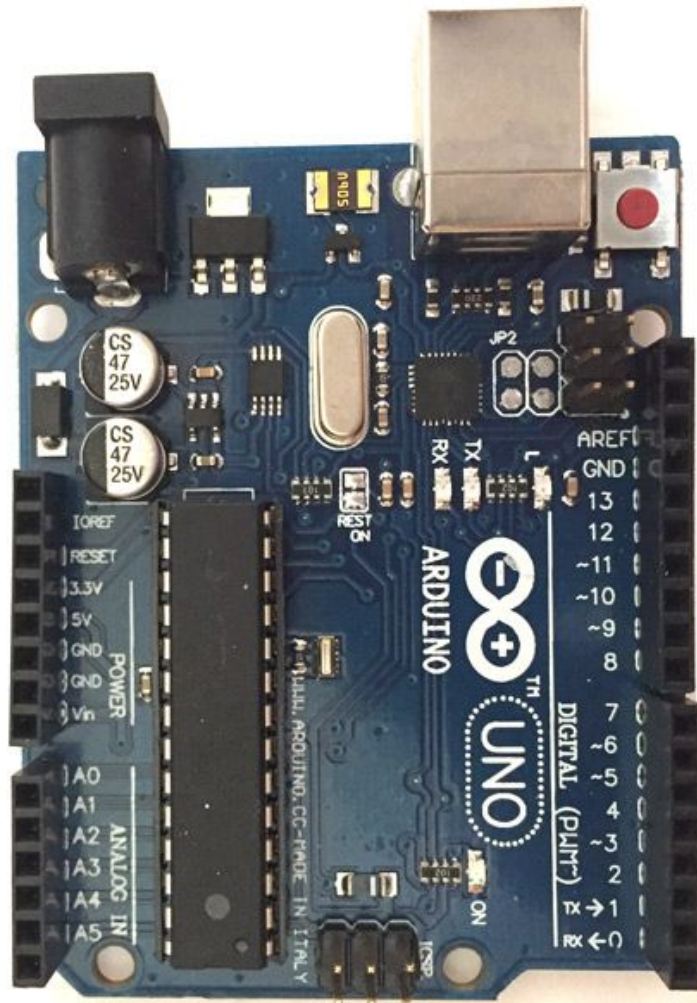
Motor basics (phase diagram)

Basics of motor driver (2phase/4phase)

Some basic ICs

- L293D
- Shift register(LED DRIVER)

ARDUINO UNO



Arduino Uno Technical Specifications

Microcontroller	ATmega328P – 8 bit AVR family microcontroller
Operating Voltage	5V
Recommended Input Voltage	7-12V
Input Voltage Limits	6-20V
Analog Input Pins	6 (A0 – A5)
Digital I/O Pins	14 (Out of which 6 provide PWM output)

DC Current on I/O Pins	40 mA
DC Current on 3.3V Pin	50 mA
Flash Memory	32 KB (0.5 KB is used for Bootloader)
SRAM	2 KB
EEPROM	1 KB
Frequency (Clock Speed)	16 MHz

SOFTWARE

We will be using Arduino IDE to write and upload the codes.

Links for Download of Software:

Window Installer-[Click Here](#)

Zip file- [click here](#)

Sketch – The first new terminology is the Arduino program called “sketch”. Software structure consist of two main functions –

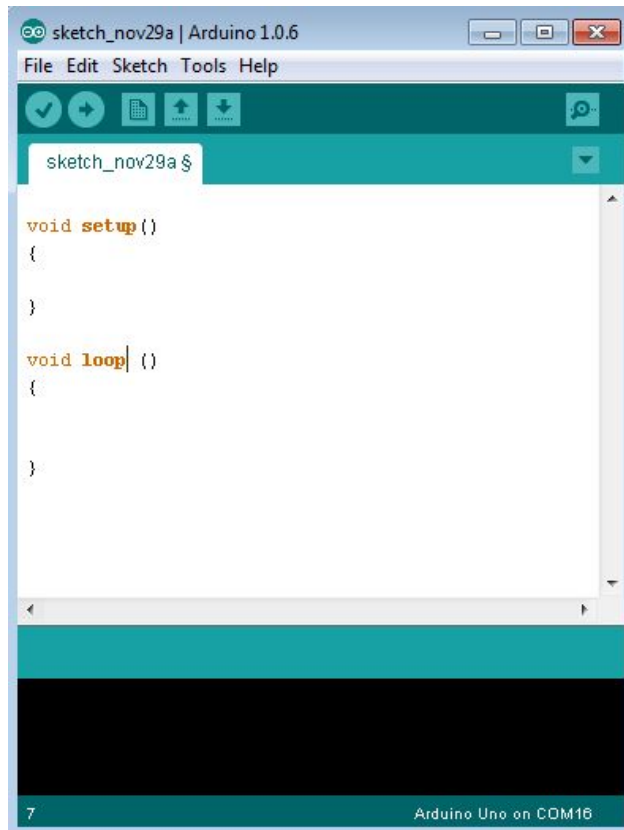
Setup() function-The setup() function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.

- INPUT – -
- OUTPUT – -
- RETURN – -

Loop() function-After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops

consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

- INPUT – -
- OUTPUT – -
- RETURN – -



FOR FURTHER HELP REGARDING SOFTWARE INSTALLATION OR PROGRAMMING [CLICK HERE](#) OR CONTACT YOUR ALLOTTED SUPER COORDINATOR

INTRODUCTION

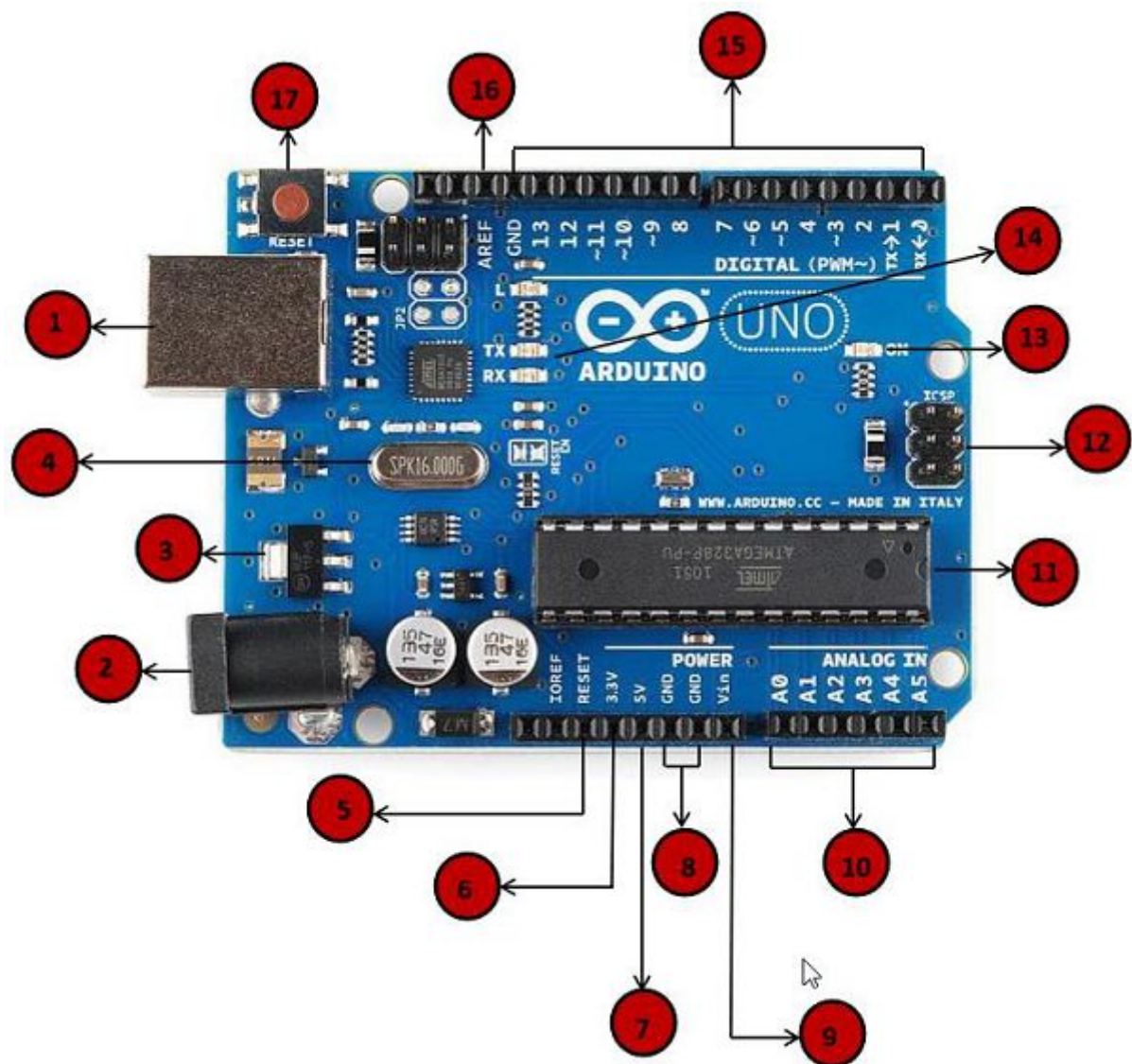
Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a

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.





In this chapter, we will learn about the different components on the Arduino board. We will study the Arduino UNO board because it is the most popular board in the Arduino board family. In addition, it is the best board to get started with electronics and coding. Some boards look a bit different from the one given below, but most Arduinos have majority of these components in common.



<p>1</p>	<p>Power USB</p> <p>Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection (1).</p>
<p>2</p>	<p>Power (Barrel Jack)</p>

	<p>Arduino boards can be powered directly from the AC mains power supply by connecting it to the Barrel Jack (2).</p>
	<p>Voltage Regulator</p> <p>The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.</p>
	<p>Crystal Oscillator</p> <p>The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.</p>
	<p>Arduino Reset</p> <p>You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).</p>

	<p>Pins (3.3, 5, GND, Vin)</p> <ul style="list-style-type: none"> • 3.3V (6) – Supply 3.3 output volt • 5V (7) – Supply 5 output volt • Most of the components used with Arduino board works fine with 3.3 volt and 5 volt. • GND (8)(Ground) – There are several GND pins on the Arduino, any of which can be used to ground your circuit. • Vin (9) – This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.
	<p>Analog pins</p> <p>The Arduino UNO board has five analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.</p>
	<p>Main microcontroller</p> <p>Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.</p>

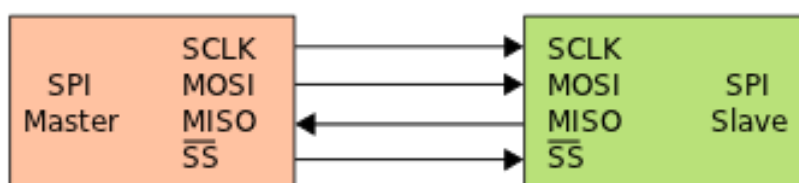
	<p>ICSP pin</p> <p>Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus.</p>
	<p>Power LED indicator</p> <p>This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.</p>
	<p>TX and RX LEDs</p> <p>On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.</p>
	<p>Digital I/O</p> <p>The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital</p>

	output pins to drive different modules like LEDs, relays, etc. The pins labeled “~” can be used to generate PWM.
16	<p>AREF</p> <p>AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.</p>

Communication Protocols

SPI

The SPI (Serial Peripheral Interface) is the protocol used by the ICSP (in-circuit serial programming) facility transmitted over the ubiquitous 6-pin (2×3 pin) header used to program AVR chips. It is useful not only for programming AVR chips, but also for other types of communications between ICs.



One thing I like about this protocol (as compared to those using the UARTs TXD & RXD lines) is that the connection does not need to be reversed. As you can see in the diagram above, MOSI connects to MOSI and MISO to MISO.

Besides using it to communicate or control other AVR chips, you can also hook other useful ICs to your Arduino as well. Examples include a temperature chip and a 4-digit 7-segment LED display driver. The Arduino IDE even has a library that

implements this protocol, making use of it a breeze (okay, maybe not that simple, but at least a lot easier). Both ATmega chips and ATtiny chips support this protocol, but ATtiny support is slightly limited – see the section on USI (below) for details.

I2C/TWI

This protocol is known as both I2C (Inter-Integrated Circuit) and as TWI (two wire interface).

It is available on both ATtiny and ATmega chips. Since it uses only two wires, so connections are simple. One line is for the clock and the other for the data. Like the SPI interface, these lines do not need reversing either. The pin names used for this protocol are SCL (clock) and SDA (data). Since it was designed for inter-IC communication, its primary value is driving other chips over a common serial bus without tying up a lot of pins. Each device has its own unique address, so you can connect up to 128 devices to your AVR chip. There are literally thousands of ICs that you can hook to your Arduino or ATtiny to greatly expand its capabilities. Examples include

- Real time clocks
- Data acquisition
- I/O pin expanders
- LED drivers
- Display drivers

and many more. The last example – display drivers is a far superior way to drive displays requiring a lot of pins that using a shift register. The prices for these chips are surprisingly affordable in many cases as well. John over at TronixStuff has an excellent tutorial on using this protocol with your Arduino that includes examples on using some of these ICs.

As with the SPI protocol, you don't have to roll your own code either. The Arduino IDE has a library that implements this protocol, using simple functions such as `read()` and `write()`. Here's an example:

```
#include "Wire.h" // enable I2C bus

Wire.beginTransmission(deviceAddress);
Wire.write(1);
Wire.write(digits[one]);
Wire.write(digits[ten]);
Wire.write(digits[hundred]);
Wire.write(digits[thousand]);
Wire.endTransmission();
```

The Arduino *Wire* library is also compatible with C++ stream i/o, so you can use standard stream functions as well.

Before struggling with shift registers and other hacks to expand your I/O pins, first look to see if there is an I2C chip that does it first.

USI

USI which stands for Universal Serial Interface, does not stand for a communication protocol, but is Atmel's implementation of serial communication on many of the ATtiny series chips. Most of the ATtinys do not support either I2C or SPI directly. Instead they provide the USI feature. This interface can be used for both two wire (I2C/TWI) and three wire (SPI) synchronous data transfer – either master or slave. Here are some specifics from the ATtiny spec sheets on how these protocols are implemented with USI and where this implementation differs.

Three Wire

Two Wire

All in all, Atmel has done a great job to support these useful protocols in such tiny and inexpensive chips using this interface.

UART/USART

UART stands for Universal Asynchronous Receiver/Transmitter. USART is a synonym for UART. Instead of being a protocol, a UART/USART is actually a piece of computer hardware that implements asynchronous serial communication. The well-known RS-232 serial protocol is one of many that operate on this hardware. You probably use this protocol regularly, as it is the one used to communicate over the serial port (via USB) connection to your computer.

Almost all ATmega series chips, but only a few of the ATtiny chips (Attiny2313/4313 mainly) support this protocol. Surprisingly, this ATtiny series only comes with a maximum of 4 megabytes of RAM, which rules it out for all but the simplest of projects.

The USART hardware is implemented over the TXD & RXD pins. Its presence also permits fuller support of the SPI protocol than does chips containing USI only (most ATtiny chips).

If you need some basic serial communication on an ATtiny, all hope is not lost. The Arduino IDE has a Software Serial library which implements this protocol in software using two digital pins. The Arduino site also has a nice tutorial on using this library.

USB

I think it is pretty safe to assume we all know what this protocol is (Universal Serial Bus), and of course we would all love to have it on all our AVR chips. Since this protocol is no doubt more expensive to implement, it is supported on only certain ATmega chips and no ATtiny ones. The ATmega chips used for various Arduinos do not have this capability, but instead rely on external chips or cables to permit basic serial communication and programming.

While some non-Arduino boards use the ATmegaXXU2/4 series boards, until recently these were not supported by the Arduino IDE. With version 1.0 now released of the IDE, it is supported. The official Arduino board is called Leonardo. Since it has not yet been released, the version 1.0 of the IDE has it commented out in the *boards.txt* file. If you want to use your own project with this chip with the Arduino IDE, simply un-comment the Leonardo board lines. The file is found in this path for your installation:

```
arduino-1.0\hardware\arduino\boards.txt
```

To re-enable this board, simply open the file in a text editor and look for lines starting with "leonardo". Remove the pound (#) sign in front of each line and save the file.

There are some other AVR chips such as ATXmega which also support USB natively, but these use 3.3 volts and are therefore not compatible with the Arduino ecosystem.

Native USB provides more than simply eliminating an extra IC. Your Arduino/AVR chip can now function as a USB device. That means instead of being limited to serial communication, the AVR chip can mimic popular USB peripherals such as keyboards and mice. The number of cool applications are endless.

For most of us, our ATmegas and ATtinys do not have built-in USB. In like manner to the Software Serial library, there is a great software implementation for USB communication as well. It is called VUSB (virtual USB), and you can even run it on the ATtiny series chips. Not as high performance as native hardware, it is more than enough for emulating a mouse or keyboard. You can even use it to turn an ATtiny into an ICSP programmer.

I/O FUNCTIONS

pinMode() Function

The pinMode() function is used to configure a specific pin to behave either as an input or an output. It is possible to enable the internal pull-up resistors with the mode INPUT_PULLUP. Additionally, the INPUT mode explicitly disables the internal pull-ups.

pinMode() Function Syntax

```
Void setup () {  
  pinMode (pin , mode);  
}
```

- pin – the number of the pin whose mode you wish to set
- mode – INPUT, OUTPUT, or INPUT_PULLUP.

digitalWrite() Function

The digitalWrite() function is used to write a HIGH or a LOW value to a digital pin. If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW. If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor.

If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim. Without explicitly setting pinMode(), digitalWrite() will have enabled the internal pull-up resistor, which acts like a large current-limiting resistor.

digitalWrite() Function Syntax

```
void loop() {  
    digitalWrite (pin ,value);  
}
```

- pin – the number of the pin whose mode you wish to set
- value – HIGH, or LOW.

analogRead() function

Arduino is able to detect whether there is a voltage applied to one of its pins and report it through the digitalRead() function. There is a difference between an on/off sensor (which detects the presence of an object) and an analog sensor, whose value continuously changes. In order to read this type of sensor, we need a different type of pin.

In the lower-right part of the Arduino board, you will see six pins marked "Analog In". These special pins not only tell whether there is a voltage applied to them, but

also its value. By using the `analogRead()` function, we can read the voltage applied to one of the pins.

This function returns a number between 0 and 1023, which represents voltages between 0 and 5 volts. For example, if there is a voltage of 2.5 V applied to pin number 0, `analogRead(0)` returns 512.

`analogRead()` function Syntax

```
analogRead(pin);
```

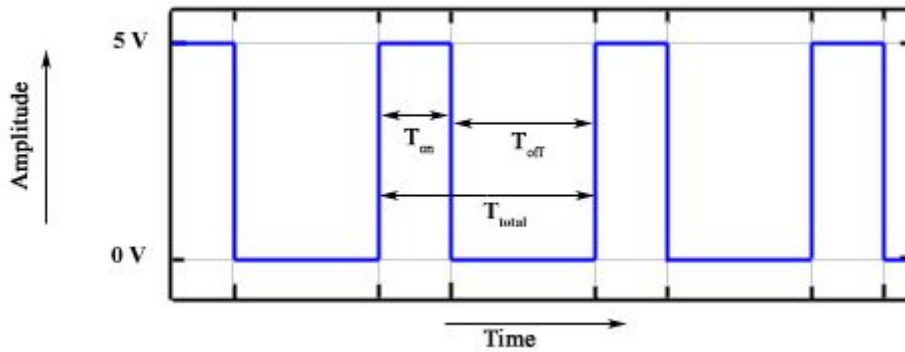
- `pin` – the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Pulse Width Modulation or PWM is a common technique used to vary the width of the pulses in a pulse-train. PWM has many applications such as controlling servos and speed controllers, limiting the effective power of motors and LEDs.

Basic Principle of PWM

Pulse width modulation is basically, a square wave with a varying high and low time. A basic PWM signal is shown in the following figure.

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width.



There are various terms associated with PWM –

- On-Time – Duration of time signal is high.
- Off-Time – Duration of time signal is low.
- Period – It is represented as the sum of on-time and off-time of PWM signal.
- Duty Cycle – It is represented as the percentage of time signal that remains on during the period of the PWM signal.

Period

As shown in the figure, T_{on} denotes the on-time and T_{off} denotes the off-time of signal. Period is the sum of both on and off times and is calculated as shown in the following equation –

$$T_{total} = T_{on} + T_{off}$$

Duty Cycle

Duty cycle is calculated as the on-time of the period of time. Using the period calculated above, duty cycle is calculated as –

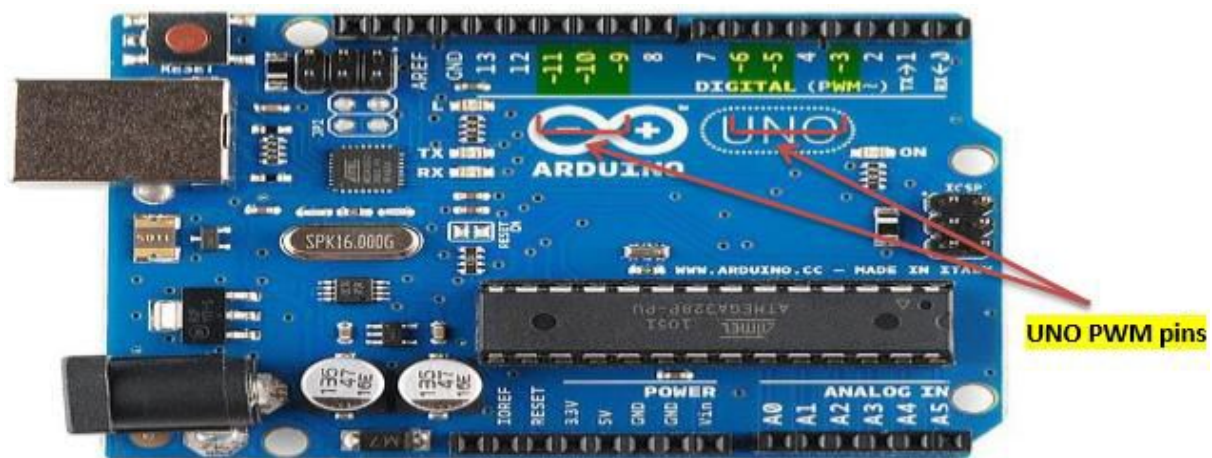
$$D = \frac{T_{on}}{T_{on} + T_{off}} = \frac{T_{on}}{T_{total}}$$

analogWrite() Function

The `analogWrite()` function writes an analog value (PWM wave) to a pin. It can be used to light a LED at varying brightness or drive a motor at various speeds. After a

call of the `analogWrite()` function, the pin will generate a steady square wave of the specified duty cycle until the next call to `analogWrite()` or a call to `digitalRead()` or `digitalWrite()` on the same pin. The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Leonardo also run at 980 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support `analogWrite()` on pins 9, 10, and 11.



The Arduino Due supports `analogWrite()` on pins 2 through 13, and pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.

You do not need to call `pinMode()` to set the pin as an output before calling `analogWrite()`.

A call to [`analogWrite\(\)`](#) is on a scale of 0 - 255, such that `analogWrite(255)` requests a 100% duty cycle (always on), and `analogWrite(127)` is a 50% duty cycle (on half the time)

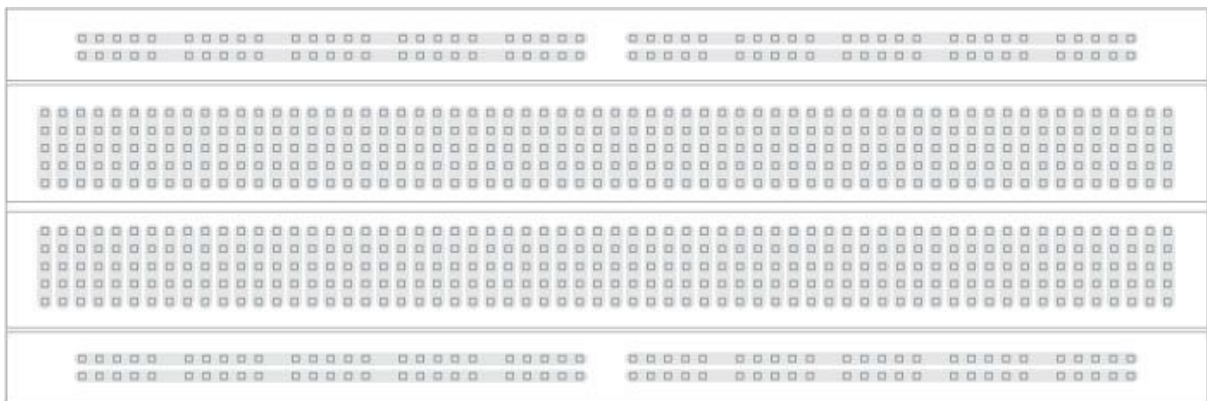
`analogWrite()` Function Syntax

```
analogWrite ( pin , value ) ;
```

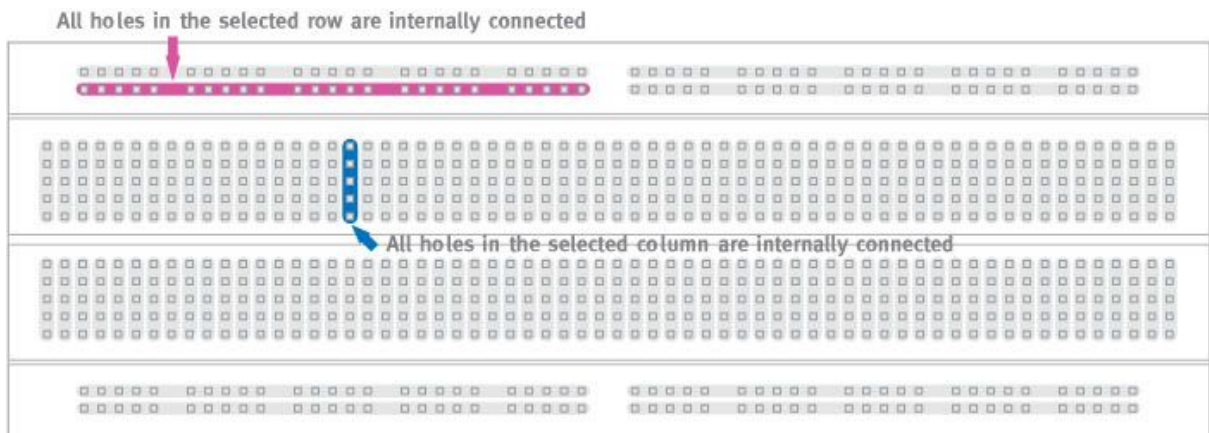
value – the duty cycle: between 0 (always off) and 255 (always on).

What is a breadboard?

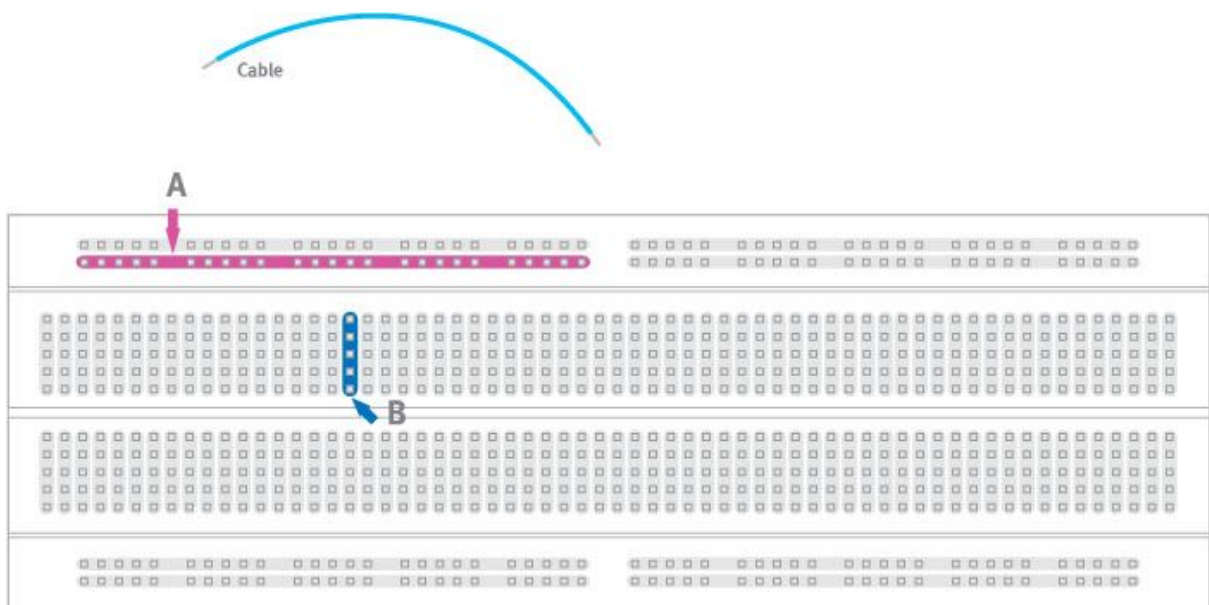
A breadboard is a solderless device for temporary prototype with electronics and test circuit designs. Most electronic components in electronic circuits can be interconnected by inserting their leads or terminals into the holes and then making connections through wires where appropriate. The breadboard has strips of metal underneath the board and connect the holes on the top of the board. The metal strips are laid out as shown below. Note that the top and bottom rows of holes are connected horizontally and split in the middle while the remaining holes are connected vertically.



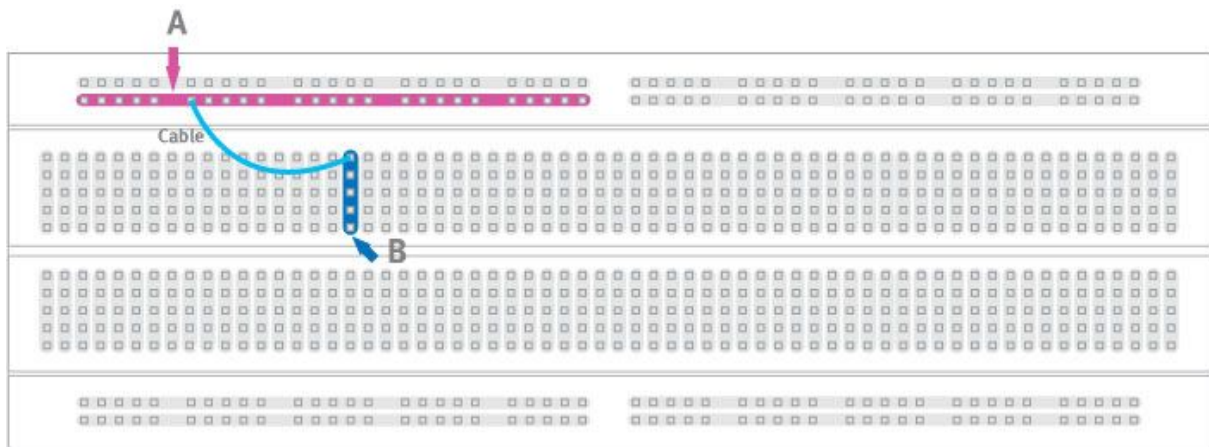
Note how all holes in the selected row are connected together, so the holes in the selected column. The set of connected holes can be called a node:



To interconnect the selected row (node A) and column (node B) a cable going from any hole in the row to any hole in the column is needed:



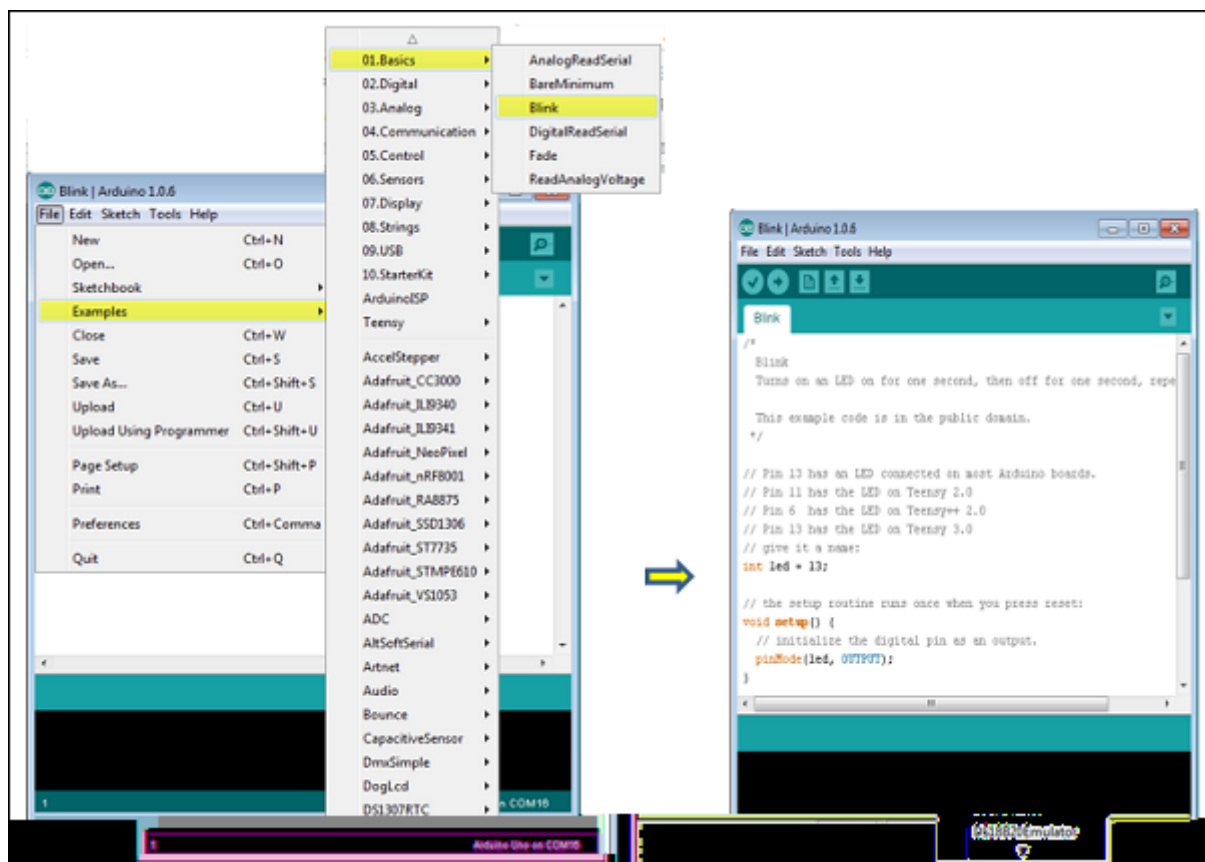
Now the selected column (node B) and row (node A) are interconnected:



ASSIGNMENTS

TASK 1.

Arduino IDE has few built-in examples for beginners:



Read all the codes from the following examples sets from Arduino IDE:

01.BASICS

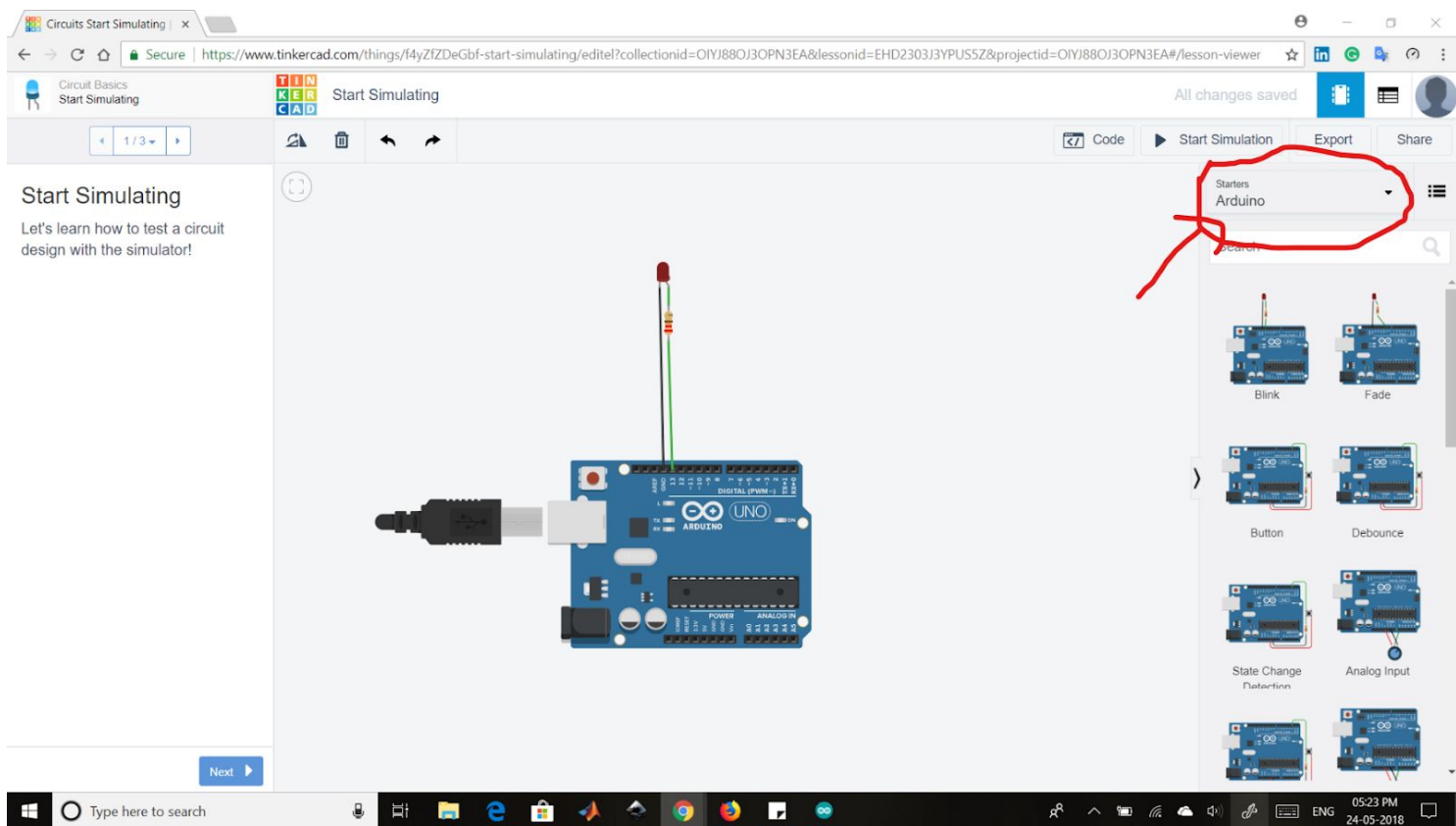
02.DIGITAL

03.ANALOG

TASK 2

We will be using Tinkercad Circuits for building and simulating the circuits.

Go to the website, create an autodesk account, open the Autodesk Circuits workspace, set the Search bar as shown below and simulate all the above codes:



Click below to visit the website

<https://www.tinkercad.com/circuits>

For further help watch the following videos from
<https://www.youtube.com/playlist?list=PLA567CE235D39FA84>

TUTORIAL 1,2,3,4,6,7,8

GO ENVISAGE!

FOR ALL THE MARVEL FANS

Arduino will return

To be Continued.....