Lab-05 : ADC value on LCD

1. **Objectives: After this lab, the learner will be able to:**
   - Interface LCD module with Arduino board
   - Interface ADC
2. **Facilities**
   - Proteus software
   - Arduino IDE
3. **Prerequest**
   - Basic electronic
   - C programming
4. **Introduction**

   This tutorial is the definitive guide for **Arduino ADC & analogRead() For Analog Input Voltage**. We'll start off by explaining how an ADC work, what are the Arduino ADC characteristics and how to make the best use of it. We'll also discuss the analogRead() function and how to use it for reading analog inputs with Arduino.

   Then, we'll implement a couple of Arduino example projects (**LED Dimmer**, and **DC Motor Speed Control**) **Using Arduino ADC & analogRead**. You'll also learn a lot of tips and tricks for Arduino ADC & analogRead that will help you in your Arduino measurement projects. It's a fundamental topic in this Arduino Programming Series of Tutorials, so make sure you get the hang of it. And let's get right into it!

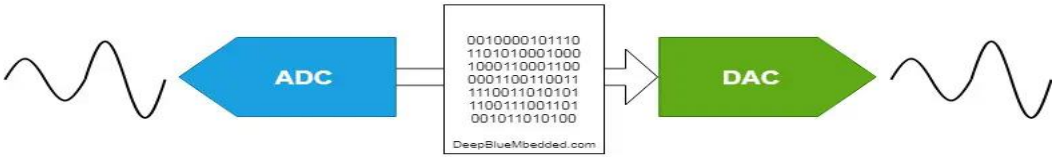
5. **Introduction To ADC**

   An **ADC** (**A**nalog to **D**igital **C**onverter) is an electronic circuit that's usually integrated into different microcontrollers or comes in as a dedicated IC. We typically use an ADC in order to measure/read the analog voltage from different sources or sensors.

   Most parameters and variables are analog in nature and the electronic sensors that we use to capture this information are also analog. Just like temperature, light, pressure, and other sensors are all analog.

   Therefore, we need to read the analog voltage value using our digital microcontrollers. And this is when we start learning about the ADC peripheral and try to configure it so as to get this task done.

   The ADC does the inverse operation of a DAC. While an ADC (A/D) converts analog voltage to digital data, the DAC (D/A) converts digital numbers to an analog voltage on the output pin.



This is an in-depth article (tutorial) on ADC, how it works, different types of ADC, error sources, sampling, and much more. Consider checking it out if it's your first time learning about the ADC.

Arduino ADC (Analog To Digital Converter)

The **ADC** (**A**nalog to **D**igital **C**onverter) in Arduino is used to read analog voltage input and convert it to its digital representation. This is essential for interfacing various types of sensors and modules with Arduino which provide an analog voltage output. The Arduino UNO (atmega328p microcontroller) has a total of **6 analog input pins** that are internally connected to the ADC to be used for reading analog voltage inputs. The Arduino's internal ADC is **10 Bits in resolution**, which means it has an output **range** of **0 up to 1023**.

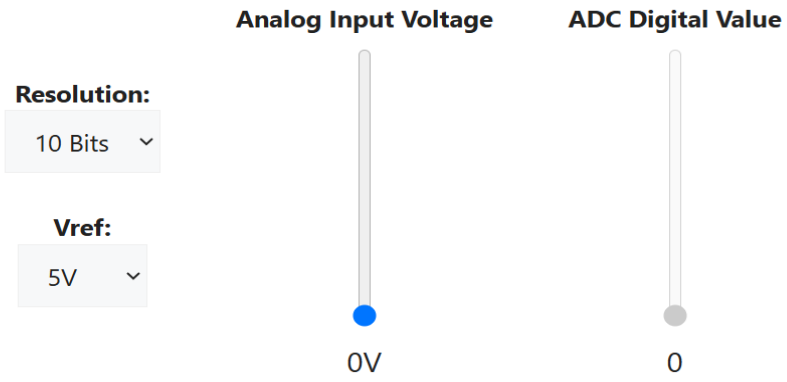
The analog voltage reference for Arduino ADC is by default $\mathbf{V_{REF}}$ **= +5v**, which means we can measure analog input voltage up to 5v. Above that limit, the input pins can get damaged and the ADC output will saturate at 1023 anyway. So you need to be careful and stay within the working voltage limits.

In this section, we'll dive deeper into the Arduino ADC characteristics and parameters to make the best use of it and to also know its fundamental limitations.

Arduino ADC Resolution

The **Arduino ADC resolution is 10 bits**, the digital output range is therefore from 0 up to 1023. And the analog input range is from 0v up to 5v (which is the default analog reference voltage V_{REF} = +5v).

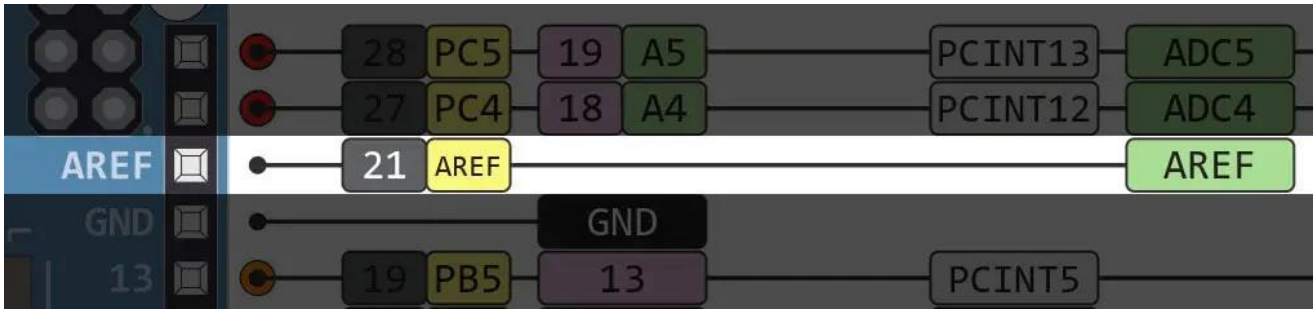
You can use the interactive tool below to set an analog input voltage and see the ADC digital output value that corresponds to the analog input voltage. The output equation for the ADC is as follows: **ADC Output = (Analog input voltage / $\mathbf{V_{REF}}$) x ($\mathbf{2^n - 1}$)**.



Arduino ADC Reference Voltage

The **AREF** (**A**nalog **Ref**erence) pin can be used to provide an external reference voltage for the analog-to-digital conversion of inputs to the analog pins (A0-A5). The reference

voltage essentially specifies the maximum analog input voltage after which the ADC's output will saturate at 1023.



Let's recall the ADC's output equation which is: **ADC Output = (Analog input voltage / V_{REF}) x (2^n – 1)**. Where the "default" V_{REF} = **5v** and **n** is the ADC resolution which is **10**bits.

Using the AREF pin we can change the default VREF voltage and set it to any desired voltage level which allows for more precise A/D conversions. Let's say we hook up the AREF pin to an external 2.5v DC supply, now we'll end up having double the resolution of the default 5v analog reference.

Because instead of dividing the 0v-5v range into discrete 1024 digital levels, we'll now divide the 0v-2.5v range into discrete 1024 digital levels. Which is exactly double the default resolution. But you must make sure that the input analog voltage will not exceed 2.5v after setting the AREF to 2.5v. Otherwise, the ADC will saturate at 1023 if the analog input voltage exceeds the new AREF (2.5v).

Use the interactive ADC tool above to test it yourself and see the effect of lowering the AREF voltage from 5v down to 2.5v and how it increases the ADC resolution by a factor of 2.

V_{REF}	Analog Input Voltage	ADC Digital Output
5v	0.01v	2
2.5v	0.01v	4

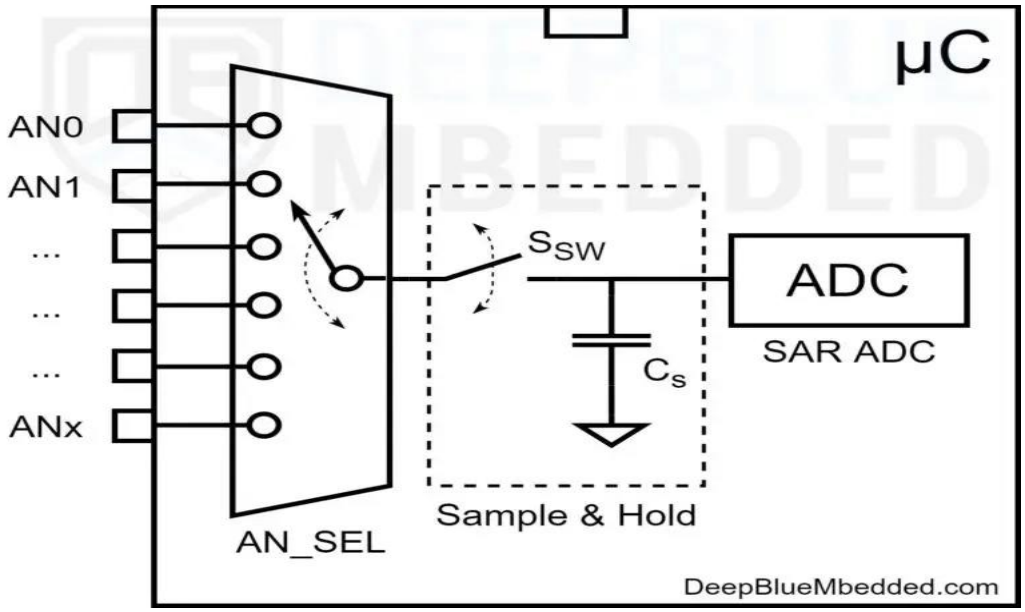
For the exact same analog input (0.01v), you get more ADC ticks (levels) at V_{REF}=2.5v compared to V_{REF}=5v. This is what we mean by the resolution being doubled (increased by a factor of 2).

Arduino ADC Sampling Time

The ADC in Arduino or any other microcontroller can't just convert the analog voltage on a pin that is continuously changing. Therefore, it needs to take a snapshot of the analog voltage on the pin at a specific moment and store that voltage in a small capacitor

(Sampling Capacitor "C_S"). The voltage copy on the capacitor will remain constant after the sampling switch (S_{SW}) is opened.

Therefore, the ADC will have a stable copy of the analog input pin voltage available on the C_S capacitor and it can start conversion with no issue. The whole process of closing the sampling switch S_{SW} until the sampling capacitor is charged to the exact same voltage level of the input pin, then reopening the S_{SW} again is what we call **ADC Sampling**.



As stated in the Atmega328p datasheet, the ADC sampling time is around **1.5 to 2 ADC Clocks**. Given that the default **ADC Clock Speed** (in Arduino UNO) is **125kHz**, therefore, the **ADC Sampling Time** is **16µs**.

❗ Note

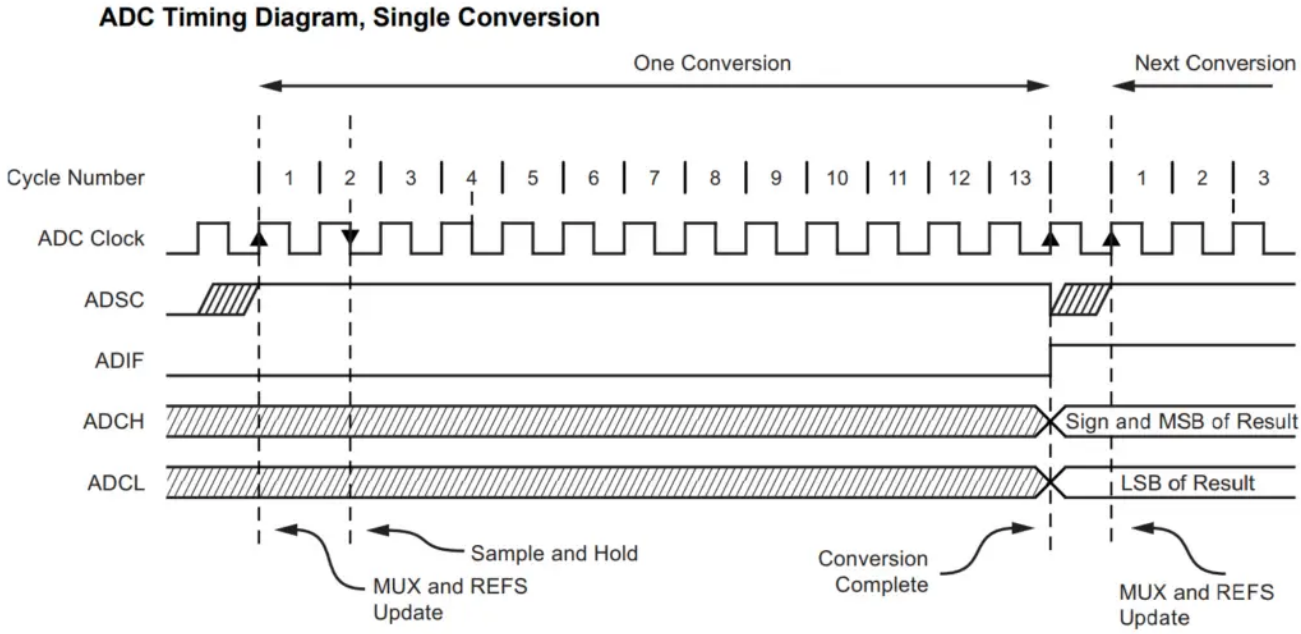
The ADC has a maximum clock speed of 200kHz, but it needs to run at a lower clock frequency in order to achieve the 10-bit resolution. That's why the Arduino ADC default clock frequency is set to **125kHz**.

Arduino ADC Conversion Time

The ADC conversion time usually refers to the time it takes the ADC to complete a single A/D conversion operation. The sampling process that we've discussed in the previous section is part of the ADC conversion process. It actually proceeds the SOC (start of conversion) that triggers the SAR (successive approximation register) ADC to start its operation.

The SAR ADC takes around 11.5 ADC clocks to complete a single 10-Bit A/D conversion process. Add to that the sampling time which is 1.5 ADC clock. Therefore, we end up with a conversion time of **13 ADC Clocks**.

Given that the default **ADC Clock Speed** (in Arduino UNO) is **125kHz**, therefore, the **ADC Conversion Time** is **104µs**. Which is the typical (default) ADC conversion time for Arduino UNO (atmega328p).



Arduino ADC Sampling Rate

The ADC Sampling Rate is a measure of an ADC speed which means the number of conversions (samples) that the ADC can take per second. And it's expressed in ksps or Msps (kilo or Mega Samples Per Second).

To find out the ADC's maximum sampling rate, we'll run it at the maximum ADC clock speed (which is 200kHz). Given that a single A/D conversion takes 13 ADC Clocks, therefore the **conversion time at maximum speed is 65µs**.

Therefore, the **Arduino ADC Sampling Rate is 15.4k samples per second**. (1/65µs)

Arduino ADC Speed

The Arduino ADC speed also refers to the ADC's sampling rate. By default, the ADC clock is 125kHz, resulting in a conversion time of 104µs. And therefore, the **Arduino ADC default Speed is 9,600 samples per second**.

By increasing the ADC clock speed, we'll be sacrificing some resolution (accuracy) of course, but this will definitely increase the ADC Speed. At maximum ADC clock speed (200kHz), the conversion time goes down to 65µs). And therefore, the **Arduino ADC Max Speed is 15,400 samples per second**.

Arduino ADC Analog Pins

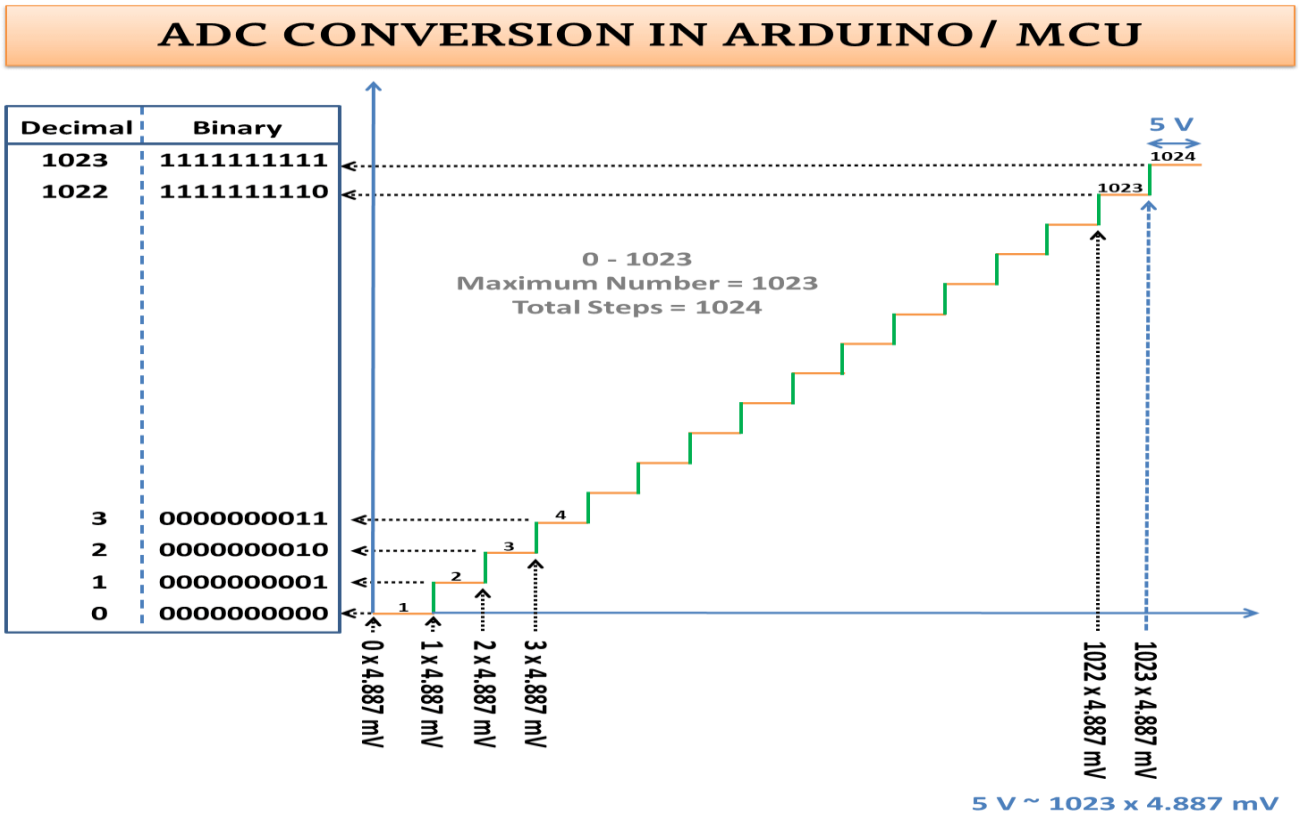
The Arduino UNO has 6 analog input pins labeled from A0 to A5 as shown in the figure below. Those pins can be used with analog peripherals in the Arduino microcontroller such as ADC (A/D Converter) and the Analog Comparator.

Arduino analogRead() Function

Reads the value from the specified analog pin. Arduino boards contain a multichannel, 10-bit ADC. This means that it will map the analog input voltages between 0 and the V_{REF} voltage (5V or 3.3V) into integer values between 0 and 1023.

On an Arduino UNO, for example, this yields a resolution between readings of (5 volts / 1024 units) or, 0.0049 volts (**4.9 mV**) per unit. In other words, the smallest input voltage change that the ADC can sense is around 4.9 mV.

In Arduino, we have an ADC converter chip inbuilt in the microcontroller ATMEGA328. This ADC is a 10-bit converter.



analogRead Syntax

```
analogRead(pin);
```

analogRead Parameters

pin: the name of the analog input pin to read from (A0 to A5 on most boards, A0 to A6 on MKR boards, A0 to A7 on the Mini and Nano, A0 to A15 on the Mega).

analogRead Example

```
AN0_Result = analogRead(A0);
```

analogRead Return

The analog reading on the pin. Although it is limited to the resolution of the analog to digital converter (0-1023 for 10 bits or 0-4095 for 12 bits). Data type: int.

Arduino analogRead Range

The analogRead() function returns a value within the 10 bits resolution range (0 – 1023).

Which corresponds to an analog input voltage range (0v – 5v).

Arduino analogRead Speed

To measure, the analogRead() function speed, I've developed the measurement code example below. Which uses Arduino Direct Pin Manipulation to toggle an IO pin before and after 4x calls to the analogRead() function "to be measured".
The resulting pulse waveform is captured by my oscilloscope and we'll get the pulse width and divide it by 4 to get the total execution time of the analogRead() function.

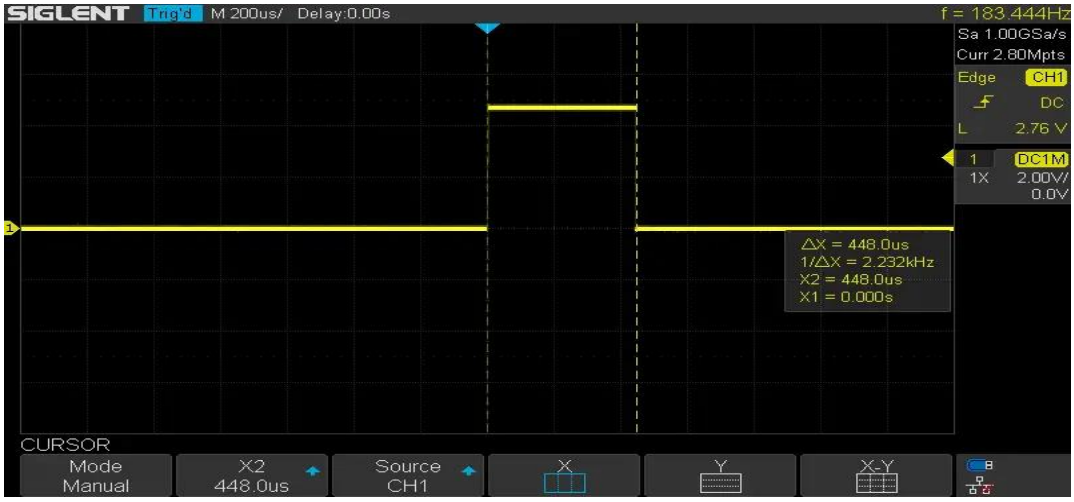
Here is the code example for analogRead() speed measurement.

```
/* LAB Name: Arduino AnalogRead Speed Measurement
* Author: Khaled Magdy
* For More Info Visit: www.DeepBlueMbedded.com*/
// ---[ Arduino Pin Manipulation Macros ]---
#define SET_PIN_MODE_OUTPUT(port, pin) DDR ## port |= (1 << pin)
#define SET_PIN_HIGH(port, pin) (PORT ## port |= (1 << pin))
#define SET_PIN_LOW(port, pin) ((PORT ## port) &= ~(1 << (pin)))

int AN0_Result = 0;
void setup()
{
  SET_PIN_MODE_OUTPUT(B, 0);  // Set Pin8 (PB0) As Output
}
void loop()
{
  SET_PIN_HIGH(B, 0);
```

```
  AN0_Result = analogRead(A0);
  AN0_Result = analogRead(A0);
  AN0_Result = analogRead(A0);
  AN0_Result = analogRead(A0);
  SET_PIN_LOW(B, 0);
  delay(5);
}
```

Here is the result after running this code on my Arduino UNO board and taking the measurement on my DSO.



It's around 448µs for the 4x calls to the analogRead() function. Therefore, the **analogRead() speed is 112µs**.

Given that the ADC default A/D conversion time is 104µs as we've seen earlier, the analogRead() function takes another 8µs for housekeeping logic operations, CPU context switching, and such.

Arduino Analog Input (Reading Analog Voltage)

Now, let's move to reading actual analog input voltage using Arduino ADC analog input pins. We'll divide this section into 3 parts for all possible sinareos and how to deal with each case to maximize the ADC measurements accuracy.

In your projects you'll be dealing with different levels of analog voltages that you may need to measure with the ADC analog input pins. The analog voltage input can fall into one of the following categories:

1. Typical Range (0v – 5v)
2. High Voltage (>5v)
3. Small Signal (<1v)

And you need to deal with each type of analog input voltage in a different way to get accurate and reliable results of the ADC.

Arduino Analog Input (Typical Range 0v-5v)

For analog input voltage within the typical range of (0v – 5v), you just connect the "to be measured" voltage signal to the Arduino's analog input pin directly. And you can just read the pin using the analogRead() function. Here is an example:

```
int AN0_Result = 0;
float AN0_Voltage = 0;
```

```
void setup()
{
  // NOTHING To Do
}
void loop()
{
  AN0_Result = analogRead(A0);
  AN0_Voltage = AN0_Result * (5 / 1024);
  delay(10);
}
```

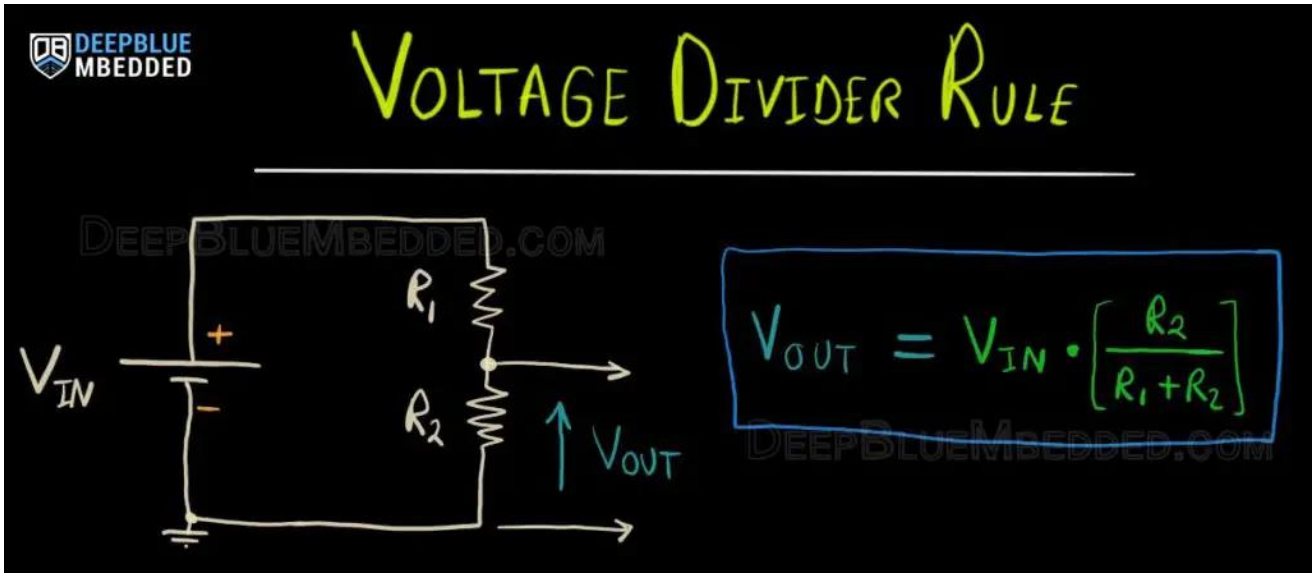
No extra care is needed while dealing with a signal in the typical default range (0v – 5v).

Arduino Analog Input (High Voltage >5v)

If you need to measure an analog voltage that's higher than 5v, you just can't hook it up to the Arduino analog input pin for two reasons. First of which and most importantly, it's going to damage the IO pin which can't handle any voltage above +5.5v. And secondly, the ADC will saturate at 1023 the moment you reach 5v on the input pin, any voltage change above that won't induce a change in the ADC reading that will stay at 1023.

Attenuate The Input Voltage (Voltage Divider)

The proper way to read high voltage using Arduino's ADC is to create a voltage divider network to attenuate the "to be measured" voltage with a specific ratio. So that the voltage present on the analog input pins falls within the (0v – 5v) range. Now, it's easy to read the analog input voltage on the pin and reconstruct the actual "to be measured" voltage by multiplying the measured voltage by the attenuation ratio.



Arduino Analog Input (Small Signal <1v)

If you're reading a sensor or an electronic device that has a small signal output voltage (less than 1v), the Arduino's ADC in default settings will not be sufficient for this task. You'll be losing a lot of resolution actually because of the default V_{REF} being 5v.

Go back to the Arduino ADC interactive tool, set the V_{REF} to 5v, and change the input voltage slider up and down within the 1v range. You'll notice that we're leaving 80% of the ADC range completely unused!

1- Reduce The V_{REF} To Enhance The Resolution

By reducing the V_{REF} down to 1v, you can see that we're now using the full ADC range and the measurement resolution has improved significantly. At the default V_{REF} = 5v setting, the minimum voltage change the ADC could sense was **4.88mv**. After reducing the V_{REF} down to 1v, the ADC can now sense down to **1mv** of change in the analog input voltage.

2- Amplify The Input Signal

Another solution, would be to amplify the input signal itself. Especially if we're talking about a few mv analog input signal. This is extremely challenging for the Arduino's 10-Bit ADC. But it becomes doable and much easier if we amplify the small signal with an Op-Amp based signal conditioning circuit.

This will indeed facilitate the job of our ADC to measure the voltage that was a few millivolts and now became (0v – 5v). But it brings in some new challenges that we need to deal with. Things like Op-Amp zero-drift, nonlinearity, early saturation, etc.

6. How to Display ADC value on LCD using Arduino

6.1. Introduction

Arduino has 10 bit ADC pins so whenever you apply voltage on these pins it will give you a value ranging from 0 to 1023 depending on the voltage provided. One can easily get this value using a simple function in Arduino analogRead(); but the real problem is to convert this analog value into the actual voltage present on the pin. Suppose you are using A0 pin of arduino and you are providing 3.3V over to this pin, now when you use this analoagRead() function then it will give you some value say 543, but you wanna know what's the actual voltage at this pin which is 3.3V so now converting this 543 to 3.3 is a bit tricky part. It's not difficult but involves a little calculations, which I am gonna cover today in detail. Before going any further, make sure you have already installed the Arduino Library For Proteus, if not then first do it because without this library you won't be able to use Arduino board in Proteus. So, let's get started with **How to Display ADC value on LCD using Arduino**.

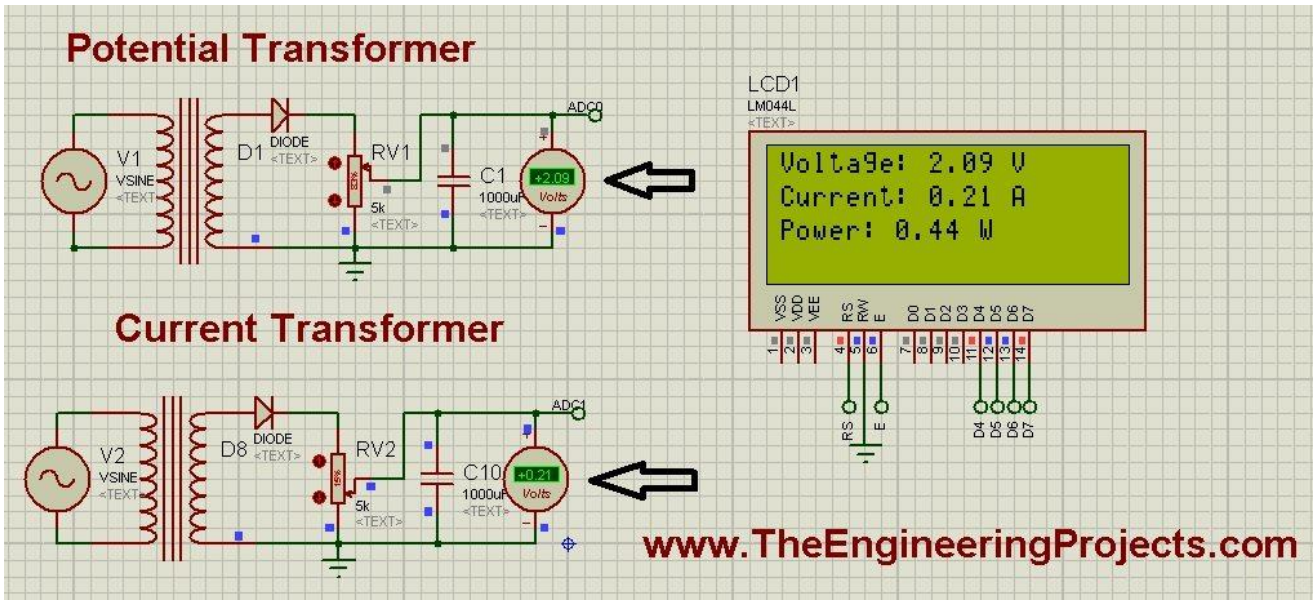
6.2. Steps

6.2.1. Step1: Circuit Designing in Proteus

- First of all, I have designed a circuit in Proteus for Displaying ADC value on LCD using Arduino.
- In this circuit, I have used two transformers which I have named as Potential Transformer and Current Transformer. I am supplying 220V to these transformers which is then converted into 5V.
- I have set the turn ratio of these transformers such that they give maximum 5V at the output.
- Now,rest of the circuit is simple, I have just connected the LCD with Arduino so that we could display these ADC value over to LCD.

Note:

- Before connecting any pin to ADC, make sure that the output should be less than 5V. Otherwise, there's a chance that the pin burnt out.
- If you don't know about LCD then read Ciruit Designing of LCD with Arduino in Proteus.
- You should also download this New LCD Library for Proteus.
- Here's the circuit diagram of displaying ADC value on LCD using Arduino in Proteus ISIS:



6.2.2. Step 2: Arduino Code Designing

- Now copy the below code and paste it into Arduino software. Compile your code and get the Arduino hex file.
- If you dont know How to get the hex file from Arduino then read Arduino Library for Proteus, I have explained it in detail there.

```
#include <LiquidCrystal.h>;
#define NUM_SAMPLES 10
int sum = 0;
unsigned char sample_count = 0;
float voltage = 0.0;
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
const int PT = A0;
const int CT = A1;
float Cur;
float Vol;
float Power;
<strong>void setup()</strong> {
// set up the LCD's number of columns and rows:
lcd.begin(20, 4);
// Print a message to the LCD.
```

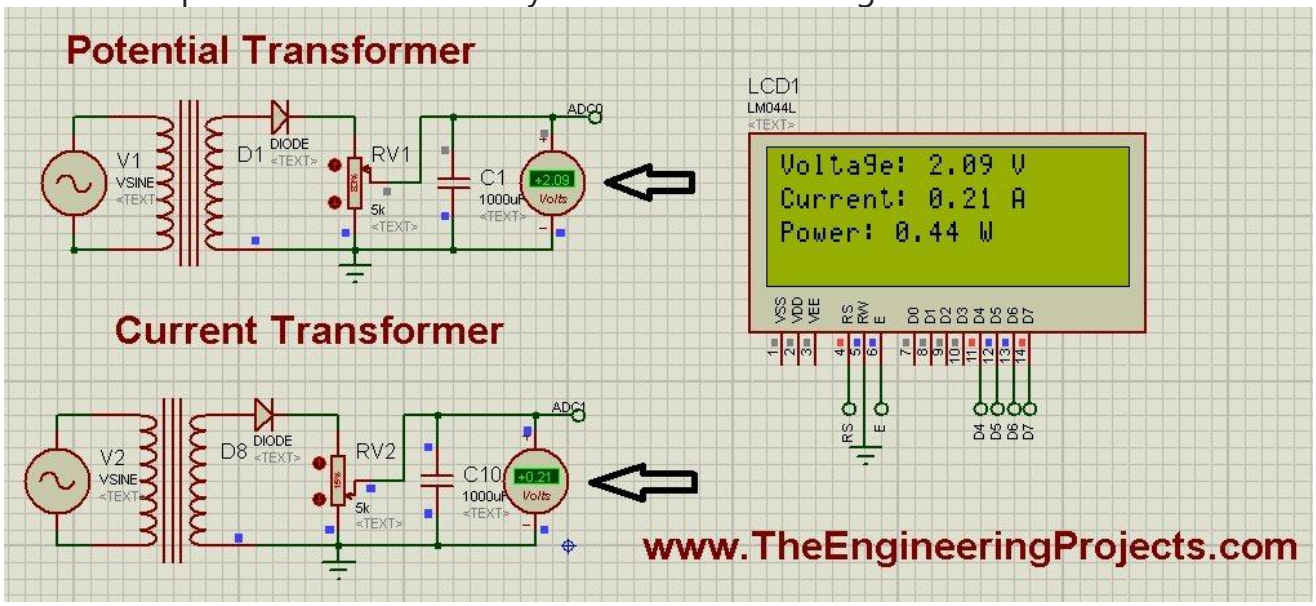
```
lcd.setCursor(6,1);
lcd.print("Welcome To");
lcd.setCursor(5,2);
lcd.print("Energy Meter");
//delay(5000);
lcd.clear();
Constants();
}
<strong>void loop()</strong> {
lcd.setCursor(0, 2);
ShowVoltage(9, 0, PT);
Vol = voltage;
ShowVoltage(9, 1, CT);
Cur = voltage;
Power = Vol * Cur;
lcd.setCursor(7,2);
lcd.print(Power);
}
<strong>void Constants()</strong>
{
lcd.setCursor(0,0);
lcd.print("Voltage: ");
lcd.setCursor(0,1);
lcd.print("Current: ");
lcd.setCursor(0,2);
lcd.print("Power: ");
lcd.setCursor(14,0);
lcd.print("V");
lcd.setCursor(14,1);
lcd.print("A");
lcd.setCursor(12,2);
lcd.print("W");
}
<strong>void ShowVoltage (int x,int y, unsigned int value)</strong>
{
while (sample_count &lt; NUM_SAMPLES)
{
sum += analogRead(value);
sample_count++;
delay(10);
}
voltage = ((float)sum / (float)NUM_SAMPLES * 5.015) / 1024.0;
```

```
lcd.setCursor(x, y);
lcd.print(voltage);
sample_count = 0;
sum = 0;
}
```

- The code is quite simple and self explanatory, the only difficulty is in ShowVoltage function. In this function, I have first taken an average of 10 ADC values and after that I have applied a simple formula over it and then it will start start giving the voltage value which I have simply displayed over the LCD.
- Now everything's done, so Get your Hex File from Arduino Software and let's check the results whether it displayed ADC value on LCD using Arduino or not

6.2.3. Step 3: Result

- We have designed the electronic circuit in Proteus and have also designed our code and uploaded the hex file in Arduino.
- Now press start button and you will see something like this:



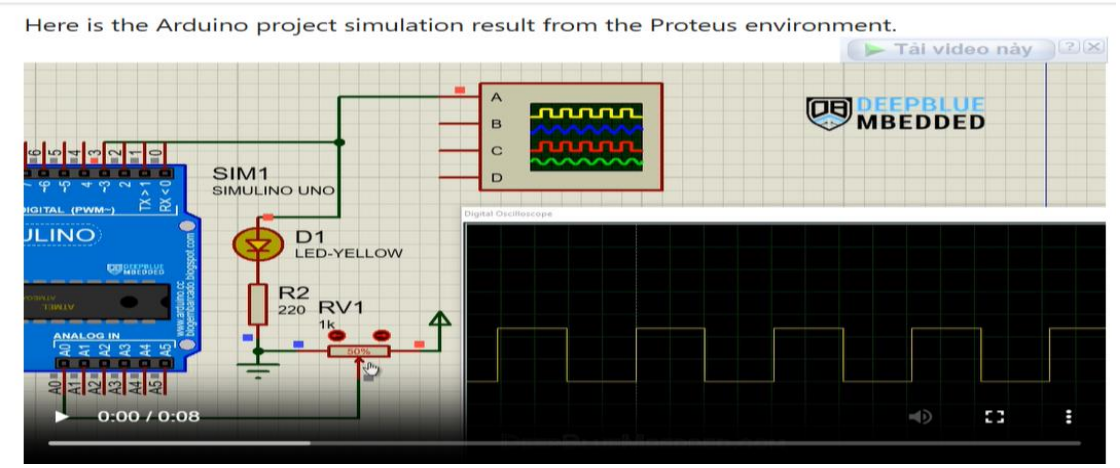
- Now if you compare the voltages in voltmeter and on LCD, you can see they are exactly the same. You can check the value of variable resistor and the values in LCD will also change as the voltage in voltmeter change.

That's all for today, hope I have conveyed some knowledge today and now you can easily *Display ADC value on LCD using Arduino*. In the next post we will explore more Arduino features. Till then take care and have fun.

7. Arduino ADC Example – LED Dimmer

In this example project, we'll control an LED brightness with an Arduino PWM output pin. We'll use a potentiometer and the analogRead() function to get the potentiometer reading and use it to control the PWM's duty cycle for LED Dimming.

7.1. Shematic



7.2. Code Example

```
/* LAB Name: Arduino LED Dimmer (Potentiometer + PWM)
* Author: Khaled Magdy
* For More Info Visit: www.DeepBlueMbedded.com*/
#define LED_PIN 3
void setup()
{
  pinMode(LED_PIN, OUTPUT);
}

void loop()
{
  analogWrite(LED_PIN, (analogRead(A0)>>2));
}
```

7.3. Code Explanation

First of all, we define the IO pin used for the LED output. It has to be a PWM-enabled IO pin (for UNO: 3, 5, 6, 9, 10, or 11).

```
#define LED_PIN 3
```

setup()

in the setup() function, we'll set the pinMode to be output.

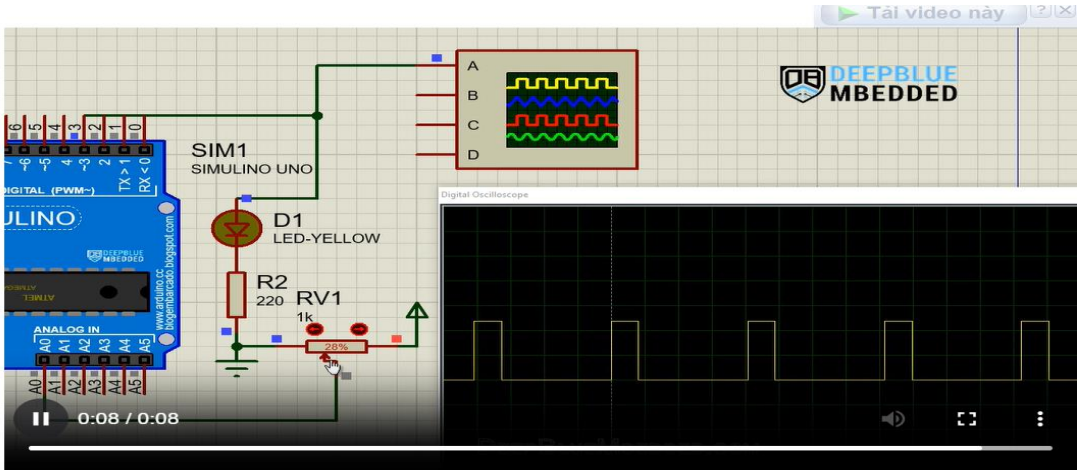
```
pinMode(LED_PIN, OUTPUT);
```

loop()

in the loop() function, we'll read the analog input pin of the potentiometer (which is **10-Bit** resolution) and convert it to **8-Bit** resolution by dividing by 4 (or right-shifting twice " >>2"). Now, we've mapped the analog input value to the PWM duty cycle range.
Using Arduino analogWrite() function, we'll write back the ADC scaled reading as a duty cycle for the PWM to control the speed of the motor accordingly.

```
analogWrite(LED_PIN, (analogRead(A0)>>2));
```

7.4. Result

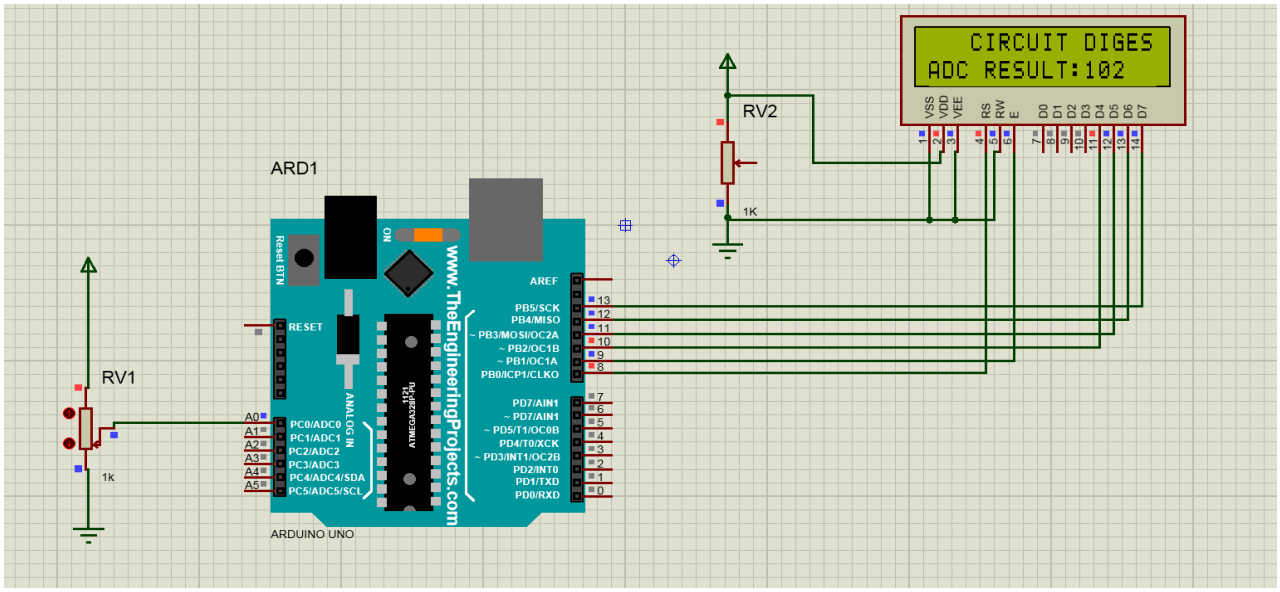


8. Arduino ADC Example – DC Motor Speed Control

In this example project, we'll control a DC Motor's speed with an Arduino PWM output pin + a BJT Transistor. We'll use a potentiometer and the analogRead() function to get the potentiometer reading and use it to control the PWM's duty cycle and consequently the motor's speed.

9. Display ADC value on LCD

9.1. Shematic



9.2. Connections

PIN1 or VSS to ground

PIN2 or VDD or VCC to +5v power

PIN3 or VEE to ground (gives maximum contrast best for a beginner)

PIN4 or RS (Register Selection) to PIN8 of ARDUINO UNO

PIN5 or RW (Read/Write) to ground (puts LCD in read mode eases the communication for user)

PIN6 or E (Enable) to PIN9 of ARDUINO UNO

PIN11 or D4 to PIN10 of ARDUINO UNO

PIN12 or D5 to PIN11 of ARDUINO UNO

PIN13 or D6 to PIN12 of ARDUINO UNO

PIN14 or D7 to PIN13 of ARDUINO UNO

The ARDUINO IDE allows the user to use **LCD in 4 bit mode**. This type of communication enables the user to decrease the pin usage on ARDUINO, unlike other the ARDUINO need not be programmed separately for using it in 4 it mode because by default the ARDUINO is set up to communicate in 4 bit mode. In the circuit you can see we used 4bit communication (D4-D7). So from mere observation from above table we are connecting 6 pins of LCD to controller in which 4 pins are data pins and 2 pins for control.

9.3.Source code

For interfacing an LCD to the ARDUINO UNO, we need to know a few things

1. analogRead(pin);
2. analogReference();
3. analogReadResolution(bits);

First of all the UNO ADC channels has a default reference value of 5V. This means we can give a maximum input voltage of 5V for ADC conversion at any input channel. Since some sensors provide voltages from 0-2.5V, with a 5V reference we get lesser accuracy, so we have a instruction that enables us to change this reference value. So for changing the reference value we have ("analogReference();")

As default we get the maximum board ADC resolution which is 10bits, this resolution can be changed by using instruction ("analogReadResolution(bits);"). This resolution change can come in handy for some cases.

Now if the above conditions are set to default, we can read value from ADC of channel '0' by directly calling function "analogRead(pin);", here "pin" represents pin where we connected analog signal, in this case it would be "A0". The value from ADC can be taken into an integer as "int ADCVALUE = analogRead(A0); ", by this instruction the value after ADC gets stored in the integer "ADCVALUE".

NOW let's talk a bit about 16x2 LCD. First we need to enable the header file ('#include <LiquidCrystal.h>'), this header file has instructions written in it, which enables the user to interface an LCD to UNO in 4 bit mode without any fuzz. With this header file we need not have to send data to LCD bit by bit, this will all be taken care of and we don't have to write a program for sending data or a command to LCD bit by bit.

Second we need to tell the board which type of LCD we are using here. Since we have so many different types of LCD (like 20x4, 16x2, 16x1 etc.). In here we are going to interface a 16x2 LCD to the UNO so we get 'lcd.begin(16, 2);'. For 16x1 we get 'lcd.begin(16, 1);'.

In this instruction we are going to tell the board where we connected the pins, The pins which are connected are to be represented in order as "RS, En, D4, D5, D6, D7". These pins are to be represented correctly. Since we connected RS to PIN0 and so on as show in circuit diagram, We represent the pin number to board as "LiquidCrystal lcd(0, 1, 8, 9, 10, 11);".

After above there all there is left is to send data, the data which needs to be displayed in LCD should be written as " cd.print("hello, world!");". With this command the LCD displays 'hello, world!'.

As you can see we need not worry about any this else, we just have to initialize and the UNO will be ready to display data. We don't have to write a program loop to send the data BYTE by BYTE here.

```
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins

LiquidCrystal lcd(8, 9, 10, 11, 12, 13); // REGISTER SELECT PIN,ENAB
LE PIN,D4 PIN,D5 PIN, D6 PIN, D7 PIN

char ADCSHOW[5]; //initializing a character of size 5 for showing th
e ADC result

void setup()

{

// set up the LCD's number of columns and rows:
```

```
lcd.begin(16, 2);

}

void loop()

{

// set the cursor to column 0, line 1

lcd.print("   CIRCUIT DIGEST"); //print name

lcd.setCursor(0, 1); // set the cursor to column 0, line

lcd.print("ADC RESULT:"); //print name

String ADCVALUE = String(analogRead(A0)); //intailizing a string and
storing ADC value in it

ADCVALUE.toCharArray(ADCSHOW, 5); // convert the reading to a char a
rray

lcd.print(ADCSHOW); //showing character of ADCSHOW

lcd.print("   ");

lcd.setCursor(0, 0); // set the cursor to column 0, line1


}
```

9.4.

10. Questtions

What is ADC in Arduino?

The Arduino **ADC** (**A**nalog to **D**igital **C**onverter) is a peripheral in the Arduino's microcontroller that takes an analog input voltage on a pin and converts it to a digital value. We typically use an ADC in order to measure/read the analog voltage from different sources or sensors.

What type of ADC is Arduino UNO?

Arduino UNO has an ADC type called **SAR** (**S**uccessive **A**pproximation **R**egister) with a resolution of 10 bits.

What is analogRead() in Arduino?

The Arduino analogRead() function allows the user to read an ADC channel (pin) and convert the analog voltage input to its digital representation and return that value. It's got a range of (0 – 1023) that corresponds to analog input voltages (0v – 5v).