HACETTEPE UNIVERSITY

COMPUTER ENGINEERING

BBM 434 – EMBEDDED SYSTEMS LAB.

EXPERİMENTS Reports

Advisor: Assoc.Prof. Harun ARTUNER

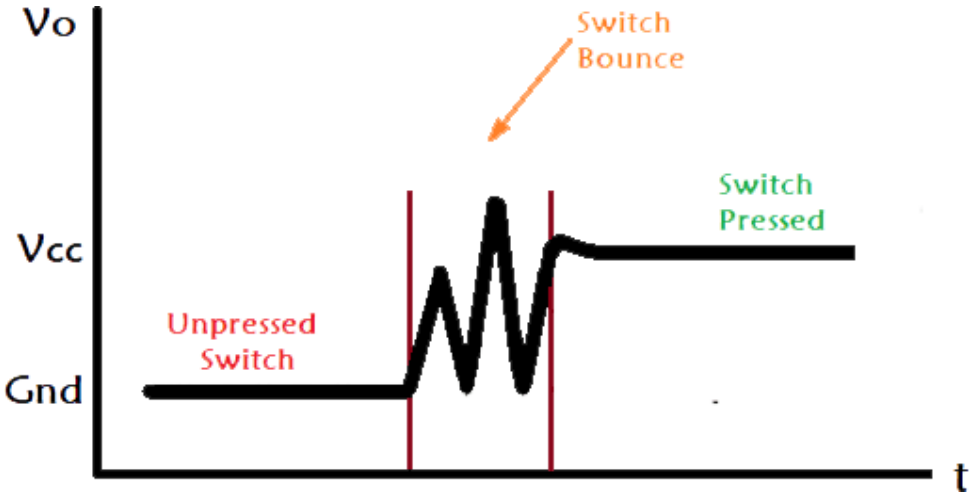
Ayoub O. W. Othman
21202955
ayoub.oothman@gmail.com

TABLE OF CONTENTS

1. Switch Debouncing

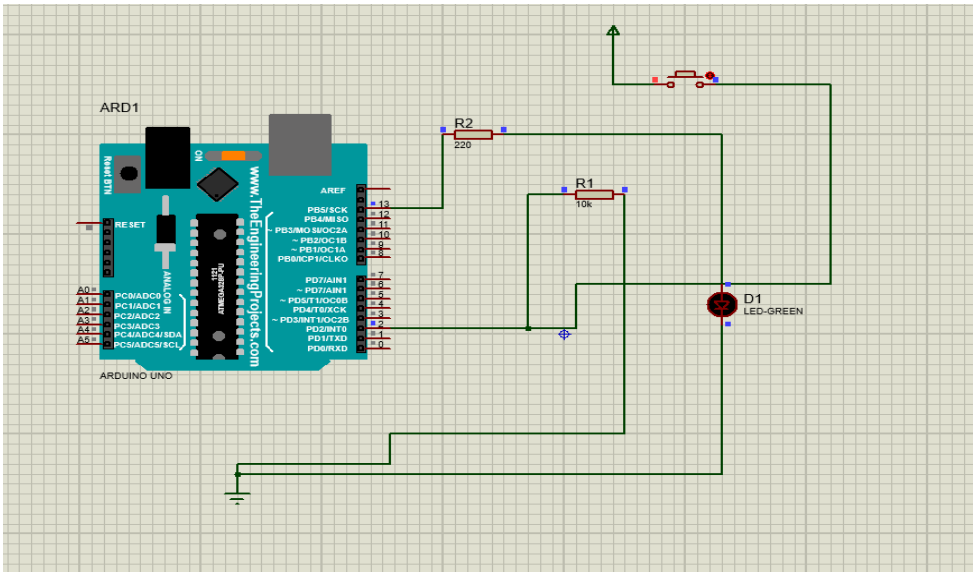
What is Switch Bouncing?

When we press a pushbutton or toggle switch or a micro switch, two metal parts come into contact to short the supply. But they don't connect instantly but the metal parts connect and disconnect several times before the actual stable connection is made. The same thing happens while releasing the button. This results the false triggering or multiple triggering like the button is pressed multiple times. Its like falling a bouncing ball from a height and it keeps bouncing on the surface, until it comes at rest.



What is Software Debouncing?

Debouncing occurs in software also, while programming programmers add delays to get rid ofsoftware debouncing. Adding a delay force the controller to stop for a particular time period, but adding delays is not a good option into the program, as it pause the program and increase the processing time. The best way is to use interrupts in the code for software bouncing.



```
const int ledPin = 13; //led attached to this pin
const int buttonPin = 2; //push button attached to this pin

int buttonState = LOW;
int ledState = -1;

long lastDebounceTime = 0;
long debounceDelay = 50;    // the debounce time; increase if the output flickers

void setup() {

  //set the mode of the pins...
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);

}//close void setup

void loop() {


  buttonState = digitalRead(buttonPin);

  //filter out any noise by setting a time buffer
  if ( (millis() - lastDebounceTime) > debounceDelay) {


    if ( (buttonState == HIGH) && (ledState < 0) ) {

      digitalWrite(ledPin, HIGH); //turn LED on
      ledState = -ledState; //now the LED is on, we need to change the state
      lastDebounceTime = millis(); //set the current time
    }
    else if ( (buttonState == HIGH) && (ledState > 0) ) {

      digitalWrite(ledPin, LOW); //turn LED off
      ledState = -ledState; //now the LED is off, we need to change the state
      lastDebounceTime = millis(); //set the current time
    }

  }

}
```

2. Prepare a Practical Working Environment

What is Proteus and How Does it Compare to Other Simulation Software?

Proteus is a simulation and electronic design development tool developed by Lab Center Electronics. It is a very useful tool as it ensures that the circuit design or firmware code is working properly before you begin to physically work on it.

It has an extensive number of components in its library which can be used to virtually design your circuit. The designs you make can be easily compiled and debugged through Proteus's virtual meters (voltmeter and ammeter), oscilloscope, serial monitor, and more.

TinkerCAD is also another tool that can be used for simulation.Developed by AutoDesk, it is a cloud-based software which is only limited to Arduino simulation.

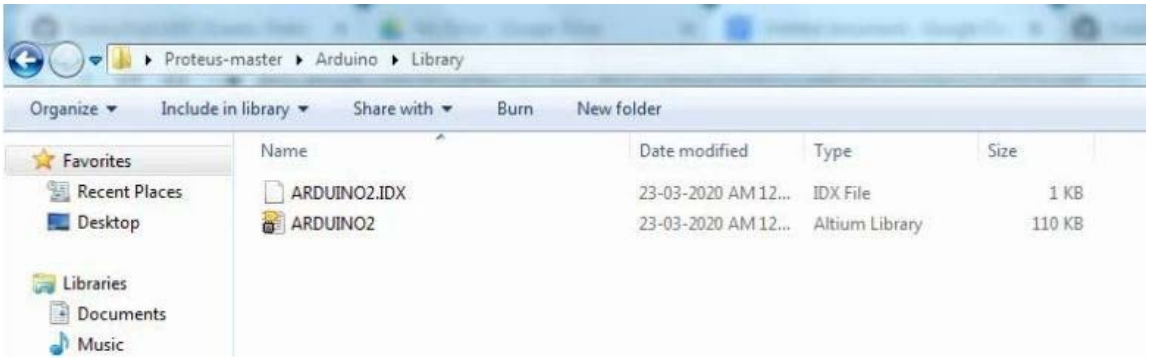
If you are a beginner to Arduino and circuit design, then I recommend you try out TinkerCAD. If you want to work on a cool project which involves some circuit design, I recommend Proteus.

How to Install an Arduino Library in Proteus?

Proteus doesn't come with an inbuilt Arduino library, so you have to install it externally. Follow the steps below to install it on your PC.

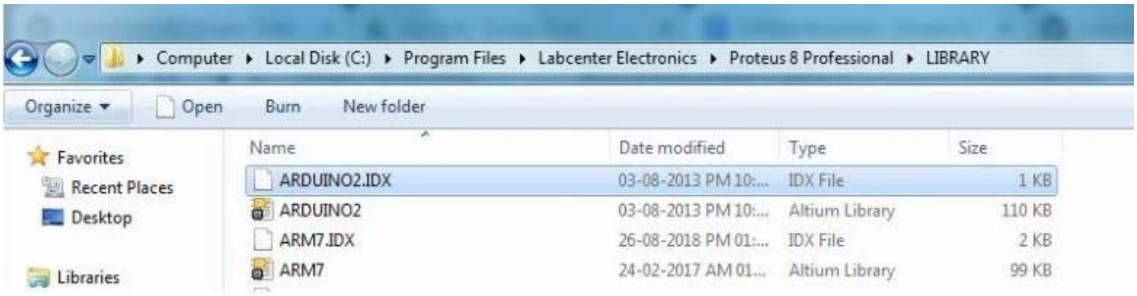
1. Download all library for Arduino

2. Extract the zip file and navigate to Proteus-master\Arduino\Library.



3. Copy both of the files and paste them in one of the following paths:

C:\Program Files\Labcenter Electronics\Proteus 8 Professional\LIBRARY
or
C:\Program Files\Labcenter Electronics\Proteus 8 Professional\Data\LIBRARY

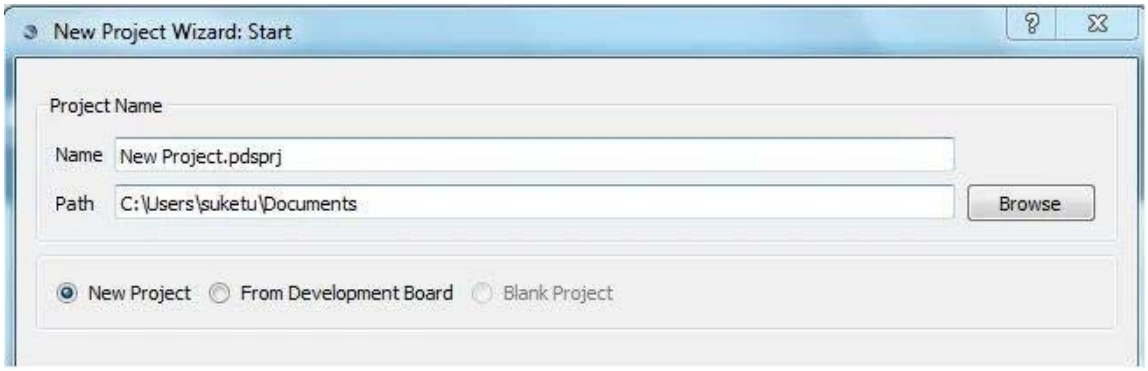


Now, open Proteus and check whether the Arduino board libraries are installed properly or not.

4. In Proteus, create a new project.

Create a new project in Proteus.

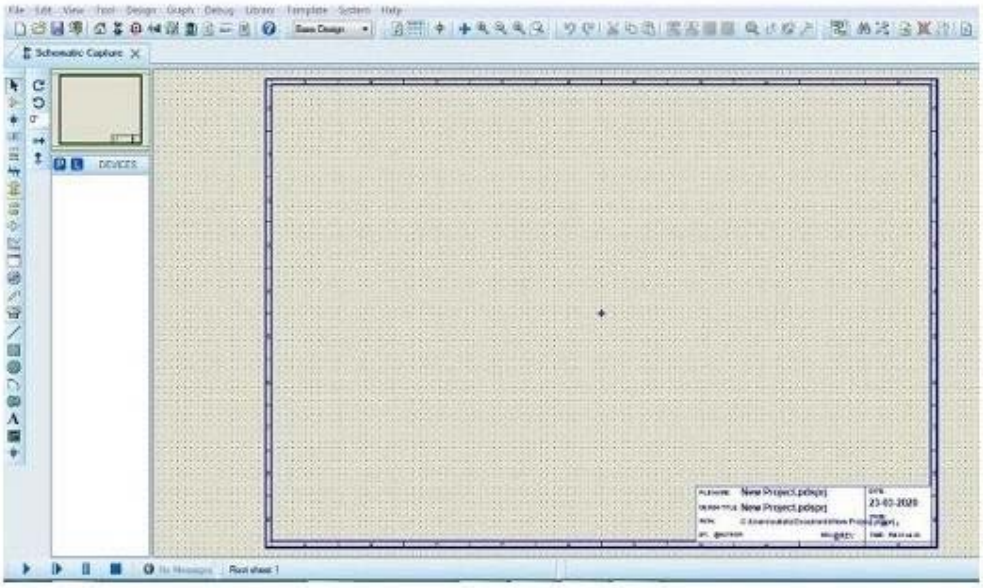
5. Decide where you would like to save your project.



Choose where to save your project.

Click Next once you are done and select the appropriate page layout according to your needs.

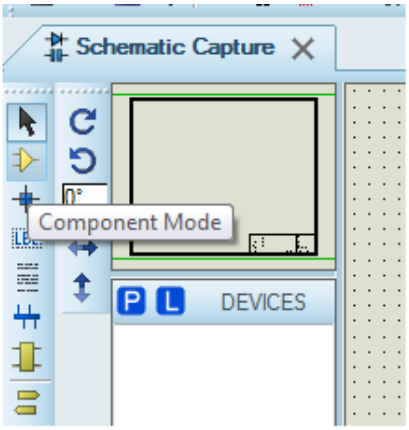
6. After finishing the settings, you will land on the empty workspace. Here, you can place components from the library.



The workspace where you can place components.

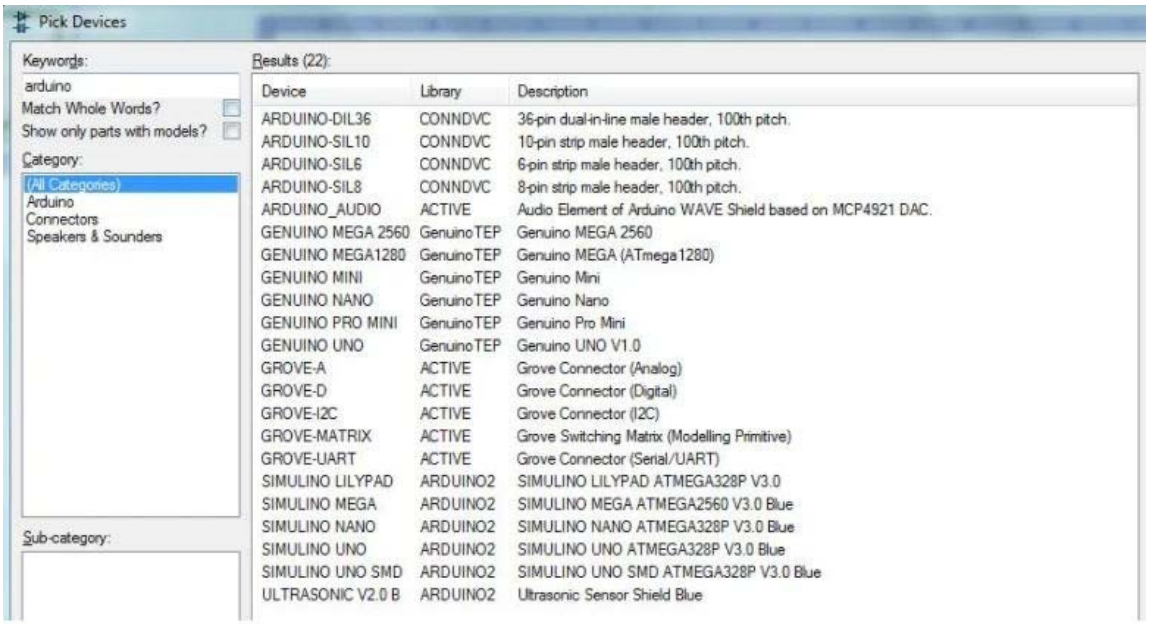
Getting Started in the Proteus Workspace

Select the op-amp symbol, which changes the mode to component mode. Then click P, which will cause a list of components to pop up. Here, you can find all types of components and footprints for simulation.



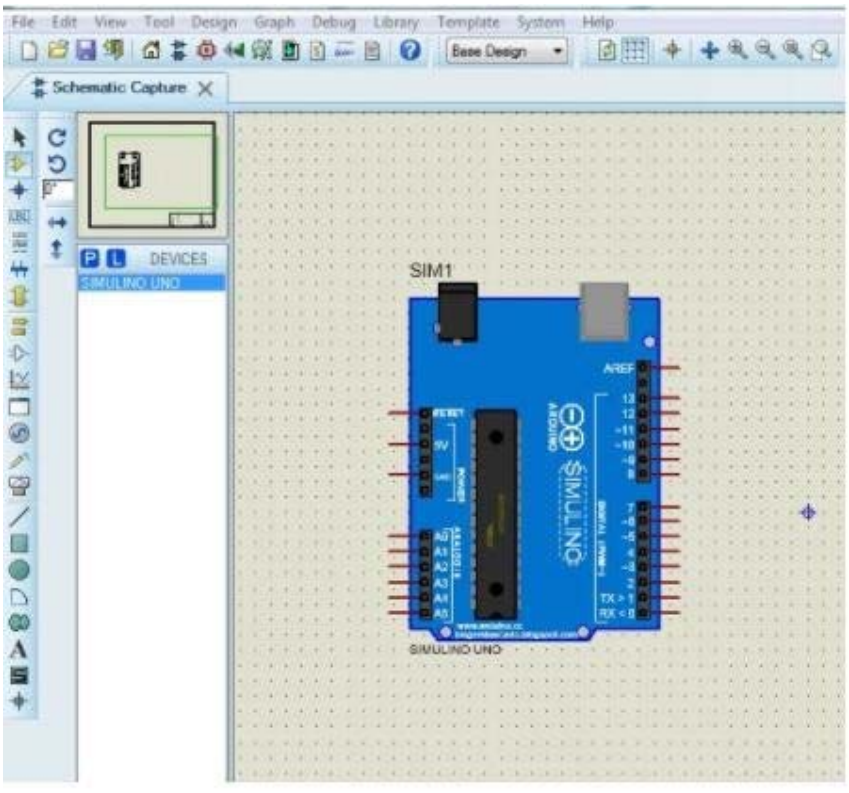
Set your mode to component mode.

Search for Arduino in the component list. All Arduino related boards and connector libraries will display. If none of the files display on your machine, you can repeat re-installing the Arduino library.



Select Arduino UNO, as you will be programming on this board in this example.

After selecting the library and clicking OK, click the spot on the workspace where you would like to drop the board.

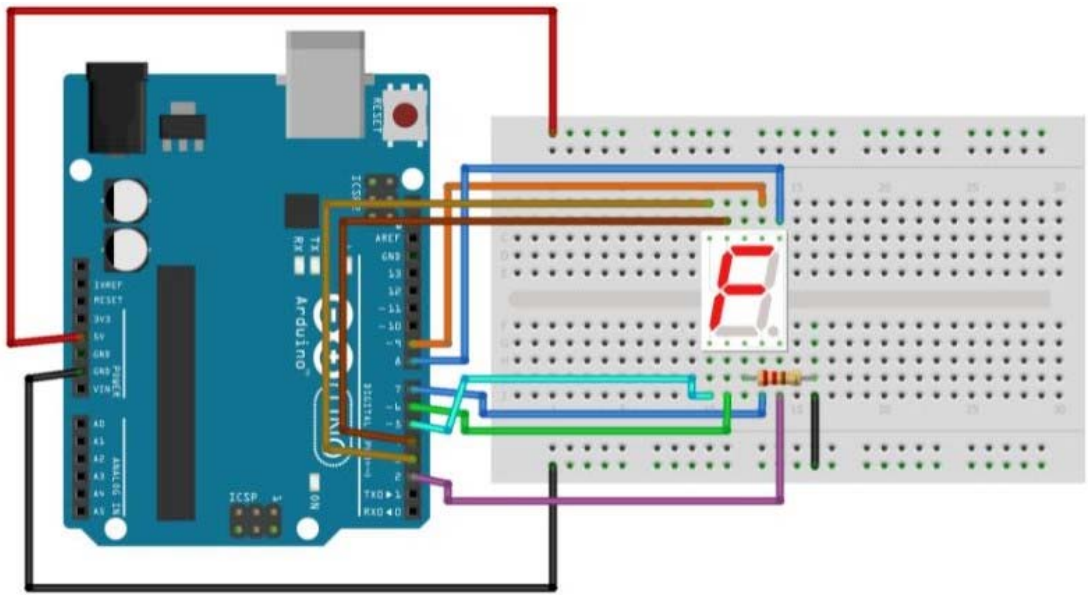


Place your Arduino by clicking the desired spot on your workspace.

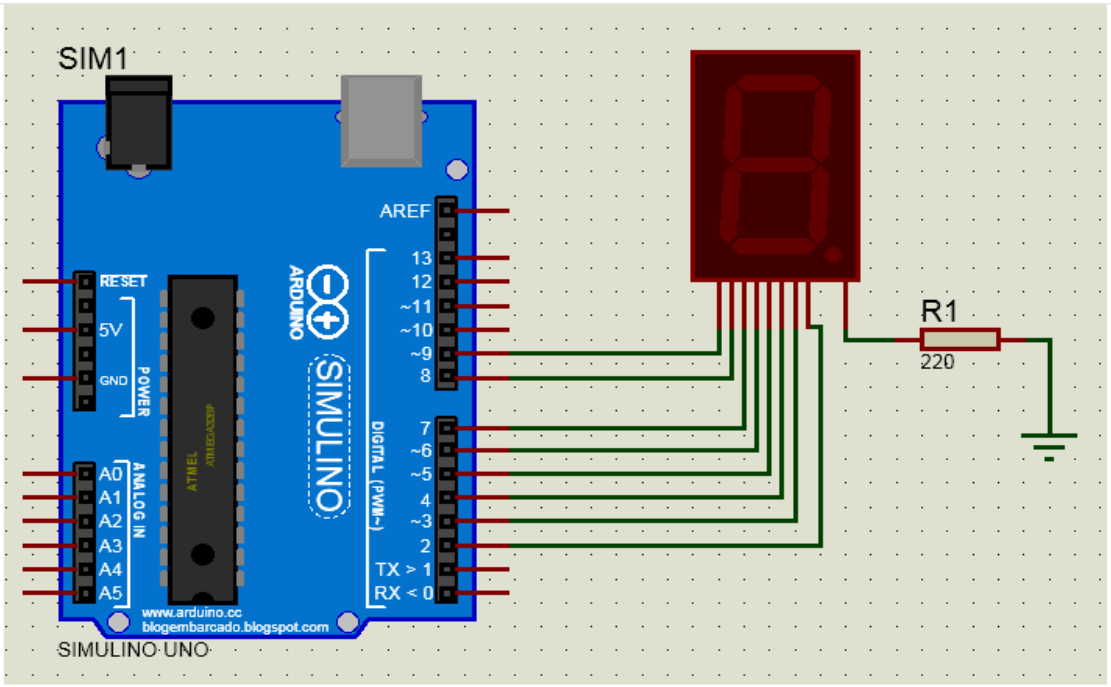
How to Simulate Projects

I'm going to show you how to simulate your Arduino projects with an example using a seven- segment up-counter and Arduino. The code and hardware will be presented in the next section, sofor time being, just focus on simulation.

First, mimic the wiring connections shown below in Proteus.

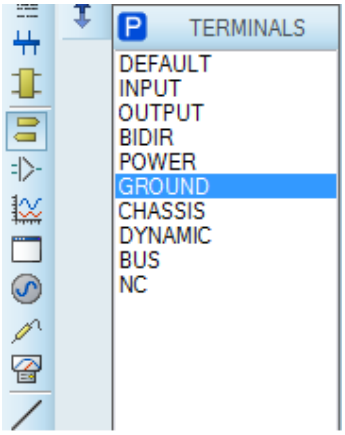


You can find all components from the list to complete hardware wiring. To connect any two components through a wire, left-click on one of the connector ends and drag the mouse towards another end to connect it.



Create connections by left clicking and dragging the connector path between components

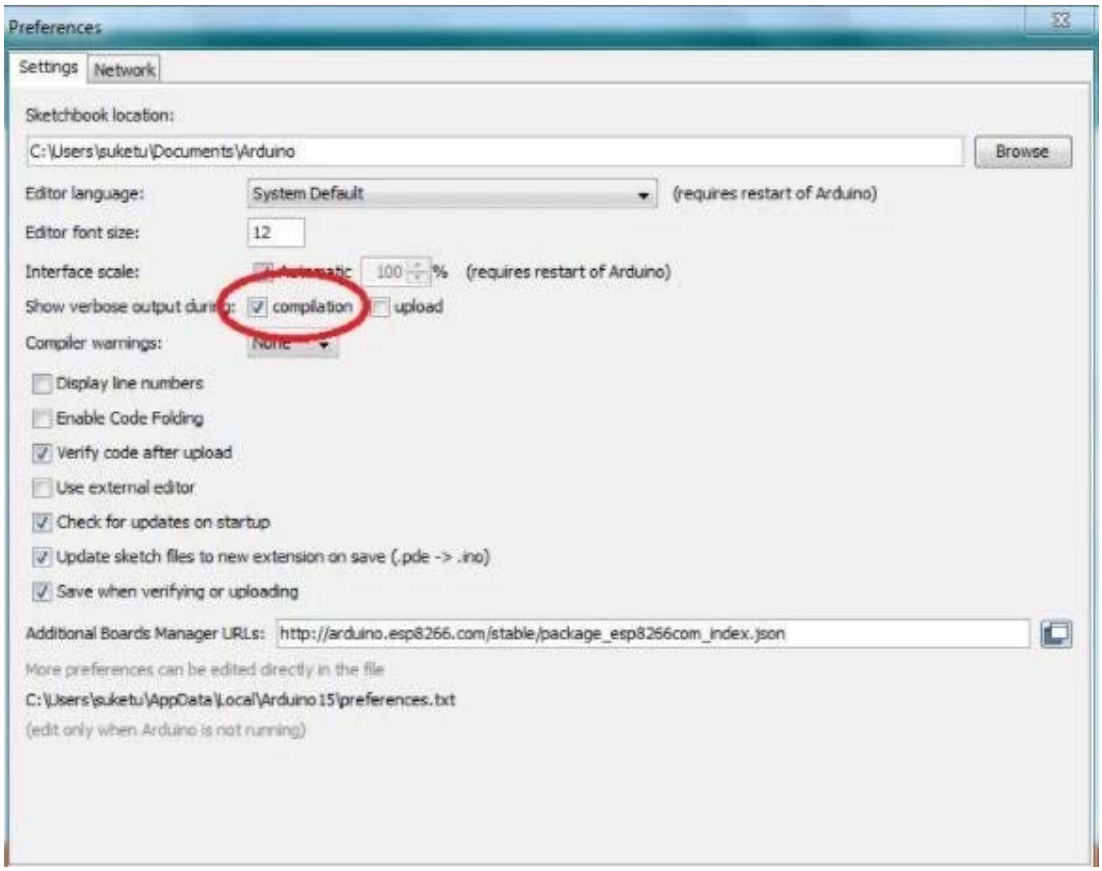
Power pins, such as Vcc and Gnd, can be found in Terminal Mode



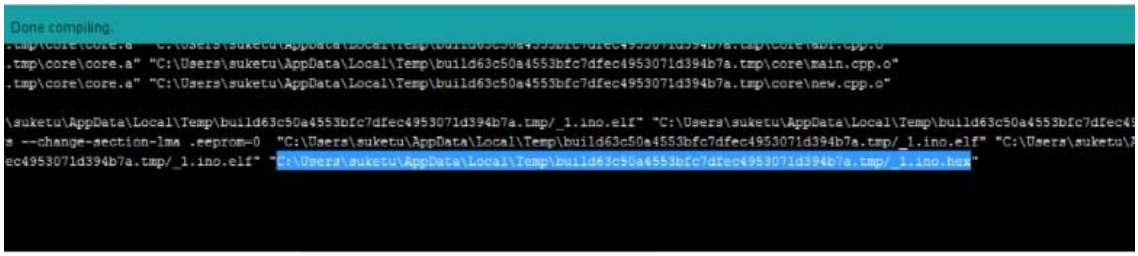
Locate power pins

Once you are done making the connections, upload the hex file of the code to your Arduino. To do this, open the code you downloaded from GitHub in Arduino IDE.

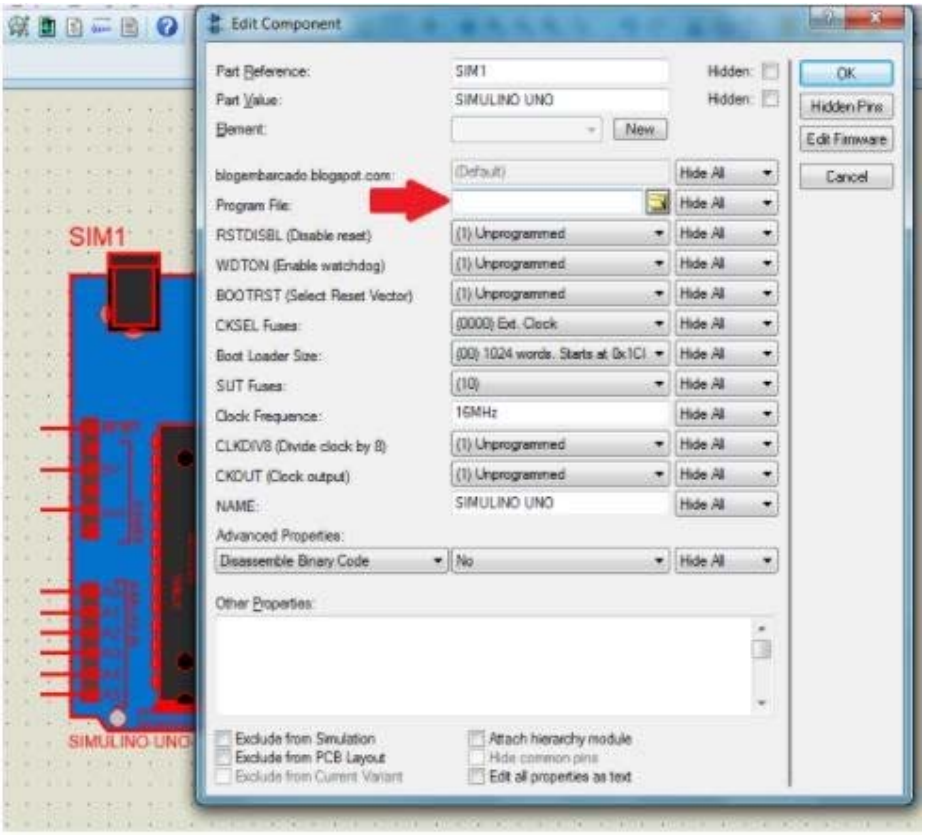
Check whether the compilation option is checked to generate the hex file. It can be found from File —> Preferences.



Compile the code and copy the hex file-path.

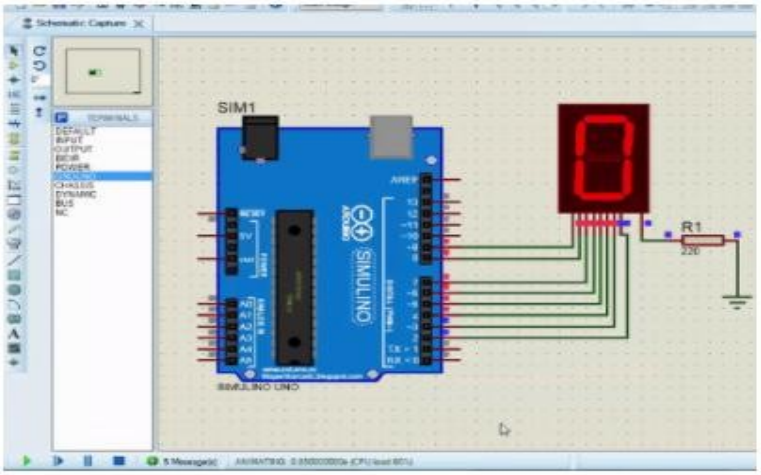


Double click on the Arduino board to insert the hex file of code.



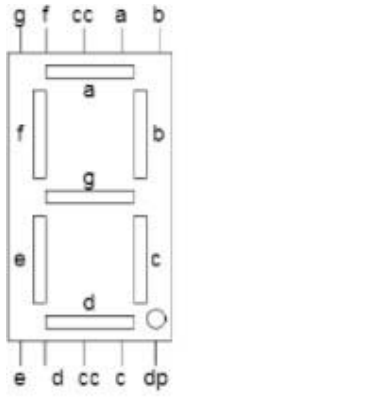
Paste hex file.

After inserting a hex file, you can start the simulation by pressing the play key.



Interfacing a Seven-segment Display to Perform Up-counting

As the name suggests, a seven-segment display is a combination of 7 LEDs which are used to display alphabets and numbers. It also has a small dot LED known as dp.



Seven-segment display pinout.

All LEDs in a seven-segment display are named alphabetically, making it very easy for a programmer to display a number or alphabet.

Here, connect all LEDs to the digital pins of the Arduino. A HIGH state from the digitalWrite function will turn ON an LED and vice versa. Below is a chart to display numbers using a common cathode seven-segment LED.

	a	b	c	d	e	f	g
0	ON	ON	ON	ON	ON	ON	
1		ON	ON				
2	ON	ON		ON	ON		ON
3	ON	ON	ON	ON			ON
4		ON	ON			ON	ON
5	ON		ON	ON		ON	ON
6	ON		ON	ON	ON	ON	ON
7	ON	ON	ON				
8	ON	ON	ON	ON	ON	ON	ON
9	ON	ON	ON			ON	ON

Note: Empty space = OFF

As far as programming is concerned, you can display a single-digit number and ascend it at fixed intervals. I have made a function to display a single-digit number.

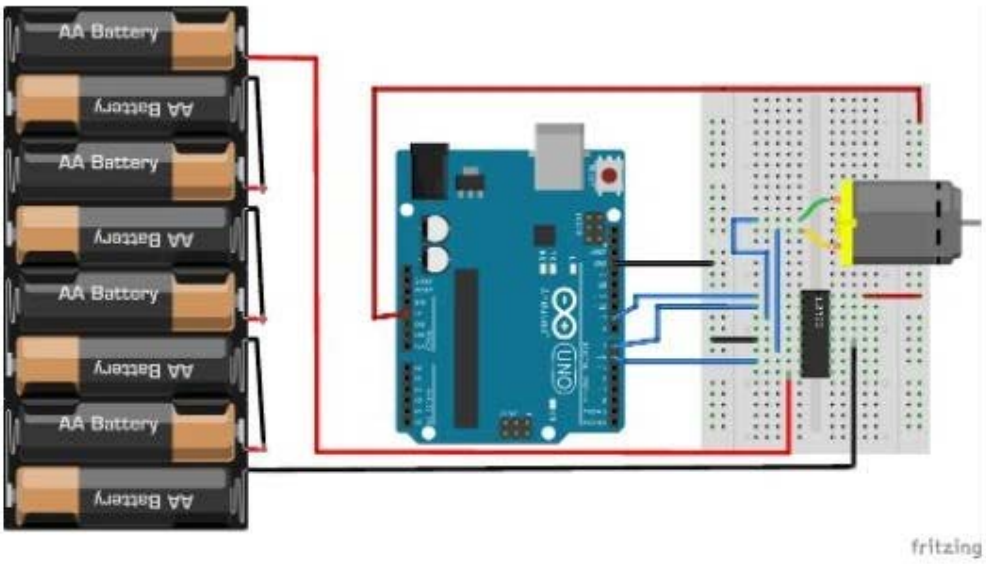
DisplayNumber(num);

Here is up-counter logic which will be repeated again and again, using a void loop().

```
void loop()
{
  for(i=0;i<=9;i++)
  {
    DisplayNumber(i);
    delay(1000);
  }
}
```

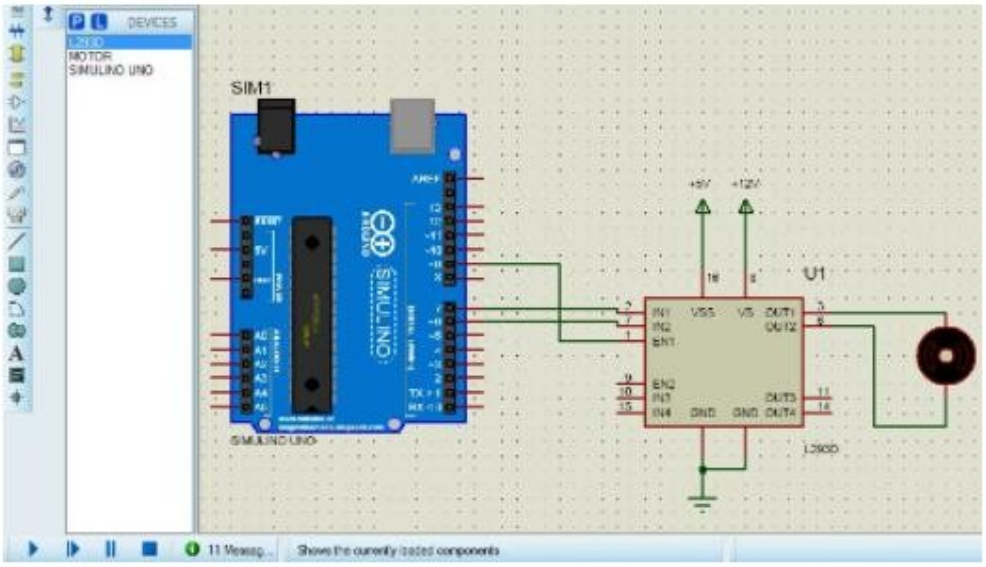
Interfacing a DC Motor With Arduino Using an L293D IC

This exercise shows you how to simulate a DC motor and observe PWM waves on the oscilloscope using Proteus. A PWM signal from the Arduino is required to change the DC motor's speed.



Four 3V batteries are connected in series to get 12V.

First, make the hardware connections on Proteus referencing the above diagram.

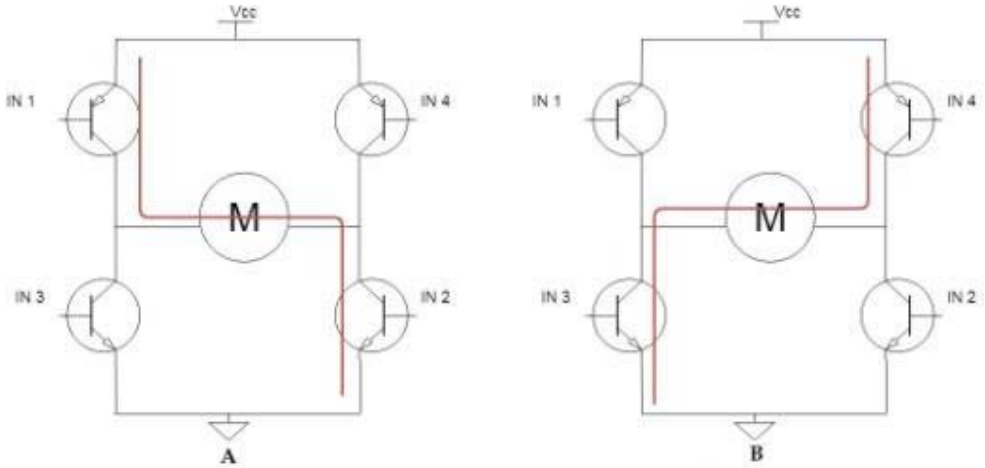


What is an L293D IC Why is it Used?

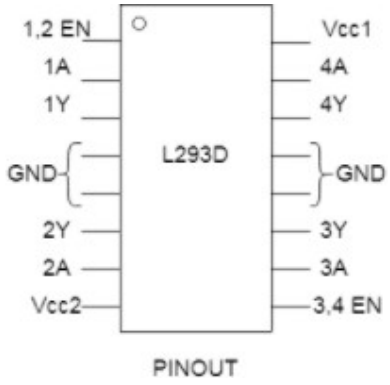
Arduino is not sufficient to drive the DC motor directly as the motor consumes more current. Arduino can source 40mA (max) from its GPIOs and a DC motor requires up to 200 - 300 mA. So, current amplification between the Arduino and the DC motor is required. That is where the L293D IC H- bridge driver comes in.

The main advantage of using an H-bridge is you only have to change the current direction to move the motor forward or backward rather than changing voltage polarity.

Two H-bridge diagrams are shown below, each with different current directions. In image A, turning the first and second transistor on causes the motor to turn clockwise. In image B, turning the third and fourth transistor on will cause it to turn counter-clockwise.



Here is the pinout of the IC



PINOUT

It requires two Vcc's: 5V (Vcc1) for its internal driver operation and 12V (Vcc2) for the motor. The L293D can drive two motors at a time and for each motor, it has two input pins (A) and two output pins (Y).

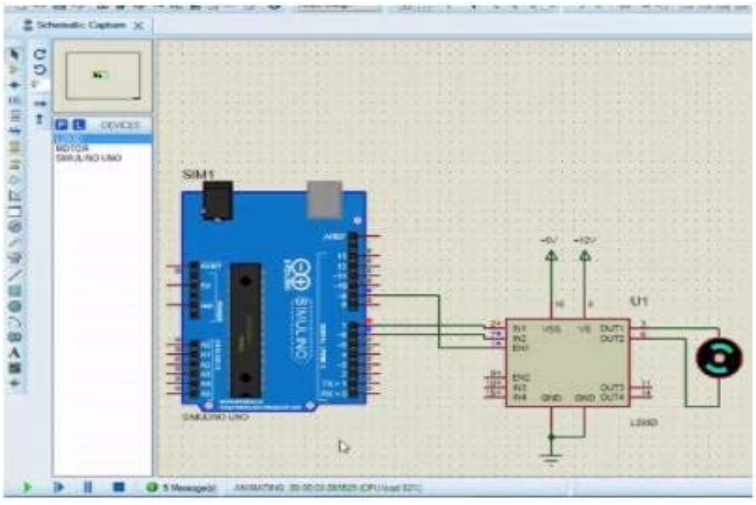
At one input pin, we have to pass a HIGH digital signal and at other a LOW signal. These signals will then be amplified and given to the motor. Basically what we have done is, we have just applied a positive signal on one pin of the motor and LOW signal to another pin of the motor. This will tend to move the motor continuously in a particular direction at maximum speed. But for assigning speed, we will source PWM pulses from Arduino to enable the L293D's pin.

Programming the Arduino and DC Motor

As far as programming is concerned, you need to set a motor to move in a particular direction and assign it a constant speed. The code is pretty straightforward.

```
digitalWrite(7,HIGH); //Move motor in clockwise direction

digitalWrite(6,LOW);

analogWrite(9,100); //speed
```

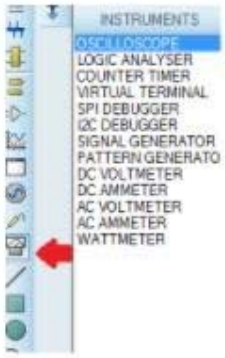
Upload the hex file of this code to the Arduino board and watch the motor rotate in the simulation.



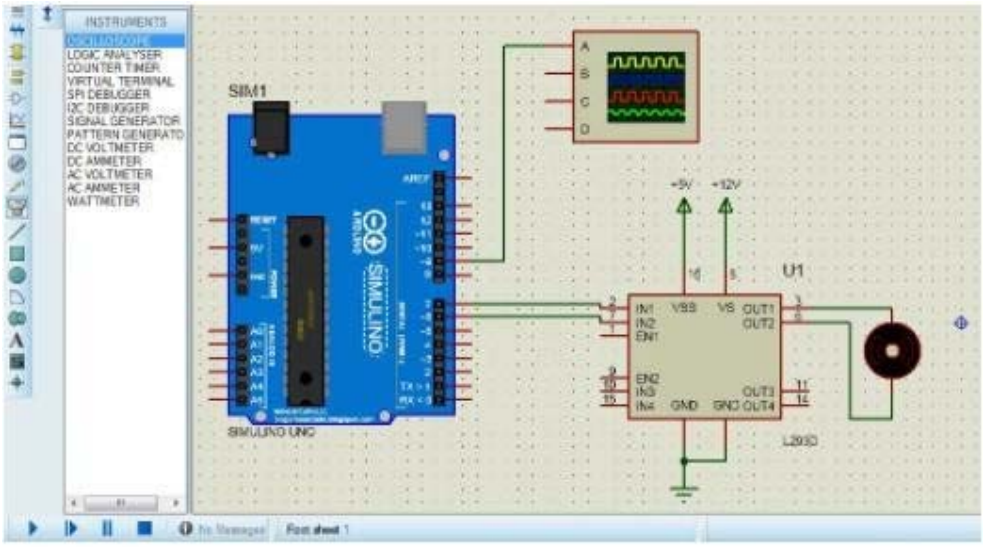
The motor moving in the Proteus simulation.

Try to change the PWM's duty cycle and observe a change in the speed of the motor.

Next, you can set up PWM wave observation on the oscilloscope in Proteus. The oscilloscope can be found in the Instruments tab.

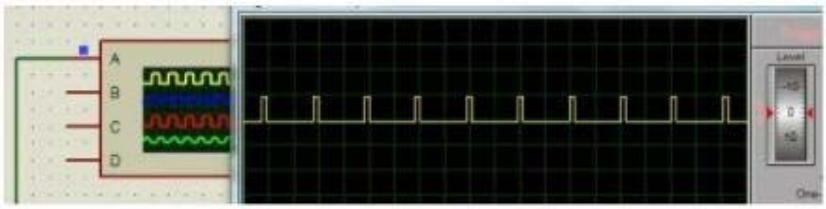


Locate the oscilloscope in the Instruments tab.

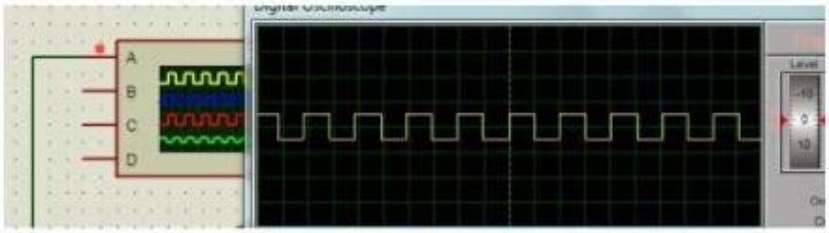


Connect the PWM pin to the oscilloscope.

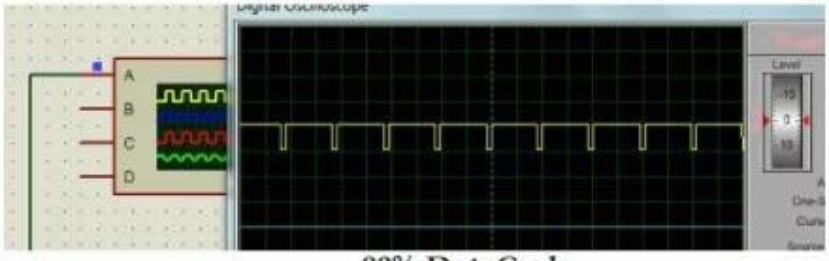
Press the Play tab to observe PWM waves. The images below show PWM waves of 10%, 50%, and 90% duty-cycle.



10% DutyCycle



50% DutyCycle



90% DutyCycle

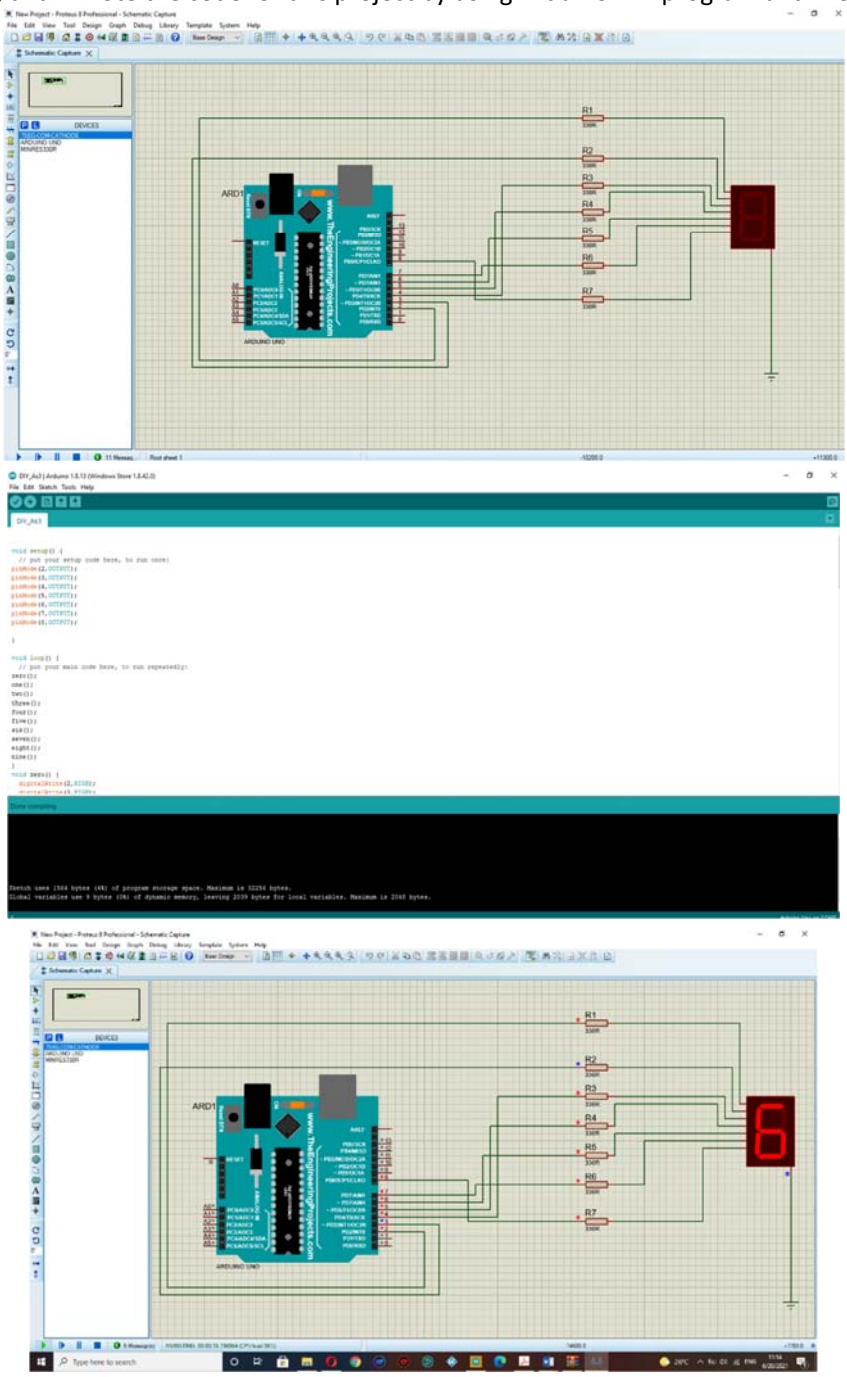
Proteus is Great for Simulation

In this article, I showed you how to use Proteus for simulation. You installed the Arduino library in Proteus and performed two exercises: seven-segment LCD display and DC motor interfacing with Arduino.

I hope you enjoyed learning about Proteus. If you have any difficulty while using it, feel free to post about it in the comment section.

3. Realization Of A Counter With A Single Digit Seven Segment Display

In this project, I used an Arduino device to create a counter showing the numbers from 0 to 9 using an 7 segment screen, and I wrote the code for this project by using Arduino DIA program and Proteus.



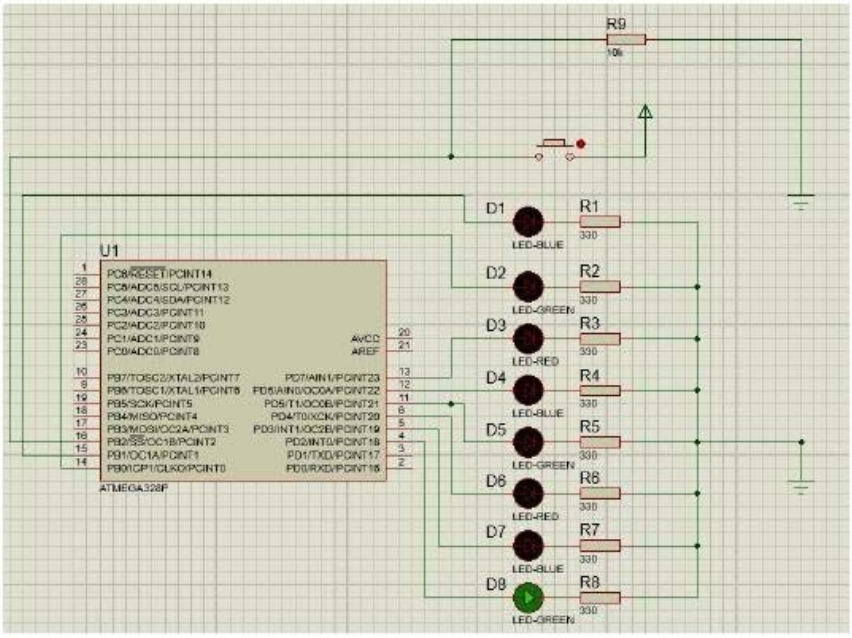
And i wrote a arduino code which makes turn 8 LEDs on one by one and when all LEDs it is turning back. Then if you press switch button it checks debounce then starts to turn LEDs on in reverse order of the previous order.

Arduino Codes

```
int buttonState = 0;

void setup()
{
    pinMode(2, OUTPUT);
    pinMode(3, OUTPUT);
    pinMode(4, OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(6, OUTPUT);
    pinMode(7, OUTPUT);
    pinMode(8, OUTPUT);
    pinMode(9, OUTPUT);
}
void reverseorder(){
  digitalWrite(2,LOW);
  digitalWrite(3,LOW);
  digitalWrite(4,LOW);
  digitalWrite(5,LOW);
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);
  digitalWrite(8,LOW);
  digitalWrite(9,LOW);

  digitalWrite(2, HIGH);
  delay(50);
  digitalWrite(3, HIGH);
  delay(50);
  digitalWrite(4, HIGH);
  delay(50);
  digitalWrite(5, HIGH);
  delay(50);
  digitalWrite(6, HIGH);
  delay(50);
  digitalWrite(7, HIGH);
  delay(50);
  digitalWrite(8, HIGH);
  delay(50);
  digitalWrite(9, HIGH);
  delay(50);

}
```

```
void order(){
  digitalWrite(2,LOW);
  digitalWrite(3,LOW);
  digitalWrite(4,LOW);
  digitalWrite(5,LOW);
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);
  digitalWrite(8,LOW);
  digitalWrite(9,LOW);

  digitalWrite(9, HIGH);
  delay(50);
  digitalWrite(8, HIGH);
  delay(50);
  digitalWrite(7, HIGH);
  delay(50);
  digitalWrite(6, HIGH);
  delay(50);
  digitalWrite(5, HIGH);
  delay(50);
  digitalWrite(4, HIGH);
  delay(50);
  digitalWrite(3, HIGH);
  delay(50);
  digitalWrite(2, HIGH);
  delay(50);

}
int counter = 0;
void loop() {
  buttonState = digitalRead(10);
  if (buttonState == HIGH) {
    order();
  } else {
    counter++;
    delay(10);
    if(counter>=3){
      reverseorder();
      counter=0;
    }
  }
}
```

Circuit Design



4. Two Arduino Uno Talk To Each Other

In this experiment I designed a circuit which have two arduino uno. One arduino uno have a pin connected a button and if you press the button other arduino turn the LED on.

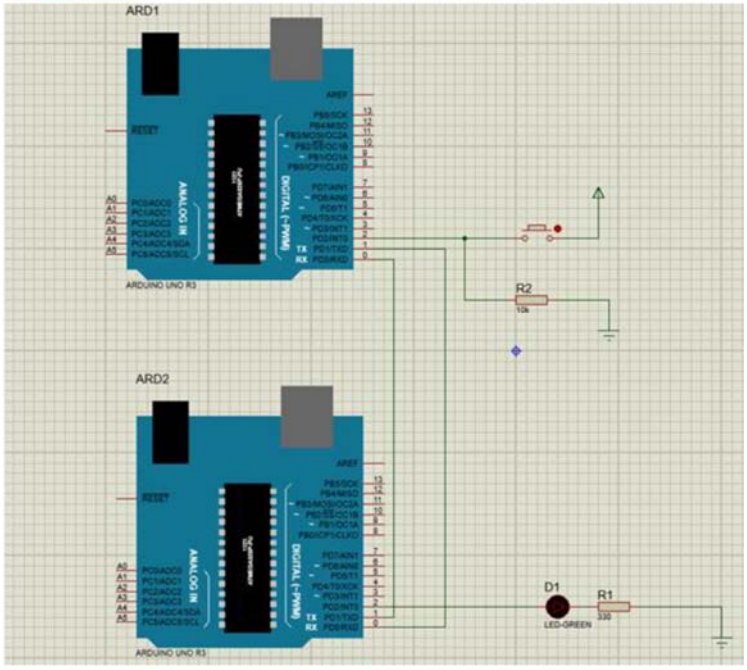
Arduino Codes 1. Arduino

```
#define led 2
int val;
int pos = 0;
void setup(){
  Serial.begin(9600);
  pinMode(led,OUTPUT);
}
void loop(){
  val = Serial.read();
  if (val == 100 && pos == 0){
    digitalWrite(led,HIGH);
    pos = 1;
  }
  else if(val == 100 && pos == 1){
    digitalWrite(led,LOW);
    pos = 0;
  }
}
```

Arduino Codes 2. Arduino

```
int val;
#define button 2
void setup() {
  Serial.begin(9600);
  pinMode(button,INPUT);
}

void loop() {
  val = digitalRead(button);
  if(val == HIGH){
    Serial.write(100);
    delay(250);
  }
}
```

Circuit Design



You can find video at this link :

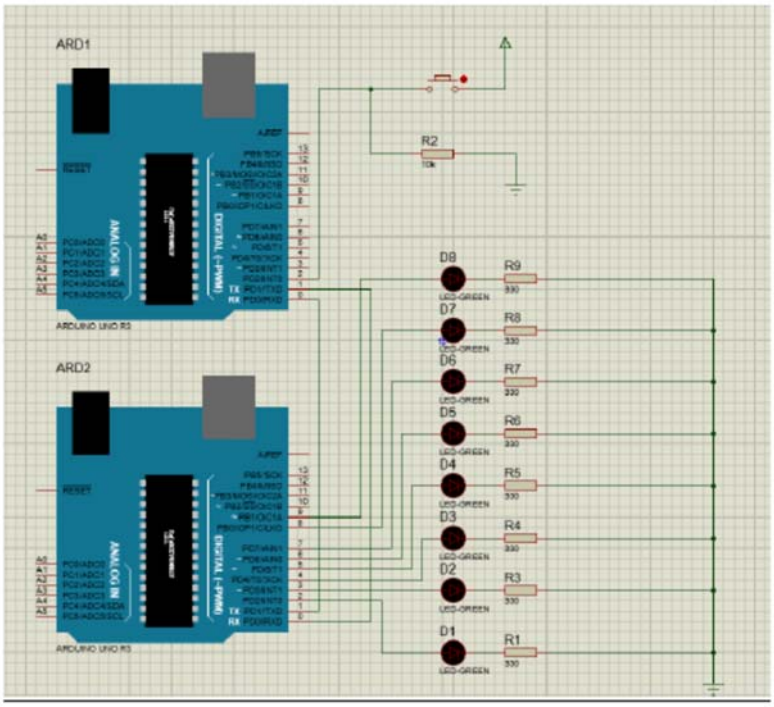
https://youtu.be/AMy_ZnsLnOA

5. Make Two Arduino Uno Talk To Each Other and Use Timer Interrupt

In this experiment, as I understood from the sheet, it was expected that one arduino card would send hardware interrupt signal to another, the time between arduino, which received the interrupt signal, sent the other between two interrupt signals from the serial port, and the arduino, which took the time difference from the serial port design circuit with two arduinos and when you push buton which connects with first arduino second arduino starts to make LEDs on one by one and when a LED blinks interrupt timer starts and second arduino starts to print 'A' to serial screen and when it finishes printing 'A's it blinks another LED.

To achieve this, I simply connected the serial pins of the 2 ardunio cards together.As I read in the manual of the Arduino uno board, hardware interrupt pins were 2 and 3 pins. I chose the 2nd pin and connected the 2nd pins of the 2 cards together.I changed the data transferred from the serial port between 2 cards 1 in 5 seconds to perform the experiment. When the data changes, the other card sends the LOW value from the second pin to the card where the time difference is kept, and thus the interrupt function is triggered. The interrupt function sends the time between the other interrupt function from the serial port to the other card, and the card that receives this data prints it on the serial monitor. In order to provide the necessary simulation

Circuit Design



Arduino Codes 1

```
int val;
#define button 2
void setup() {
  Serial.begin(9600);
  pinMode(button,INPUT);
}

void loop() {
  val = digitalRead(button);
  if(val == HIGH){
    Serial.write(100);
    delay(250);
  }
}
```

Arduino Codes 2

```
int val;
int pos = 0;
const int LEDarray[] = {2,3,4,5,6,7,8,9};
int j = 0;
void setup(){
  Serial.begin(9600);
  for(int i=0; i<8 ;i++)
  {
    pinMode(LEDarray[i], OUTPUT);
  }
  cli();
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;
  OCR1A = 15624;
  TCCR1B |= (1 << WGM12);
  TCCR1B |= (1 << CS12) | (1 << CS10);
  TIMSK1 |= (1 << OCIE1A);
  sei();
}
ISR(TIMER1_COMPA_vect){
  for(int i=0; i<8; i++){
    digitalWrite(LEDarray[i],LOW);
  }
  digitalWrite(LEDarray[j],HIGH);
  j ++;
  if(j > 7)
    j = 0;
}
```

```
void loop(){
  val = Serial.read();
  if (val == 100 && pos == 0){
    Serial.println("A");
    pos = 1;
  }
  else if(val == 100 && pos == 1){
    digitalWrite(2,LOW);
    pos = 0;
  }
}
```

In this experiment I design circuit with two arduinos and when you push button which connects with first arduino second arduino starts to make LEDs on one by one and when a LED blinks interrupt timer starts and second arduino starts to print ‘A’ to serial screen and when it finishes printing ‘A’s it blinks another LED.

6. Serial Communication via PWM

What is the PWM ? (Pulse Width Modulation)

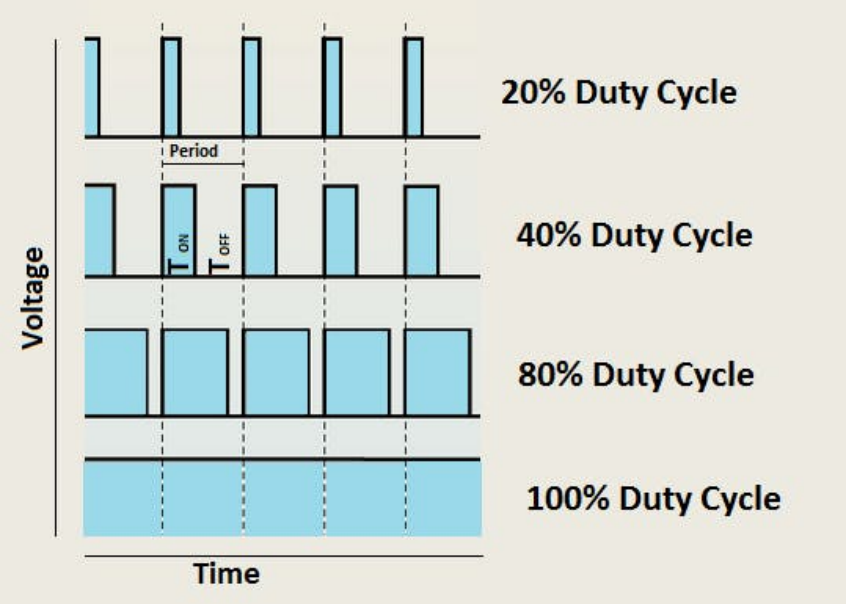
Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between the full Vcc of the board (e.g., 5 V on Uno, 3.3 V on a MKR board) 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. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and Vcc controlling the brightness of the LED.

The Arduino digital pins either gives us 5V (when turned HIGH) or 0V (when turned LOW) and the output is a square wave signal. So if we want to dim a LED, we cannot get the voltage between 0 and 5V from the digital pin but we can change the ON and OFF time of the signal. If we will change the ON and OFF time fast enough then the brightness of the led will be changed.

Before going further, let's discuss some terms associated with PWM.

TON (On Time): It is the time when the signal is high.
TOFF (Off Time): It is the time when the signal is low.
Period: It is the sum of on time and off time.
Duty Cycle: It is the percentage of time when the signal was high during the time of period.

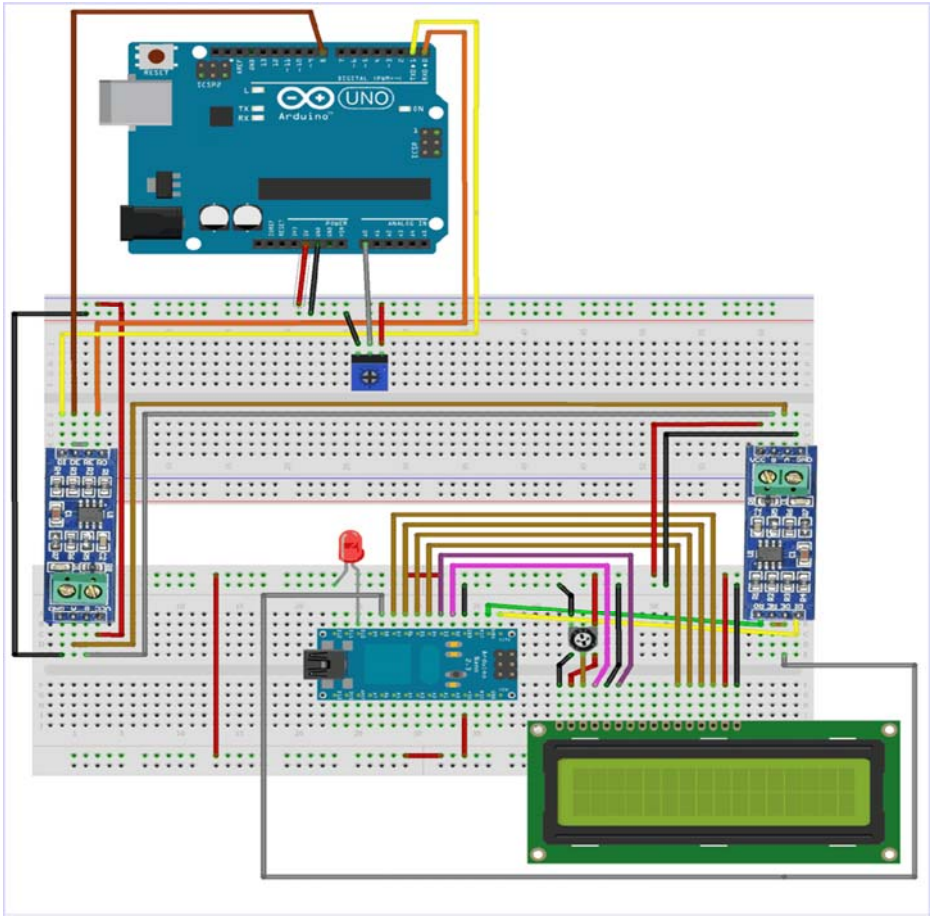
So at 50% duty cycle and 1Hz frequency, the led will be high for half a second and will be low for the other half second. If we increase the frequency to 50Hz (50 times ON and OFF per second), then the led will be seen glowing at half brightness by the human eye.



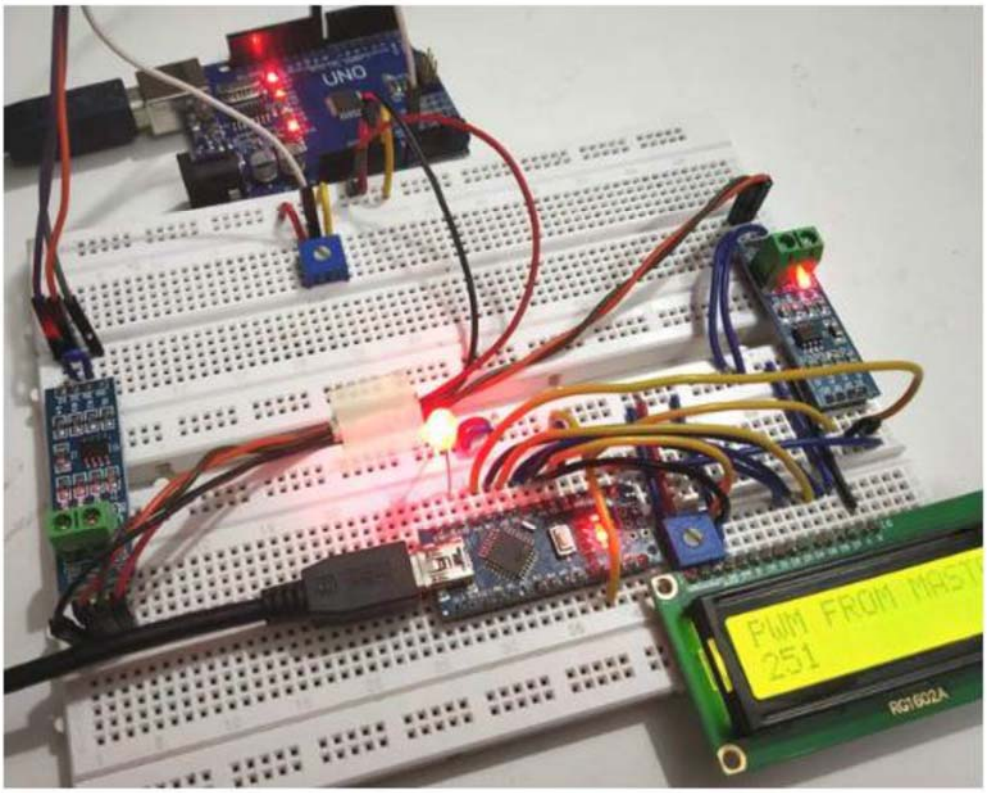
Advantages of PWM Technique

First of all, the switching loss is low. While practically no current flows when the switch is closed, the voltage drop across the switch is negligible when the switch is open.
In this way, besides the low power loss, the PWM technique is in great harmony with digital control units. In this way, the system can adapt to on-off switching changes much more easily.
In addition, PWM technique is frequently used in regulations regarding the duty cycles of signals used in communication and communication technologies. Thus, the desired signals can be obtained with various adjustments on the channel.

Circuit Diagram



And when PWM value is set at 251 using potentiometer: The LED is turned ON with full brightness as shown in below picture:



So this is how **RS485 can be used for serial communication in Arduino**.

Code Explanation

In the Master side, just simply take Analog input at pin A0 by varying the potentiometer and then Serial Write those values to the RS-485 bus through the Hardware Serial Ports (0,1) of Arduino UNO.

To Begin Serial Communication at Hardware Serial Pins (0,1) use:

```
Serial.begin(9600);
```

Before writing the *potval* value to serial port, pins DE & RE of RS-485 should be **HIGH** that is connected to the pin 8 of Arduino UNO so to Make pin 8 HIGH:

```
digitalWrite(enablePin, HIGH);
```

Next to put those values in the Serial Port connected with the RS-485 module, use the following statement

```
Serial.println(potval);
```

In the Slave side an integer value is received from the Master RS-485 that is available at the Hardware Serial port of the Arduino Nano (Pins -0,1). Simply read those value and store in a variable. The values are in form of (0 -1023). So it is converted into (0-255) as PWM technique is used to control LED brightness.

Then *AnalogWrite* those converted value to the LED pin D10 (It's a PWM pin). So depending upon the PWM value the brightness of the LED changes and also display those values in 16x2 LCD display.

In order for the Slave Arduino's RS-485 to receive the values from the Master, just make the pins DE & RE of RS-485 **LOW**. So the pin D8 (enablePin) of Arduino NANO is made LOW.

```
digitalWrite(enablePin, LOW);
```

And to read the integer data available at Serial Port and store them in a variable use

```
int pwmval = Serial.parseInt();
```

Next convert value from (0-1023 to 0-255) and store them in a variable:

```
int convert = map(pwmval,0,1023,0,255);
```

Next write the analog value (PWM) to pin D10 where LED anode is connected:

```
analogWrite(ledpin,convert);
```

To print those PWM value in 16x2 LCD display use

```
lcd.setCursor(0,0);

lcd.print("PWM FROM MASTER");

lcd.setCursor(0,1);

lcd.print(convert);
```

Code

Master (Arduino UNO):

//Master code (Arduino UNO)

```
//Serial Communication Between Arduino using RS-485
int enablePin = 8;
int pushval = A0;
int potval =0 ;
void setup()
{
Serial.begin(9600); // initialize serial at baudrate 9600:
pinMode(enablePin, OUTPUT);
pinMode(pushval,INPUT);
delay(10);
digitalWrite(enablePin, HIGH); // (always high as Master Writes data to Slave)
}
void loop()
{
int potval = analogRead(pushval); 24

Serial.println(potval); //Serial Write POTval to RS-485 Bus
delay(100);

}
```

SLAVE CODE: (Arduino NANO):

//Slave code (Arduino NANO)

```
//Serial Communication Between Two Arduinos using RS-485
//Circuit Digest
#include <LiquidCrystal.h> //Include LCD library for using LCD display functions
int enablePin = 8;
int ledpin = 10;
LiquidCrystal lcd(2,3,4,5,6,7); // Define LCD display pins RS,E,D4,D5,D6,D7
void setup()
{
lcd.begin(16,2);
lcd.print("CIRCUIT DIGEST");
lcd.setCursor(0,1);
lcd.print("RS485 ARDUINO");
delay(3000);
lcd.clear();
Serial.begin(9600); // initialize serial at baudrate 9600:
pinMode(ledpin,OUTPUT);
pinMode(enablePin, OUTPUT);
delay(10);
digitalWrite(enablePin, LOW); // (Pin 8 always LOW to receive value from Master)
}
void loop()
{
while (Serial.available()) //While have data at Serial port this loop executes
{
lcd.clear();
int pwmval = Serial.parseInt(); //Receive INTEGER value from Master throught RS-485
int convert = map(pwmval,0,1023,0,255); //Map those value from (0-1023) to (0-255)
```

```
analogWrite(ledpin,convert); //PWM write to LED
lcd.setCursor(0,0);
lcd.print("PWM FROM MASTER");
lcd.setCursor(0,1);
lcd.print(convert); //Displays the PWM value
delay(100);
}
}
```

If both RS-232 and PWM tasks are to be handled simultaneously, then they must operate in a mutually exclusive real-time interrupt driven environment. Interrupts are a means to transfer control from one part of a program to another, provided an external trigger or timer-based event occurred. After an interrupt completes control is transferred from the interrupt back to the program or any new interrupts that may have occurred during the servicing of the current interrupt. Both PWM and RS-232 generate pulses many times per second. More specifically, RS-232 generates 10 bits for every byte because of the start bit, 8-bits of data, and stop bit. Furthermore, an RS-232 signal is time dependent and will result in corrupt data if not read at exactly the correct time intervals. For example, a 9600 baud RS-232 link streams 9,600 bits/s, which translates to a required pulse sampling every 104.16 µs. However, if the RS-232 connection is buffered then the chip is storing the bytes in a hardware buffer until they are read. Most 8-bit PIC chips have a 2-byte universal synchronous asynchronous receiver transmitter (USART) hardware buffer. Therefore, it is possible to read the buffer once every 1.04 ms (1 byte [9600/10 bits per byte] or every 2.08 ms (2 bytes [9600/10 bits per byte]) before any data loss. The USART hardware buffer greatly simplifies support for RS-232 devices because they allow for an order of magnitude more clock cycles to occur between each servicing of the interrupt. While 2.08 ms doesn't sound like a great deal of time, there are actually 20,833 (2.08 ms * 10 million instructions per second [MIPS]) instructions that can be processed during that time. This interval leaves plenty of time for the PWM interrupt to operate between each byte transferred over the RS-232 link.

7. There are analog sensors available that measure temperature, light intensity, distance, position, and force.

1. In this experiment, you are asked to use the ADC inputs of Arduino. What resolution or featurescan you convert ? Give it in a table. Do a short research on what can be done with analog inputs.Carry out a suitable (material and attention-related) use as an example of experimental work. For example, you can use it as a unit that measures your AAA type batteries.

- The analog to digital converter (ADC - analog to digital converter) with 10-bit resolution is available in the processor on our Arduino UNO board. 10-bit mean is that our Arduino's microcontroller operates with 5V voltage. The 10-bit ADC, which we say has this microcontroller, can read voltages between 0V and 5V with a precision of 210 = 1024 steps. In other words, the 0Vvoltage we will give to one of the analog input pins gives us the value of 0; Likewise, 5V voltage corresponds to 1023. If we need to take a voltage lower than 5V as the top of our reference, we needto use the AREF pin on the Arduino and make a small change in the code. If we do not apply any voltage to this pin, our ADC will operate between 0-5V. We use ADC to read values from sensors. Because sensors give us analog type data. For example we plug a LM35 sensor to arduino analog pins and ADC calculate temperature with that Formula ADC Value = x * 1024 / reference voltage (+5V).

How the Arduino ADC works

This ADC is known as a successive approximation ADC and requires several clock cycles to zoom in on the correct ADC output.

The ADC converter compares the input analogue voltage to a portion of the Vref voltage using a divide by two sequence. The sample and hold capacitor is charged to the input voltage and then the input disconnected so that the same voltage is measured throughout the conversion process.

It first checks whether the input voltage is higher or lower than half of the Vref voltage, by using a DAC to generate half the reference voltage. The DAC voltage is the fed into a comparator.

The output of the DAC forms the high bit of the result (stored in a shift register). If the input voltage is higher then the bit is one, otherwise the bit zero.

If the input is lower than half of the Vref voltage then control logic generates a DAC voltage that is 1/4 the reference voltage. The comparison is made again and this forms the next bit in the ADC output.

The process continues until all the bits are collected.

ADC clock

For the Arudino the conversion process takes 13 cycles of the ADC clock - which you set using a prescaler in the ADC module. The ADC clock must be between 50kHz and 200kHz so you choose the prescaler value to get a valid ADC clock.

The ADC clock prescaler can be set as a 2^n division from 2 to 128. You obviously want the fastest conversion rate for the clock in use so for a 16MHz system clock you would calculate 16e6/200e3 = 80 so the closest could be 64.

However 16e6/64 is 250kHz and is too big. Therefore choosing a divisor of 128 must be used so the ADC clock will be 16e6/128 = 125kHz.

A conversion will take 13 ADC clock cycles :

```
13 * 1.0/(125e3) = 104us
```

Check that these settings are used in the Arduino Source code! - I have not - they are extremely likely though.

Arduino Uno sampling rate (16MHz crystal)

```
1.0 / ( 13 * 1.0/125e3) = 9615Hz
```

Actually, reading the Arduino reference page it says the sample rate is about 10kHz so this calculation matches that information.

So the maximum Arduino ADC sampling rate is:

```
9.615kHz
```

The above rate is the maximum sampling rate but to reproduce a sinewave requires double the sampling rate (Nyquist theorem). Another way of looking at it is that the bandwidth of the signal you can safely reproduce is half the sampling rate.

However this is only dealing with sinewaves and not "real signals". You cannot reproduce a square wave at 4.8kHz because the edges of the signal are at a far higher frequency (Fourier analysis) - in fact you would end up with a 4.8kHz sine wave if trying to reproduce a 4.8kHz square wave by converting it with and ADC with a 9.615kHz ADC clock. This is why digital oscilloscopes have sample rates about 10 times higher than the maximum desired operational frequency.

Changing the Arduino Sampling Rate

ADC clock calculations

If you set the system clock to 20MHz you get 20e6/128 = 156250.0 - for a bit faster conversion.

Interestingly if you go the other way as a design decision you want the fastest ADC clock rate of 200kHz, then you have to ask the question:

"What crystal clock results in a 200kHz rate after ADC prescaling?" i.e.

Xtal = 200e3 * prescale - trying 64 gives 12800000 or 12.8Mhz

12.8e6/64 = 200e3

So reducing the Xtal clock allows a faster conversion rate of 200kHz!

Giving a max samping rate of:

1.0 / (13 * 1.0/200e3) = 15384Hz **(THIS IS FOR A 12.8MHz XTAL)**

...and yes you can get crystals made to your spec! - but you'll probably use a 12MHz crystal, as its easier to get, so the sample rate above will be a bit lower.

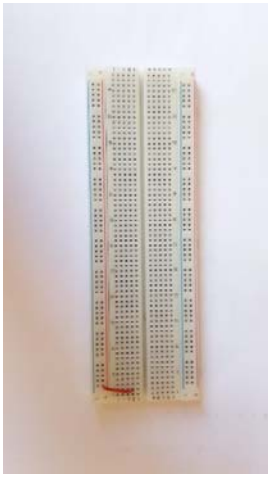
Project	Arduino	Sensor type	Jumper wires	USB-A to B Cable
temperature	yes	LM35 Temperature Sensor	yes	yes
Humidity	yes	DHT11	yes	yes
Smoke detection	yes	MQ-2 Smoke detection sensor	yes	yes
gas leak alert	yes	Analog LPG Gas Sensor (MQ5)	yes	yes
Light Detecting	yes	SparkFun PHOTOSENSTITIVE LIGHT SENSOR	yes	yes
Distance Measuring	yes	Proximity Sensor	yes	yes

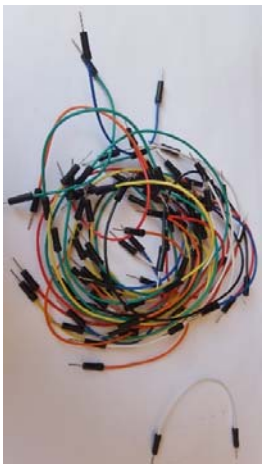
Any project where the input values are constantly changing we can use analog input for it.

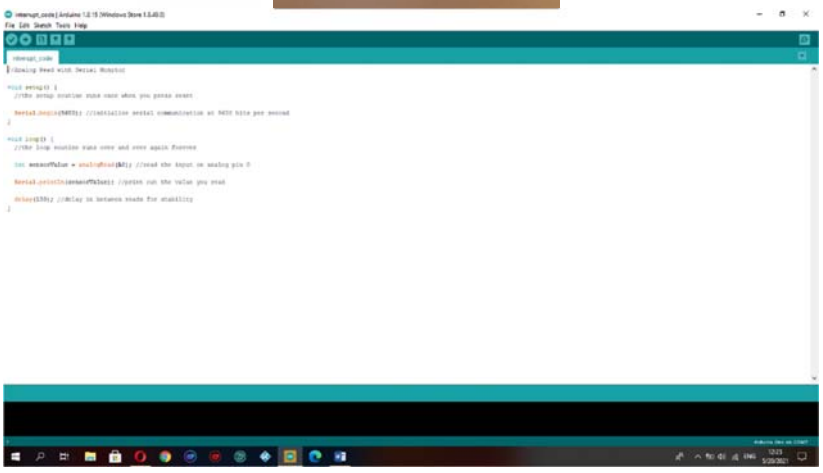
2. Perform a simple oscilloscope using analog inputs

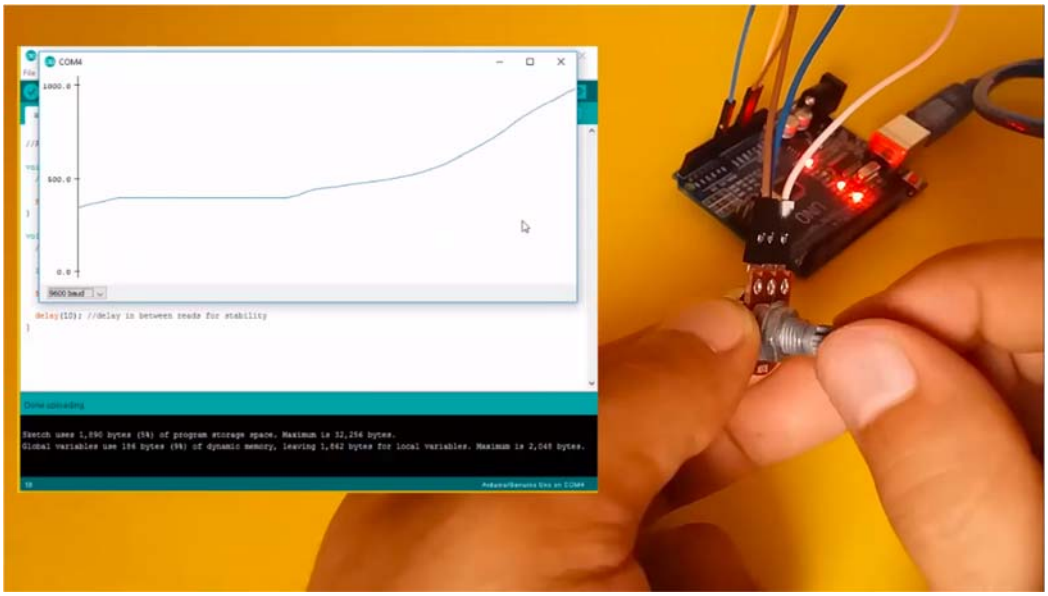
For this project we need :

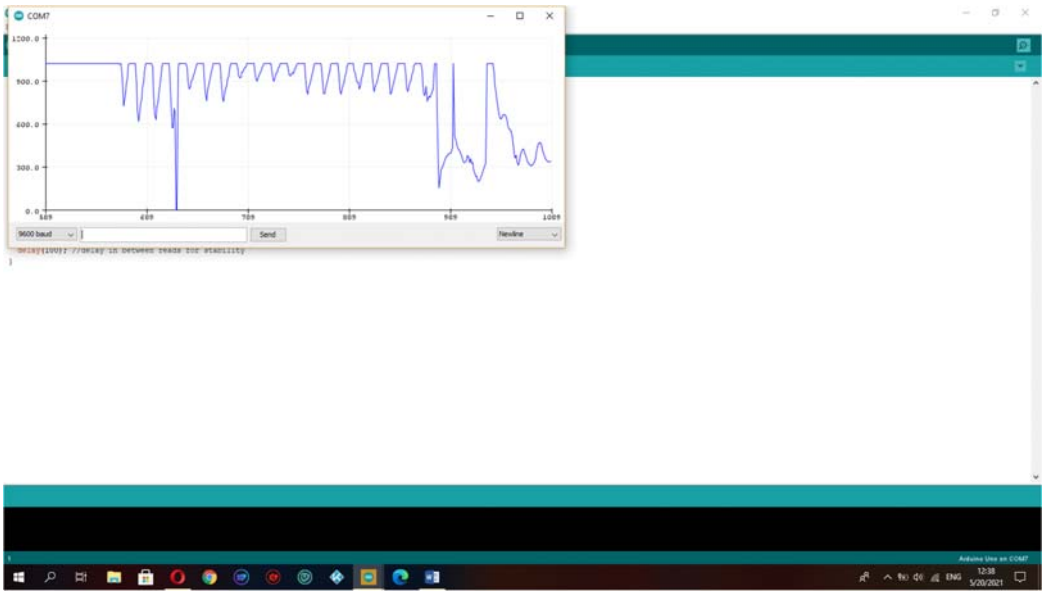
1. Arduino UNO
2. Potentiometer
3. USB cable
4. Jumper wires
5. Bored






COM4
1000.0
500.0
0.0
9600 baud
delay(10); //delay in between reads for stability
Done uploading.
Sketch uses 1,896 bytes (5%) of program storage space. Maximum is 32,256 bytes.
Global variables use 184 bytes (9%) of dynamic memory, leaving 1,842 bytes for local variables. Maximum is 2,048 bytes.


COM7
1200.0
900.0
600.0
300.0
0.0
400
700
800
900
1000
9600 baud
Send
Newline
Arduino Uno on COM7
ENG
12:38
5/20/2021


```
//Analog Read with Serial Monitor

void setup() {
  //the setup routine runs once when you press reset

  Serial.begin(9600); //initialize serial communication at 9600 bits per second
}

void loop() {
  //the loop routine runs over and over again forever

  int sensorValue = analogRead(A0); //read the input on analog pin 0

  Serial.println(sensorValue); //print out the value you read
```

```
  delay(100); //delay in between reads for stability
}
```

4. How you can transform the analog world through Ardunio ? Again, what kind of numerical values can be created regarding its features.

We can use potentiometers to observe analog input. Thanks to the potentiometer we connect the middle leg to an analog input, we can change the voltage we receive from the arduino and printit on the serial monitor. By turning the potentiometer, we ensure this change. And we can observe that the resolution of the potentiometer is 2 ^ 10.

Relating ADC Value to Voltage

The ADC reports a *ratiometric value*. This means that the ADC assumes 5V is 1023 and anything less than 5V will be a ratio between 5V and 1023.

$$\frac{Resolution\ of\ the\ ADC}{System\ Voltage} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured}$$

Analog to digital conversions are dependant on the system voltage. Because we predominantly use the 10-bit ADC of the Arduino on a 5V system, we can simplify this equation slightly:

$$\frac{1023}{5} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured}$$

If your system is 3.3V, you simply change 5V out with 3.3V in the equation. If your system is 3.3V and your ADC is reporting 512, what is the voltage measured? It is approximately 1.65V.

If the analog voltage is 2.12V what will the ADC report as a value?

$$\frac{1023}{5.00V} = \frac{x}{2.12V}$$

Rearrange things a bit and we get:

$$\frac{1023}{5.00V} * 2.12V = x$$

$$x = 434$$

The ADC should report 434