

Ahsanullah University of Science & Technology
Department of Electrical & Electronic Engineering

Laboratory Manual
FOR
ELECTRICAL AND ELECTRONIC SESSIONAL COURSE

Student Name:

Student ID:

Course No: **EEE 3110**
Course Title: **Microcontroller and Microprocessor Lab**
For the students of
Department of Electrical and Electronic Engineering
3rd Year 1st Semester

LAB NO: 0

LAB NAME: Introduction to Arduino Microcontroller and Arduino IDE.

Objective: The purpose of this laboratory is to introduce you to Arduino Microcontroller and the computer system that you will use in the remaining sessions.

Arduino

An Arduino is an open-source microcontroller development board. You can use the Arduino to read sensors and control things like motors and lights. This allows you to upload programs to this board which can then interact with things in the real world. With this, you can make devices which respond and react to the world at large. Uno is the most used and loved hardware board in the Arduino maker community. It is small, compact, cheap and can easily be interfaced with other devices.

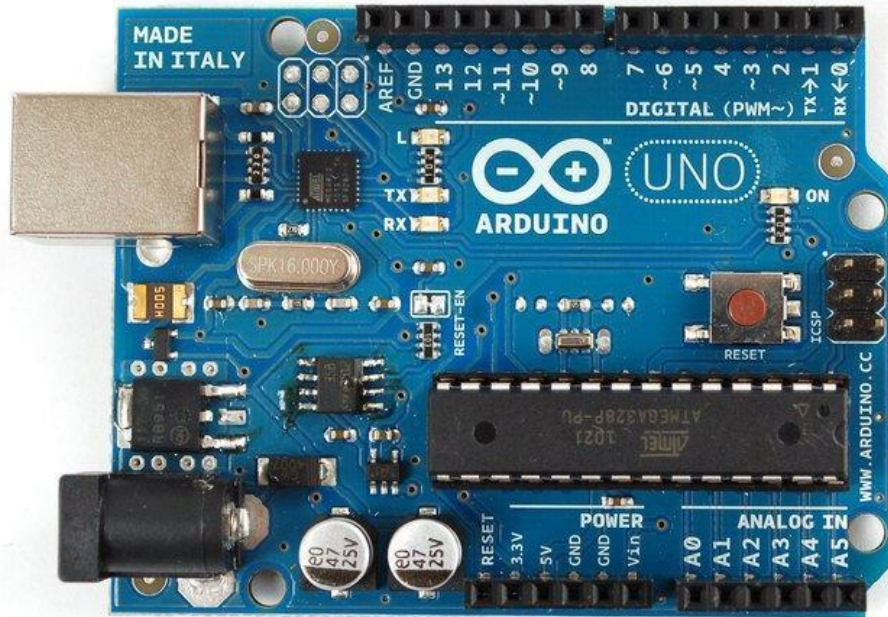


Fig: ATMEGA 328p Arduino UNO

Arduino Integrated Development Environment (IDE)

You use the Arduino IDE on your computer to create, open, and change sketches (Arduino calls programs “sketches”). You can download the Arduino IDE software from <https://www.arduino.cc/en/software>.

1. Setting up the Hardware

- Plug in your board via USB and wait for Windows to begin its driver installation process.
- Click on the Start Menu, and open up the Control Panel.
- While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.

- d. Look under Ports (COM & LPT). You should see an open port named "Arduino UNO (COMxx)".
- e. Right click on the "Arduino UNO (COMxx)" port and choose the "Update Driver Software" option.
- f. Next, choose the "Browse my computer for Driver software" option.
- g. Finally, navigate to and select the Uno's driver file, named "ArduinoUNO.inf", located in the "Drivers" folder of the Arduino Software download. Windows will finish up the driver installation from there.
- h. Double-click to open the Arduino application.

2. Parts of the IDE:

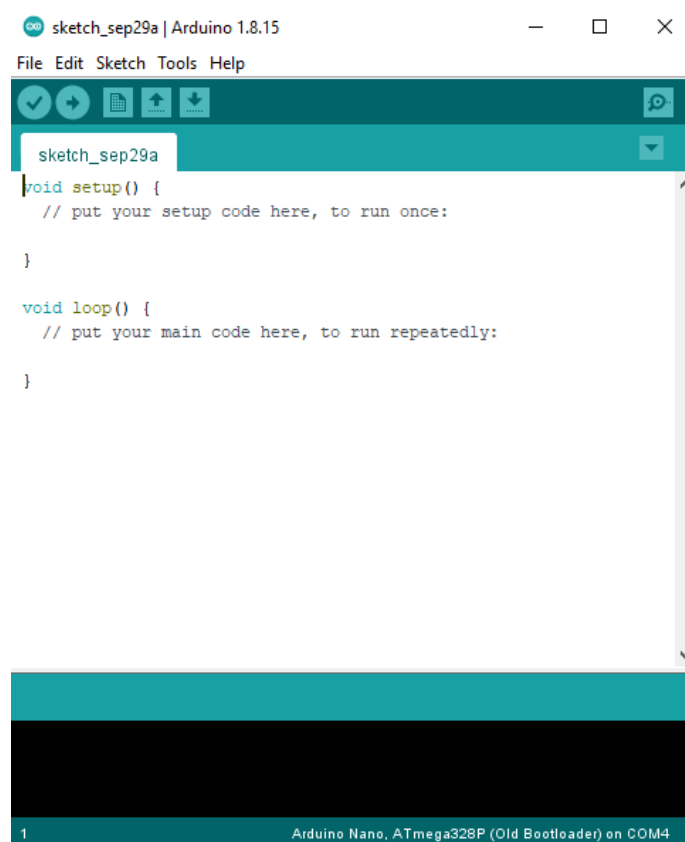


Fig: Arduino IDE

- Verify - Before your program "code" can be sent to the board, it needs to be converted into instructions that the board understands. This process is called compiling or verifying.
- Upload - This compiles and then transmits over the USB cable to your board.
- Create new Sketch - This opens a new window to create a new sketch.
- Open Existing Sketch - This loads a sketch from a file on your computer.
- Save Sketch - This saves the changes to the sketch you are working on.

- Serial Monitor – It can be used as a debugging tool, testing out concepts or to communicate directly with the Arduino board.
- Sketch Editor - This is where you write or edit sketches
- Text Console - This shows you what the IDE is currently doing and is also where error messages display if you make a mistake in typing your program. (Often called a syntax error)
- Line Number - This shows you what line number your cursor is on. It is useful since the compiler gives error messages with a line number.

Writing a Program in Arduino IDE

1. Creating a Folder:

- Make a new folder on Desktop.
- Name it as “EEE 3210_*[Your group number]*”

2. Launching the Arduino IDE:

- First make a folder on Desktop. Name it as EEE 3210.
- Click on **Start** → **Programs** → **Arduino**
- Arduino IDE software will appear.

3. Writing the program:

In most programming languages, you start with a program that simply prints “Hello, World” to the screen. The equivalent in the micro-controller world is getting a light to blink on and off. This is the simplest program we can write to show that everything is functioning correctly.

Write the following code in the sketch editor:

```
void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000);                    // wait for a second
    digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
    delay(1000); }                  // wait for a second
```

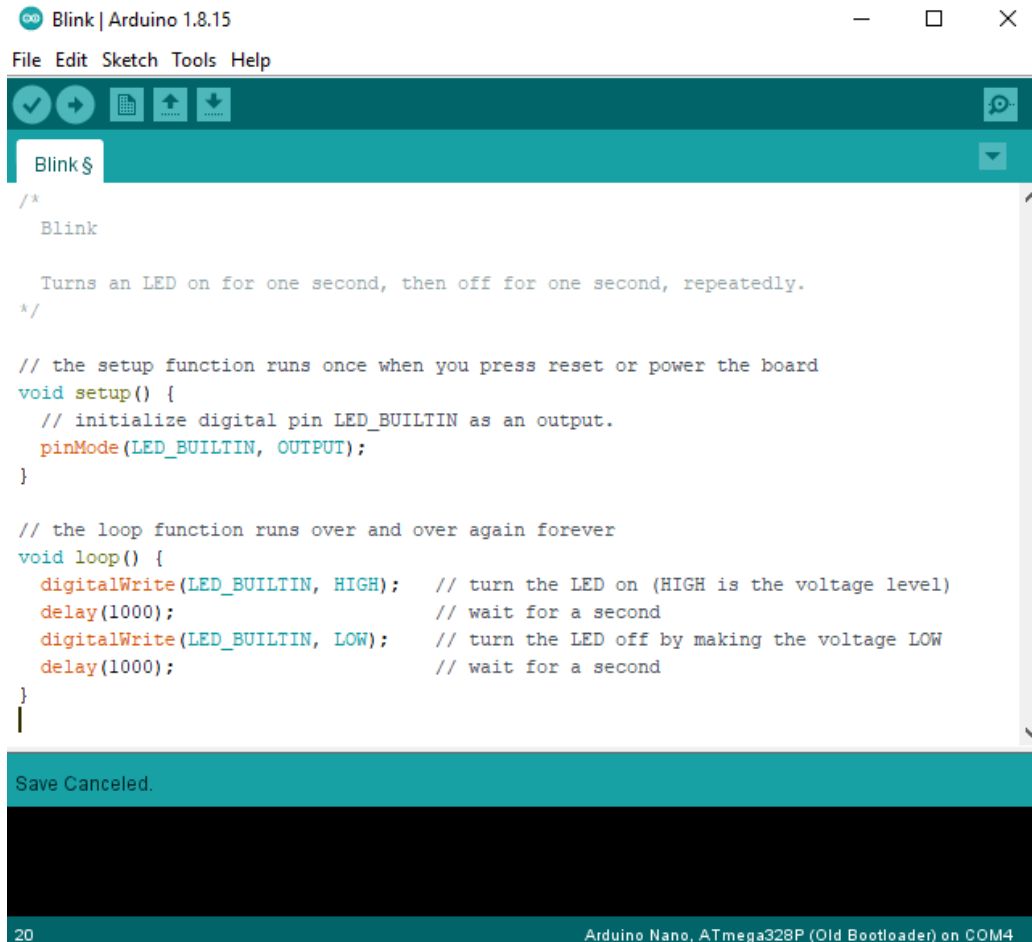


Fig: A program to blink an LED

4. Uploading the Code to Arduino Board:

Once you have written the code, compile and upload the code in your Arduino board. You will see that the built-in LED on your Arduino UNO will turn on for one second and turn off for one second repeatedly.

LAB NO: 1

LAB NAME: Introduction to Serial monitor and Arduino programming language.

Objective: The objective of this lab is to know about Serial monitor and the basic structures and syntax of Arduino programming language.

Basic Structure of Arduino programming language

The basic structure of the Arduino programming language is fairly simple and runs in at least two parts. These two required parts, or functions, enclose blocks of statements.

```
void setup() {  
    statements; // put your setup code here, to run once:  
}  
  
void loop() {  
    statements; // put your main code here, to run repeatedly:  
}
```

setup()

The setup() function is called once when your program starts. You can use it to initialize pin modes or the serial monitor. It must be included in a program even if there are no statements to run.

```
void setup() {  
    pinMode (pin, OUTPUT); // sets the 'pin' output  
}
```

loop()

The loop() function includes the code to be executed continuously allowing the program to change, respond, and control the Arduino board.

```
void loop()  
{  
    digitalWrite (pin, HIGH); // turns 'pin' on  
    delay(1000); // pauses for one second  
}
```

Serial Monitor

Serial monitor is basically the medium through which we can communicate with the Arduino. Arduino is mainly a piece of hardware. If we want to put some value in the hardware, we need to have a medium

through which we can exchange information. Serial monitor can be used as a debugging tool, testing out concepts or to communicate directly with the Arduino board.

Serial.begin(9600): This command opens serial port and sets the baud rate for serial data transmission. 9600 is a significant number which indicates baud rate for communicating with the computer.

Serial.print(): Prints data to serial port. When we want to show the output of a particular command, we use this function.

Serial.println(): Prints data in a new line.

Example 01

```
void setup(){  
    Serial.begin(9600);  
}  
  
void loop(){  
    Serial.println("Bangladesh");  
    delay(1000)  
}
```

Command	Description	Example
{ } curly braces	Curly braces define the beginning and end of function blocks and statement blocks such as the void loop() function and the for and if statements.	<i>void setup(){ Serial.begin(9600); }</i>
; semicolon	A semicolon must be used to end a statement and separate elements of the program. A semicolon is also used to separate elements in a for loop.	<i>Int x = 13;</i>
/*...*/ block comments	Block comments, or multi-line comments, are areas of text ignored by the program and are used for large text descriptions of code or comments that help others understand parts of the program. They begin with /* and end with */.	<i>/* this is a block comment */</i>
// line comments	Single line comments begin with // and end with the next line of code. Like block comments, they are ignored by the program and take no memory space.	<i>Int x = 13; // sets the value of x</i>

Table: Basic command structure

Variables

A variable is a way of naming and storing a numerical value for later use by the program. A variable can be declared in a number of locations throughout the program and where this definition takes place determines what parts of the program can use the variable.

Declaring a variable means defining its value type, as in int, long, float etc., setting a specified name, and optionally assigning an initial value. This only needs to be done once in a program but the value can be changed any time using arithmetic and various assignments.

Example 02

```
void setup(){  
  Serial.begin(9600);  
}  
  
void loop(){  
  int a = 5;  
  int b = 6;  
  int c = a+b;  
  Serial.println(c);  
  delay(1000);  
}
```

Example 03

```
void setup(){  
  Serial.begin(9600);  
  float a = 5.5555;  
  float b = 6.75765;  
  float c = a+b;  
  Serial.println(c,4);  
  delay(1000);  
}  
  
void loop(){  
}
```

A variable can be declared at the beginning of the program before void setup(), locally inside of functions, and sometimes within a statement block such as for loops. A global variable is one that can be seen and used by every function and statement in a program. This variable is declared at the beginning of

the program, before the setup() function. A local variable is one that is defined inside a function or as part of a for loop. It is only visible and can only be used inside the function in which it was declared.

Example 04

```
Int a =18;

void setup(){

  Serial.begin(9600);

  Serial.println(a,BIN); //prints binary value of a
}

void loop(){

int b = 20;

Serial.println(b,HEX); //prints Hex value of b

delay(1000); //executes a delay of 1second

}
```

User input

In order to get an input from the user we have to ask the user for input, wait for the user to enter the input through the serial monitor and then read the information from the Serial port.

while (Serial.available() == 0) : *Serial.available ()* is a function that when you call it, returns a “1” if the user inputs any data. If the User did not enter any data the function returns a “0”. We are using the comparison operator == (equal to) to ensure that the program will not continue until the User has given their input.

Serial.parseInt() : Reads the integer value the user has given as input.

Serial.readString(): Reads the string value the user has given as input.

Example 05

```
string id="";

void setup(){

  Serial.begin(9600);

  Serial.println("Please Enter your Student ID:");

  while (Serial.available()==0);

  {

  }

  id = Serial.readString();
```

```
Serial.print("Your ID is: ");
```

```
Serial.println(id);
```

```
}
```

```
void loop(){
```

```
}
```

Example 06

```
int a;
```

```
void setup()
```

```
{
```

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

```
Serial.println("Please Enter your value:");
```

```
while (Serial.available() != 0);
```

```
{
```

```
}
```

```
a = Serial.parseInt();
```

```
Serial.println("Binary:")
```

```
Serial.println(a, BIN);
```

```
Serial.println("Hexadecimal:");
```

```
Serial.println(a, HEX);
```

```
}
```

```
void loop(){
```

```
}
```

Example 07

Write a program that will add two integer inputs given by the user and show the corresponding result in Hexadecimal and Binary.

```
int a;
```

```
int b;
```

```
int c;
```

```
void setup(){
```

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

```
Serial.println("Please enter first value");  
  
while(Serial.available()!=0);  
  
{  
  
}  
  
a = Serial.parseInt();
```

```
Serial.println("Please enter first value");  
  
while(Serial.available()!=0);  
  
{  
  
}  
  
b = Serial.parseInt();
```

```
c = a+b;
```

```
Serial.println("Result in HEXADECIMAL:");  
  
Serial.println(c,HEX);  
  
Serial.println("Result in BINARY:");  
  
Serial.println(c,BIN);  
  
}
```

```
void loop()
```

```
{  
  
}
```

LAB NO: 2

LAB NAME: Blinking both internal and external Light Emitting Diode (LED) using Arduino UNO and Interfacing switches with Arduino UNO

Objective: In this experiment you will learn how to blink both internal and external LED using Arduino UNO and how to interface switches with Arduino UNO.

Light Emitting Diode (LED)

LEDs are small, powerful lights that are used in many different applications. We can easily work on blinking an LED with the help of Arduino.

Analog Input/Output Pins

A₀ – A₅ (14-19) are the analog pins in Arduino. They are used to take analog inputs. Arduino usually takes digital input but through these pins we can take analog input. They can also be used as digital pins. Analog pins do not need to be first declared as INPUT nor OUTPUT.

The port for these 6 analog pins is PORT C.

Digital Input/Output Pins

Pin 2-13 in the Arduino UNO are the digital pins. We can use them for both input and output.

PORT D constitutes digital pins DP0 – DP7. (8 pins)

PORT B constitutes digital pins DP9 – DP13. (6 pins)

pinMode(pin,mode)

pinMode() is used in void setup() to configure a specified pin to behave either as an INPUT or an OUTPUT. Arduino digital pins are default to inputs, so they don't need to be explicitly declared as inputs with pinMode(). Pins configured as INPUT are said to be in a high-impedance state.

```
pinMode( pin, OUTPUT ); // sets 'pin' to output
```

Pins configured as OUTPUT are said to be in a low-impedance state and can provide 40 mA of current to other devices/circuits. This is enough current to brightly light up an LED.

For example if we want to use 5 no. pin as output we can initialize as

```
pinMode (5, OUTPUT);
```

or we can initialize as

DDRD = 0b 0010 0000 ; DDR = Data Direction Register, D for port D

Or, DDRD = 0x20;

Both binary or hexadecimal format can be used.

For PORT B we can initialize as

```
pinMode (8, OUTPUT);
```

```
or, DDRB = 0b 0000 0001
```

```
or, DDRB = 0x 01
```

Here we have initialized 8 no. digital pin as output which is declared as '1' in LSB position.

digitalRead(pin)

`digitalRead()` reads the value from a specified digital pin with the result either HIGH or LOW. The pin can be specified as either a variable or constant (0-13).

```
Value = digitalRead(pin); // set 'value' equal to the input pin
```

digitalWrite(pin, value)

`digitalWrite()` outputs either logic level HIGH or LOW at a specified digital pin. The pin can be specified as either a variable or constant (0-13).

analogRead(pin)

`analogRead(pin)` reads the value from a specified analog pin with a 10-bit resolution. This function only works on the analog input pins (0-5). The resulting integer values range from 0 to 1023.

analogWrite(pin, value)

`analogWrite()` writes a pseudo-analog value using hardware enabled pulse width modulation (PWM) to an output pin marked PWM. The value can be specified as a variable or constant with a value from 0 – 255. A value 0 generates a steady 0 volts output at the specified pin; a value of 255 generates a steady 5 volts output at the specified pin. For values in between 0 and 255, the pin rapidly alternates between 0 and 5 volts.

Blinking internal LED of Arduino UNO

Example 01

```
void setup() {  
    // initialize digital pin LED_BUILTIN as an output.  
    pinMode(LED_BUILTIN, OUTPUT);  
}  
  
// the loop function runs over and over again forever  
void loop() {  
    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)  
    delay(1000);           // wait for a second  
    digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW  
    delay(1000);           // wait for a second  
}
```

Example 02

```
Int led = 13;

void setup() {

    // initialize digital pin LED_BUILTIN as an output.
    pinMode(led, OUTPUT);

}

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

    digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)

    delay(1000);             // wait for a second

    digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW

    delay(1000);             // wait for a second

}
```

Example 03

```
void setup()

{

    DDRB= 0b 0010 0000; / DDRB= 0x 20;

}

void loop()

{

    PORTB= 0b 0010 0000; / PORTB= 0x 20;

    delay(1000);

    PORTB= 0b 0000 0000; / PORTB= 0x 00;

    delay(1000);

}
```

Blinking external LED using Arduino UNO

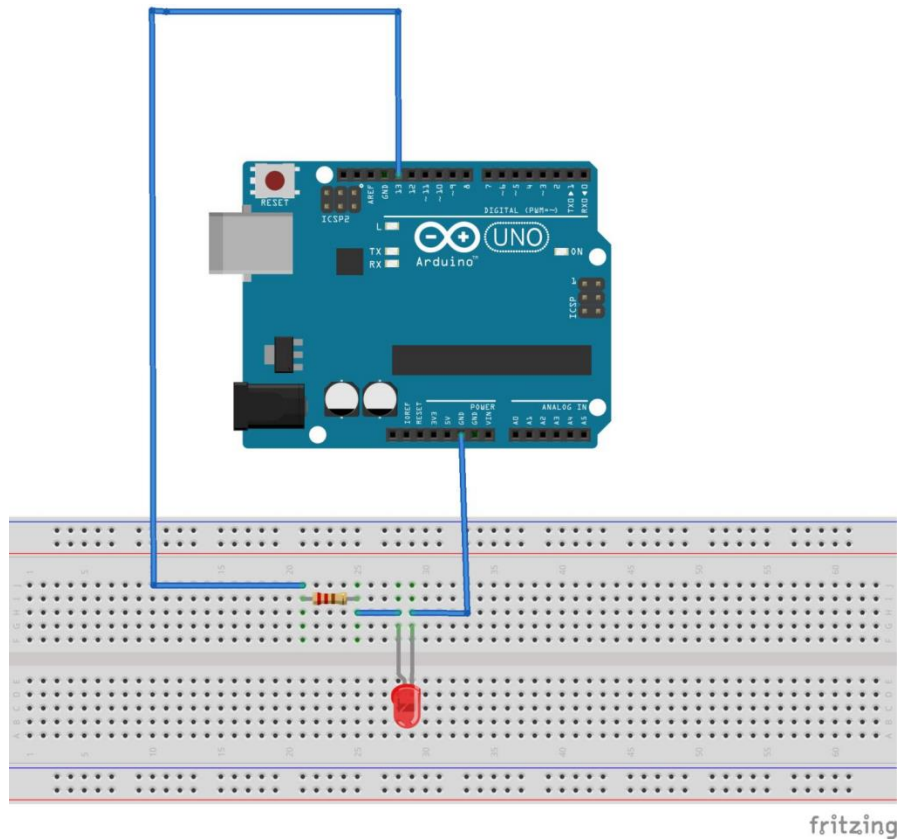


Figure: Circuit Diagram

Example 04

```
int led =13;

void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(led, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
```

```

    digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
    delay(1000);           // wait for a second
}

```

Example 05

A program to blink LED only two times.

```

int led =2;

void setup() {

    pinMode(led, OUTPUT);

    digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000);           // wait for a second
    digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
    delay(1000);           // wait for a second
    digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000);           // wait for a second
    digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
    delay(1000);           // wait for a second
}

// the loop function runs over and over again forever

void loop() {

}

```

Example 06

A program to blink LED only two times.

```

int led =2;

void setup() {

    pinMode(led, OUTPUT);

    for (int i =0;i<2;i++)

    {

        digitalWrite(led,HIGH);

        delay(1000);

        digitalWrite(led,LOW);
    }
}

```



```

    delay(1000);

}}

void loop() {

}

```

Functions

A function is a block of code that has a name and a block of statements that are executed when the function is called. The functions void setup () and void loop() have already been discussed and other built-in functions will be discussed later.

Custom functions can be written to perform repetitive tasks and reduce clutter in a program. Functions are declared by first declaring the function type. This is the type of value to be returned by the function such as 'int' for an integer type function. If no value is to be returned the function type would be void. After type, declare the name given to the function and in parenthesis any parameters being passed to the function.

```

type functionName (parameters)

{

    statements;

}

```

Example 07

```

/*blinking led user defined*/

int led =2;

int n =5;

void setup() {

    // initialize digital pin LED_BUILTIN as an output.

    pinMode(led, OUTPUT);

    blinkLED(n);

}

void loop() {

}

void blinkLED(int n){

    for (int i =0;i<n;i++)

    {

        digitalWrite(led,HIGH);

        delay(1000);
    }
}

```

```
digitalWrite(led,LOW);  
  
delay(1000);  
  
}  
}
```

Example 08

```
/*blinking led user defined*/  
  
#define led 2  
  
int n;  
  
float td = 1;  
  
void setup() {  
  
    Serial.begin(9600);  
  
    pinMode(led, OUTPUT);  
  
    Serial.print("How many times to blink?");  
  
    while (Serial.available () == 0)  
  
    {  
  
    }  
  
    n = Serial.parseInt();  
  
    blinkLED(n,td);  
  
}  
  
void loop() {  
  
}  
  
void blinkLED(int n, float td){  
  
    for (int i =0;i<n;i++)  
  
    {  
  
        digitalWrite(led,HIGH);  
  
        delay(td*1000);  
  
        digitalWrite(led,LOW);  
  
        delay(td*1000);  
  
    }  
  
}
```

Example 09

```
int a;

int b;

int c;

int led =2;

void setup(){

  Serial.begin(9600);

  Serial.println("Please Enter first value:");

  while (Serial.available()==0);

  {

  }

  a = Serial.parseInt();

  Serial.println("Please Enter second value:");

  while (Serial.available()==0);

  {

  }

  b = Serial.parseInt();

  c = a+b;

  for(int i=0; i<c ;i++)

  {

    digitalWrite(led,HIGH);

    delay(1000);

    digitalWrite(led,LOW);

    delay(1000);

  }

}

void loop(){

}
```

LAB NO: 3

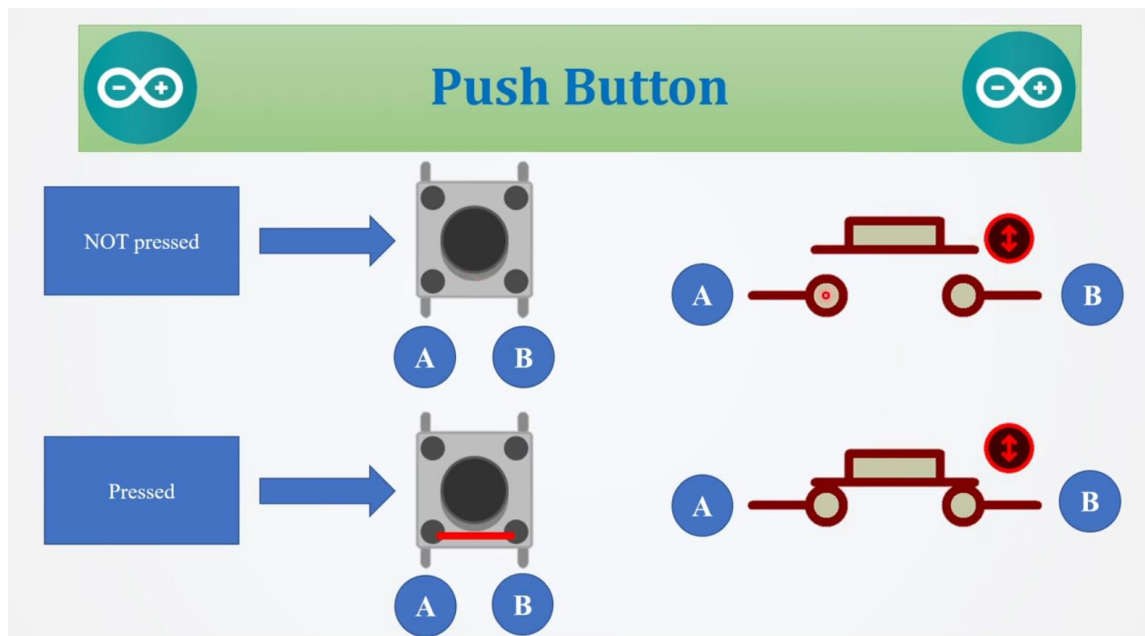
LAB NAME: Interfacing switches and keypad with Arduino UNO

Interfacing Switch with Arduino

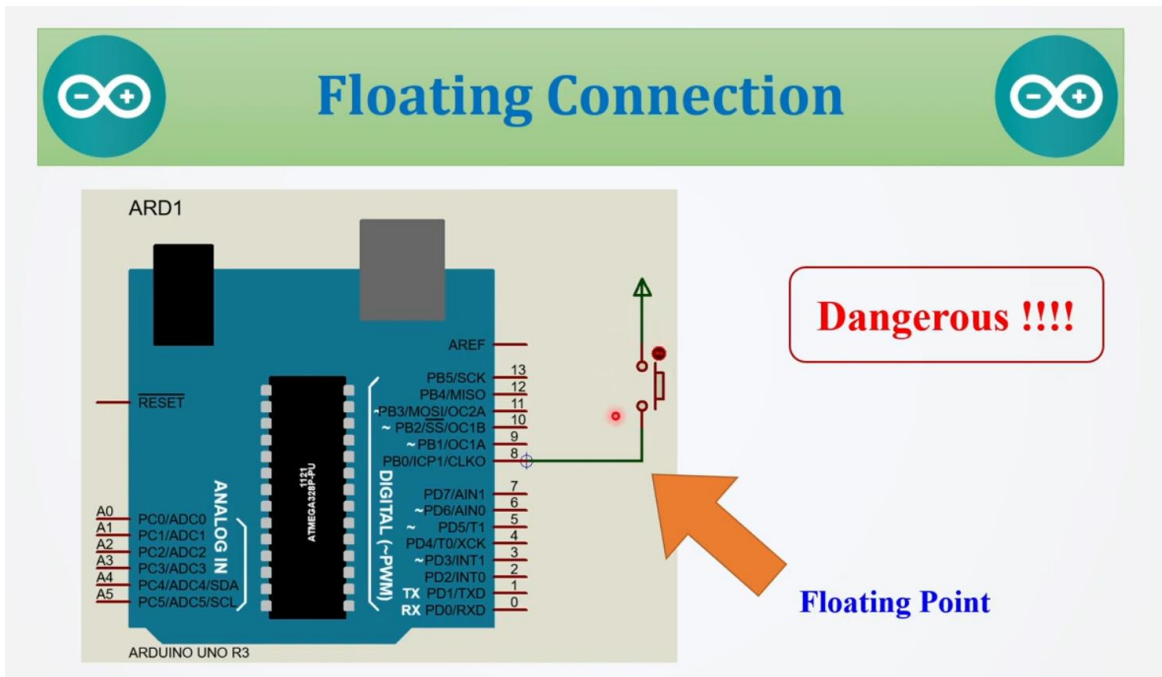
A switch is an electrical component that can disconnect or connect the conducting path in an electrical circuit, interrupting the electric current or diverting it from one conductor to another. We can easily interface a switch with Arduino. It can be used as an input device.

Device	Symbol	Description
BUTTON		SPST Push Button
SW-SPDT		Interactive SPDT Switch

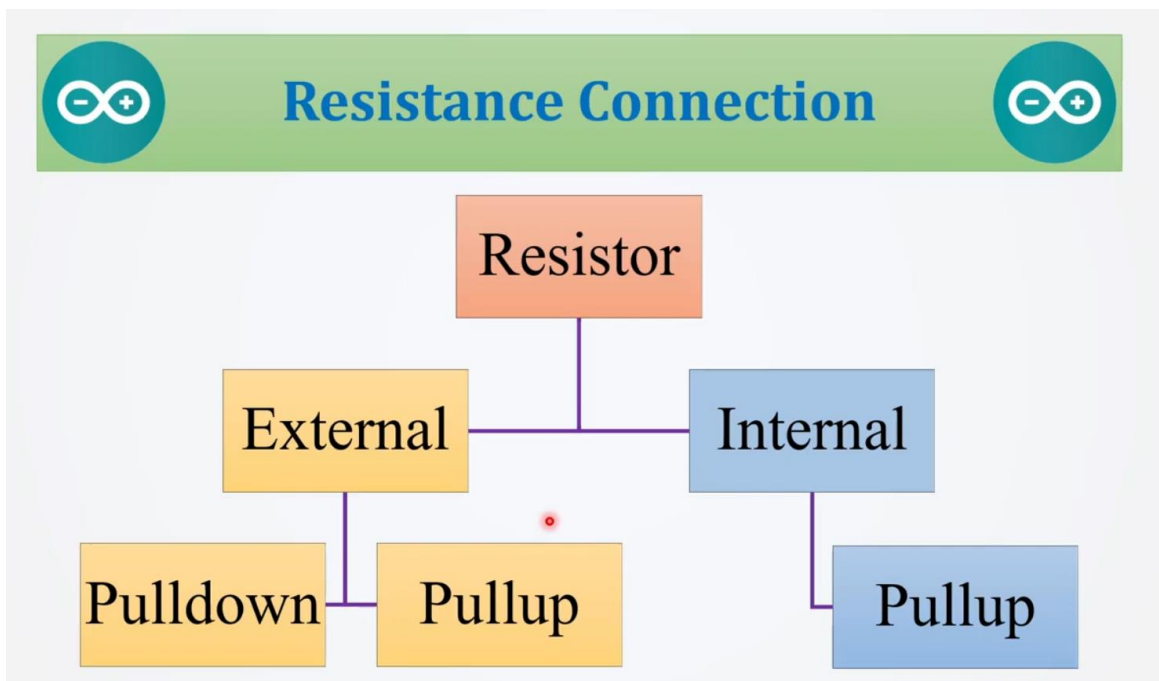
A **Push Button** switch is a type of switch which consists of a simple electric mechanism or air switch mechanism to turn something on or off. Depending on model they could operate with momentary or latching action function.



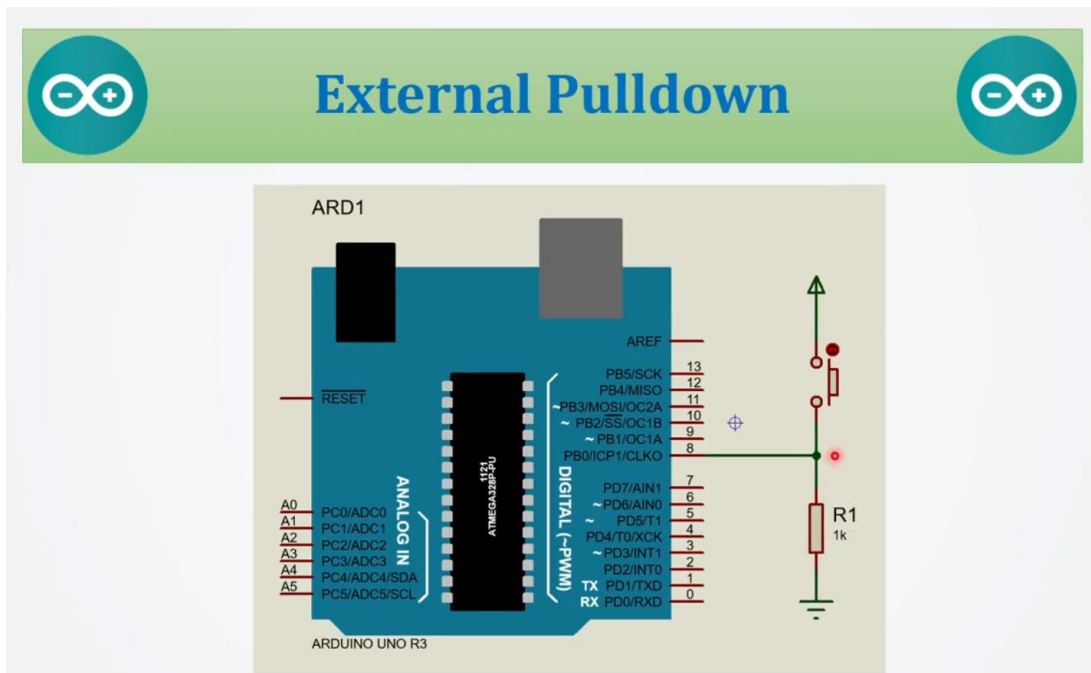
Floating Connection occurs when a certain node is neither connected to the positive bias nor the ground.



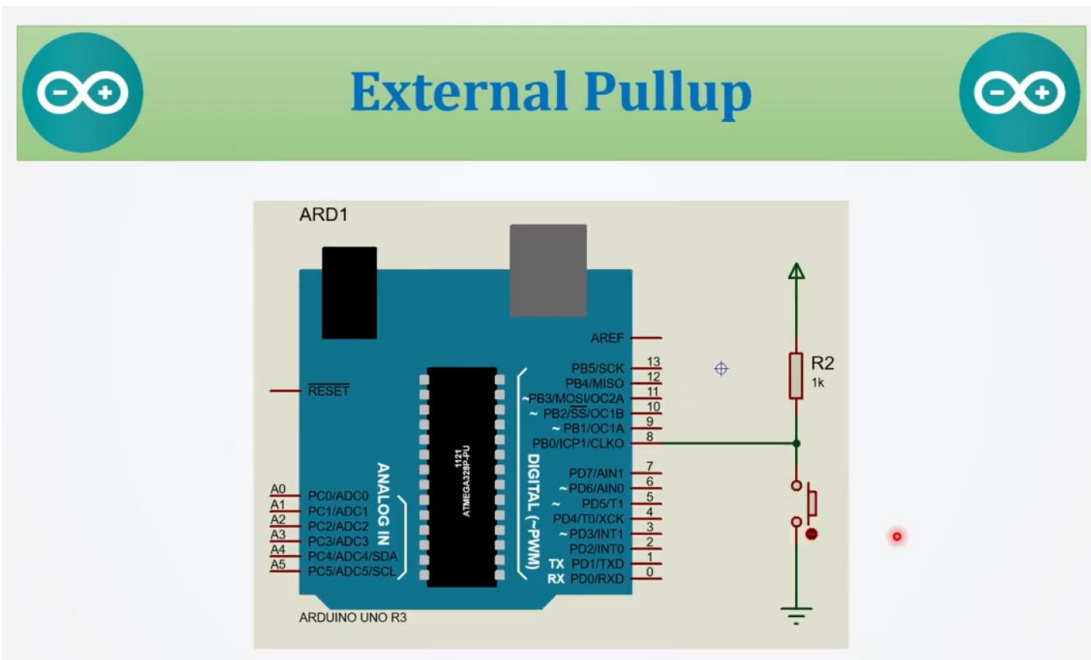
Resistance Connection



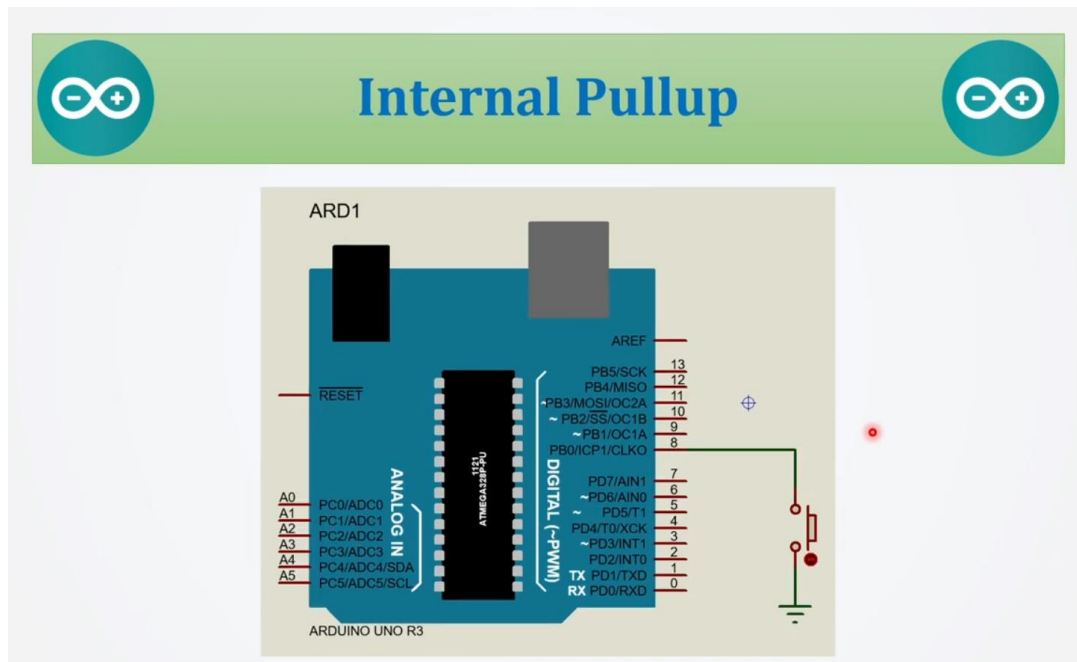
External Pull-down



External Pull-up

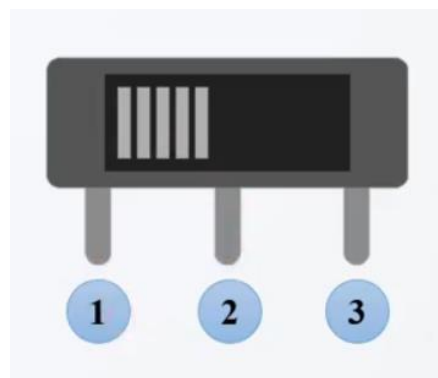


Internal Pull-up



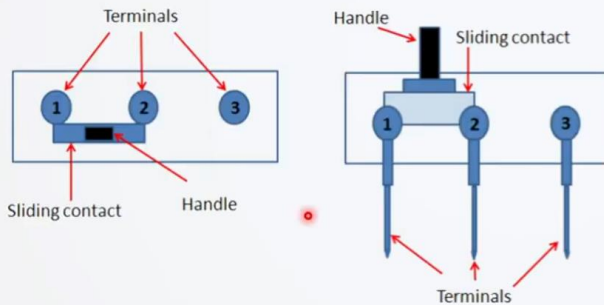
Slide switches are used to control current flow in a circuit, and typically use a mechanical slider to switch a current on and off, by sliding between an open and closed state. Well suited for controlling current flow in smaller circuits, it is very common to see slide switches used as a primary power switch in small, battery-operated electrical devices.

- It is a three terminal device.
- Also known as Single Pole Double Throw (SPDT) switch.





Pin Connection



Pin	Name
1	Throw (Terminal) 1
2	Pole (Common)
3	Throw (Terminal) 2



Operation

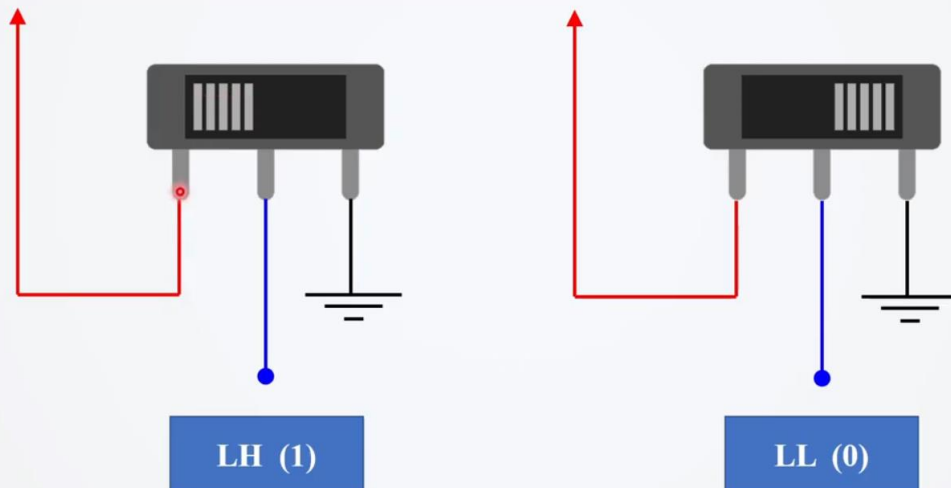
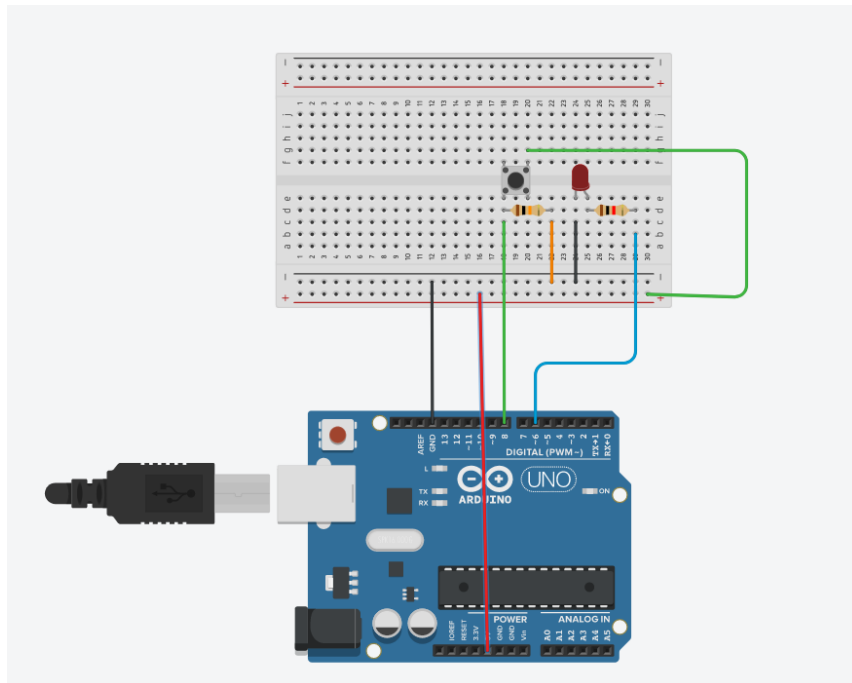


Fig: Interfacing a pushbutton switch with Arduino

Example 1 (External Pulldown)



Code:

```
#define ledPin 6 // choose the pin for the LED
#define switchPin 8 // choose the input pin (for a pushbutton)
int val = 0; // variable for reading the pin status

void setup()
{
  pinMode(ledPin, OUTPUT); // declare LED as output
  pinMode(switchPin, INPUT); // declare pushbutton as input
}

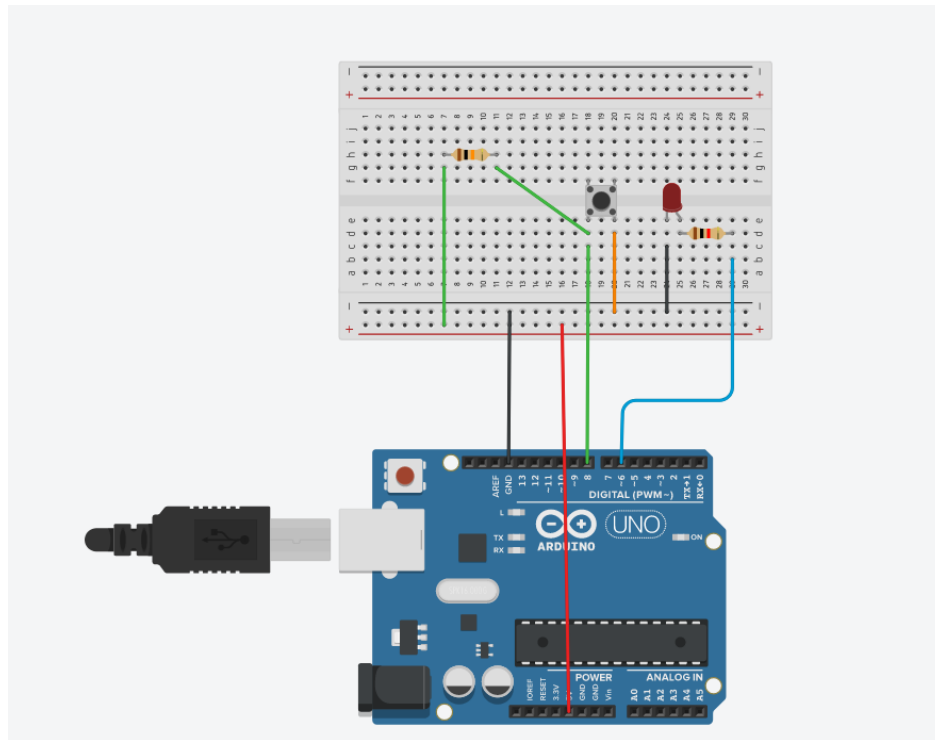
void loop()
{
  val = digitalRead(switchPin); // read input value
  if (val == HIGH)
  { // check if the input is HIGH (button released)
    digitalWrite(ledPin, HIGH); // turn LED ON
  } else {
    digitalWrite(ledPin, LOW); // turn LED OFF } }
```

```
}  
}
```

Example2(External Pulldown_Blinking LED continuously)

```
#define ledPin 6 // choose the pin for the LED  
  
#define switchPin 8 // choose the input pin (for a pushbutton)  
  
int val = 0; // variable for reading the pin status  
  
void setup()  
{  
  pinMode(ledPin, OUTPUT); // declare LED as output  
  pinMode(switchPin, INPUT); // declare pushbutton as input  
}  
  
void loop()  
{  
  val = digitalRead(switchPin); // read input value  
  
  if (val == HIGH)  
  {while(!Serial.available())  
  {  
    digitalWrite(ledPin, HIGH); // turn LED ON  
    delay(1000);  
    digitalWrite(ledPin, LOW);  
    delay(1000);  
  }  
  } else {  
    digitalWrite(ledPin, LOW); // turn LED OFF } }  
}  
}
```

Example 3 (External Pullup)



Code:

```
//Example3(External PullUP)

#define ledPin 6 // choose the pin for the LED
#define switchPin 8 // choose the input pin (for a pushbutton)

int val = 0; // variable for reading the pin status

void setup()
{
  pinMode(ledPin, OUTPUT); // declare LED as output
  pinMode(switchPin, INPUT); // declare pushbutton as input
}

void loop()
{
  val = digitalRead(switchPin); // read input value

  if (val == HIGH)
  { // check if the input is HIGH (button released)
    digitalWrite(ledPin, HIGH); // turn LED ON
  }
}
```

```

} else {

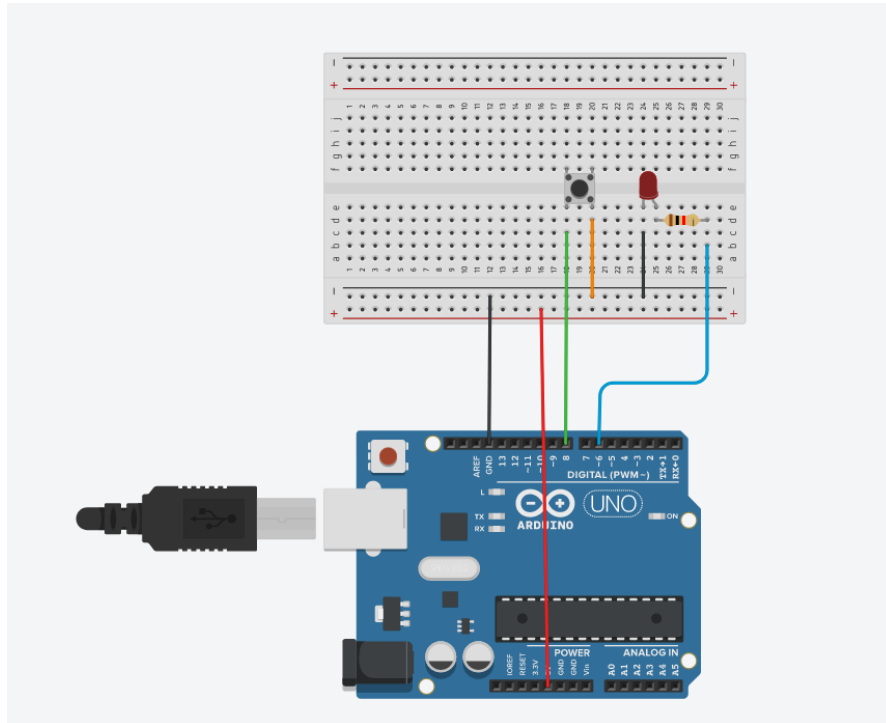
    digitalWrite(ledPin, LOW); // turn LED OFF } } //LED turns off when switch is pressed

}

}

```

Example 4 (Internal PULLUP)



Code:

```

#define ledPin 6 // choose the pin for the LED

#define switchPin 8 // choose the input pin (for a pushbutton)

int val = 0; // variable for reading the pin status

void setup()
{
    pinMode(ledPin, OUTPUT); // declare LED as output
    pinMode(switchPin, INPUT_PULLUP); // declare pushbutton as input
}

void loop()
{
    val = digitalRead(switchPin); // read input value

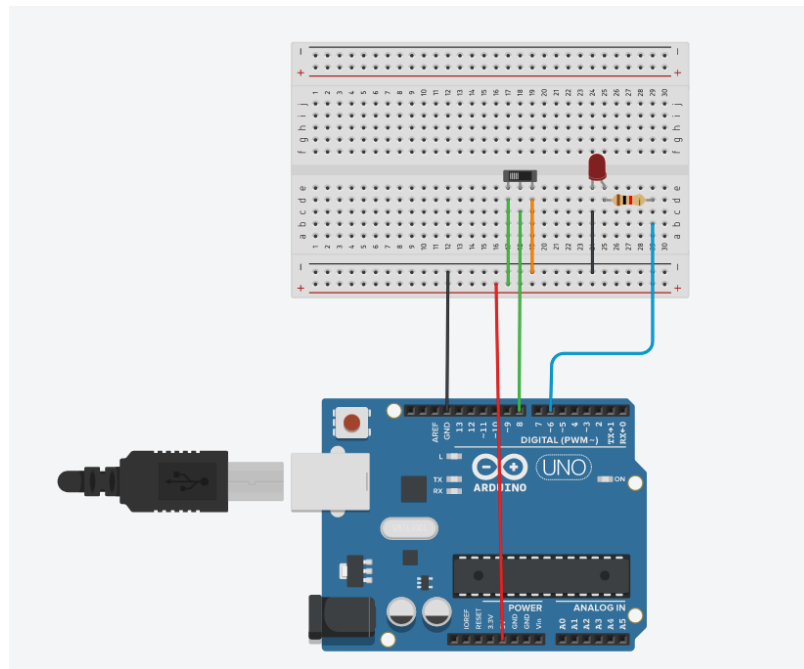
```

```

if (val == HIGH)
{ // check if the input is HIGH (button released)
  digitalWrite(ledPin, HIGH); // turn LED ON
} else {
  digitalWrite(ledPin, LOW); // turn LED OFF }}
}
}

```

Example 5 (Switch)



Code:

```

#define LED 6 // choose the pin for the LED
#define SW 8 // choose the input pin (for a pushbutton)

void setup()
{
  pinMode(LED, OUTPUT); // declare LED as output
  pinMode(SW, INPUT); // declare pushbutton as input
}

void loop()
{

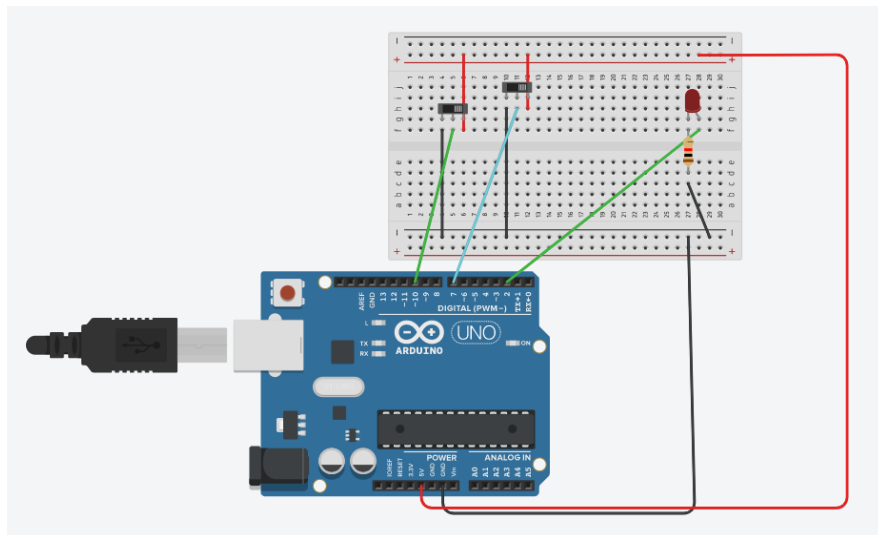
```

```

if (digitalRead(SW) == HIGH)
{ // check if the input is HIGH (button released)
  digitalWrite(LED, HIGH); // LED blinks continuously
  delay(1000);
  digitalWrite(LED, LOW);
  delay(1000);
} else {
  digitalWrite(LED, LOW); // turn LED OFF } }
}

```

Example 6 (AND Gate using Switches)



```

#define SW1 7
#define SW2 8
#define LED 2

void setup() {
  pinMode(SW1, INPUT);
  pinMode(SW2, INPUT);
  pinMode(LED, OUTPUT);
}

void loop() {

```

```

if(digitalRead(SW1) == HIGH && digitalRead(SW2) == HIGH)

digitalWrite(LED,HIGH);

else

digitalWrite(LED,LOW);

}

```

Example 7 (OR GATE)

```

#define SW1 7

#define SW2 8

#define LED 2

void setup() {

  pinMode(SW1,INPUT);

  pinMode(SW2,INPUT);

  pinMode(LED,OUTPUT);

}

void loop() {

  if(digitalRead(SW1) == LOW && digitalRead(SW2) == LOW)

    digitalWrite(LED,LOW);

  else

    digitalWrite(LED,HIGH);

}

```

Example 8

Take Attendance, Quiz and Final Exam marks as Input. A switch will be pressed for calculating the total marks. If total marks is 'PASS' then green LED will blink. Otherwise a red LED will blink. Assume pass marks is 40.

Code:

```

#define SW1 8

#define LEDG 5

#define LEDR 6

int a;

int q;

int f;

```

```

int r;

void setup() {
  Serial.begin(9600);

  pinMode(SW1,INPUT);
  pinMode(LEDG,OUTPUT);
  pinMode(LEDG,OUTPUT);

  Serial.println("Please enter attendance marks");

  while(Serial.available()!=0);

  {

  }

  a = Serial.parseInt();

  Serial.println("Please enter quiz marks");

  while(Serial.available()!=0);

  {

  }

  q = Serial.parseInt();

  Serial.println("Please enter final marks");

  while(Serial.available()!=0);

  {

  }

  f = Serial.parseInt();

  r = a+q+f;
}

void loop() {

  if(digitalRead(SW1) == HIGH && r>40)
  {digitalWrite(LEDG,HIGH);

  Serial.println("PASSED");

  }

  else if(digitalRead(SW1) == HIGH && r<40)
  {digitalWrite(LEDG,HIGH);

  Serial.println("FAILED");
}

```


}

}

Interfacing a 4 x 4 Keypad with Arduino UNO

Keypad is used as an input device to read the key pressed by the user and to process it.

A 4x4 keypad consists of 4 rows and 4 columns. Switches are placed between the rows and columns.

A 4x3 keypad consists of 4 rows and 3 columns.

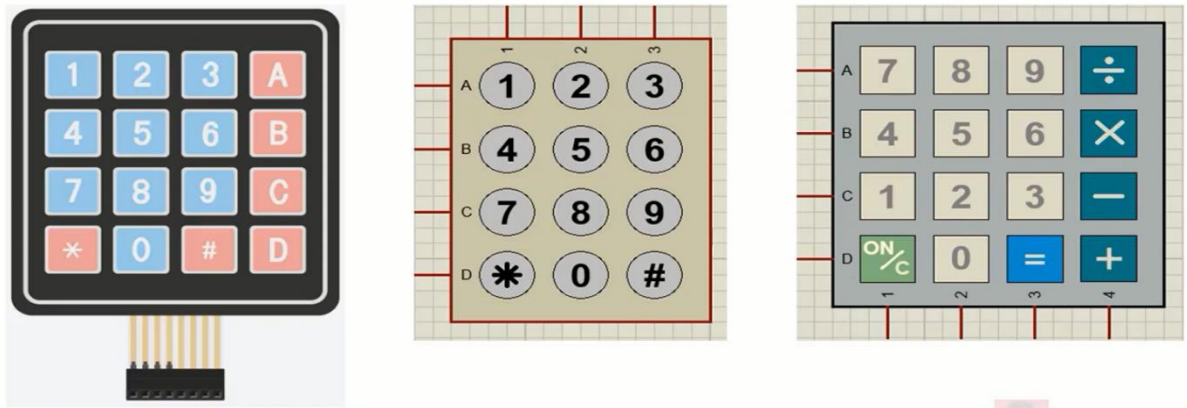


Figure: Keypad

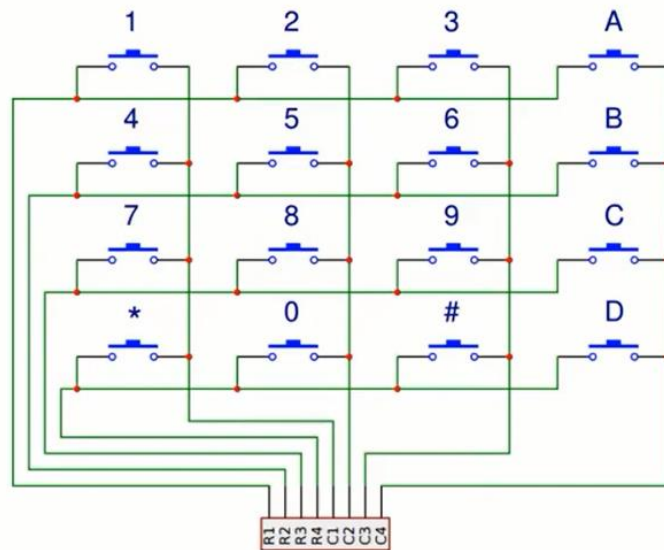


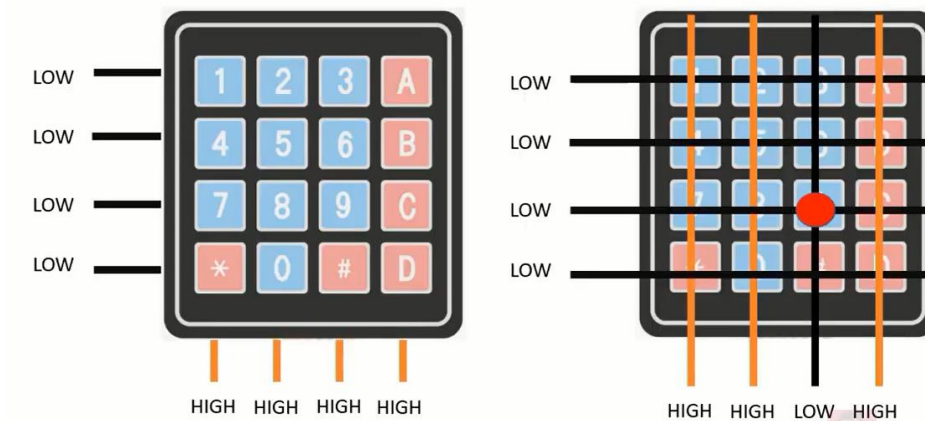
Figure: Internal diagram of a 4 x 4 Keypad

Working Principle of Keypad

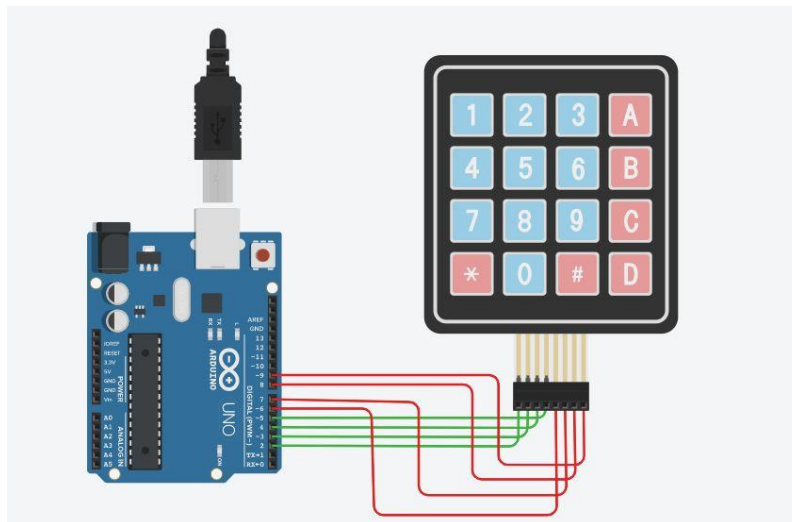
When no key is pressed, microcontroller sends low signal to all row lines & high signal to all column lines.

Whenever a user presses a key, then corresponding row & column gets shorted. Then the status of the

corresponding column becomes low. So, the microcontroller can now realize which button has been pressed.



Interfacing with Arduino



R1, R2, R3, R4 pinouts of keypad are connected to digital pins 2, 3, 4 & 5 of the Arduino. C1, C2, C3, C4 pinouts of keypad are connected to digital pins 6, 7, 8 & 9 of the Arduino.

Keypad	Arduino Digital Pins
R1	2
R2	3
R3	4
R4	5
C1	6
C2	7
C3	8
C4	9

Code

```
keypad $
#include <Keypad.h>
const byte ROWS = 4; /* four rows */
const byte COLS = 4; /* four columns */
/* define the symbols on the buttons of the keypads */
char Keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {2,3,4,5}; /* connect to the row pinouts of the keypad */
byte colPins[COLS] = {6, 7, 8, 9}; /* connect to the column pinouts of the keypad */

/* initialize an instance of class NewKeypad */
Keypad customKeypad = Keypad( makeKeymap(Keys), rowPins, colPins, ROWS, COLS);

void setup() {
  Serial.begin(9600);
}

void loop() {
  char customKey = customKeypad.getKey();

  if (customKey) {
    Serial.println(customKey);
  }
}
```

Functions Used

1. ***makeKeymap(keys)***: This function is used to initialize the internal keymap to be equal to the user defined key map (in function syntax given above, keys).
2. ***Keypad customKeypad = Keypad(makeKeymap(keys), rowPins, colPins, rows, cols)***: This defines an object customKeypad of the class Keypad and initializes it.
rowPins and colPins are the pins on Arduino to which the rows and columns of the keypad are connected to rows and cols are the number of rows and columns the keypad has.
3. ***customKeypad.getKey()***: This function is used to identify which key is pressed on the keypad.

LAB NO: 4

LAB NAME: Basic Arithmetic Operations and Seven Segment Module

Arithmetic operators include addition, subtraction, multiplication, and division. They return the sum, difference, product, or quotient of the operands.

$y = y + 2;$

$x = x - 5;$

$i = y * 2;$

$j = y / 2;$

The operation is conducted using different data type of the operands.

Data Type	Example	Description
byte	byte a = 18;	Byte stores an 8-bit numerical value without decimal points. They have a range of 0-255.
int	int b = 150;	Integers are the primary datatype for storage of numbers without decimal points and store a 16-bit value with range of 32,767 to -32,768.
long	long c = 7000;	Extended size datatype for long integers, without decimal points, stored in a 32-bit value with a range of 2,147,483,647 to -2,147,483,648.
float	float pi = 3.14;	A datatype for floating point numbers, or numbers that have a decimal point. Floating point numbers have greater resolution than integers and are stored as a 32-bit value.

There are also other operators like compound assignments, comparison operators and logical operators.

Compound assignments	$x ++$ // increments x by +1 $x --$ // decrements x by -1 $x += y$ // increments x by +y $x -= y$ // decrements x by -y $x *= y$ // multiplies x by y $x /= y$ // divides x by y	Compound assignments combine an arithmetic operation with a variable assignment. These are commonly found in for loops.
Comparison operators	$x == y$ // x is equal to y $x != y$ // x is not equal to y $x < y$ // x is less than y $x > y$ // x is greater than y $x <= y$ // x is less than or equal to y $x >= y$ // x is greater than or equal to y	Comparisons of one variable or constant against another are often used in if statements to test if a specified condition is true.

Logical operators	Logical AND: <i>if (x > 0 && x < 5)</i> Logical OR: <i>if (x > 0 y > 0)</i> Logical NOT: <i>if (!x > 0)</i>	Logical operators are usually a way to compare two expressions and return a TRUE or FALSE depending on the operator.
-------------------	--	--

Arrays

An array is a collection of values that are accessed with an index number. Any value in the array maybe called upon by calling the name of the array and the index number of the value. Arrays are zero indexed, with the first value in the array beginning at index number 0. An array needs to be declared and optionally assigned values before they can be used.

```
int myArray [ ] = {10, 12, 17};
```

Likewise, it is possible to declare an array by declaring the array type and size and later assign values to an index position.

```
int myArray [10]; // declares and array with 11 positions
```

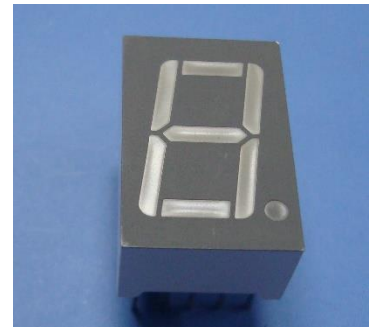
```
myArray [3] = 20; // assigns the 4th index the value 20
```

To retrieve a value from an array, assign a variable to the array and index position:

```
x = myArray[3]; // x now equals 20
```

Seven Segment Module

Seven segment display is an output display device that provide a way to display information in the form of text or decimal numbers. It is widely used in digital clocks, basic calculators, electronic meters, and other electronic devices that display numerical information. It consists of seven segments of light emitting diodes (LEDs) which is assembled like numerical 8. If we consider the dot point then there are total eight LEDs.



Two configurations are available for a 7-segment display.

1. Common Cathode (CC)
2. Common Anode (CA)

The internal structure of both types is nearly the same. The difference is the polarity of the LEDs and common terminal.

Common Cathode: In a common cathode seven-segment display, all seven LEDs and the dot LED have the cathodes connected to pin 3 or pin 8.

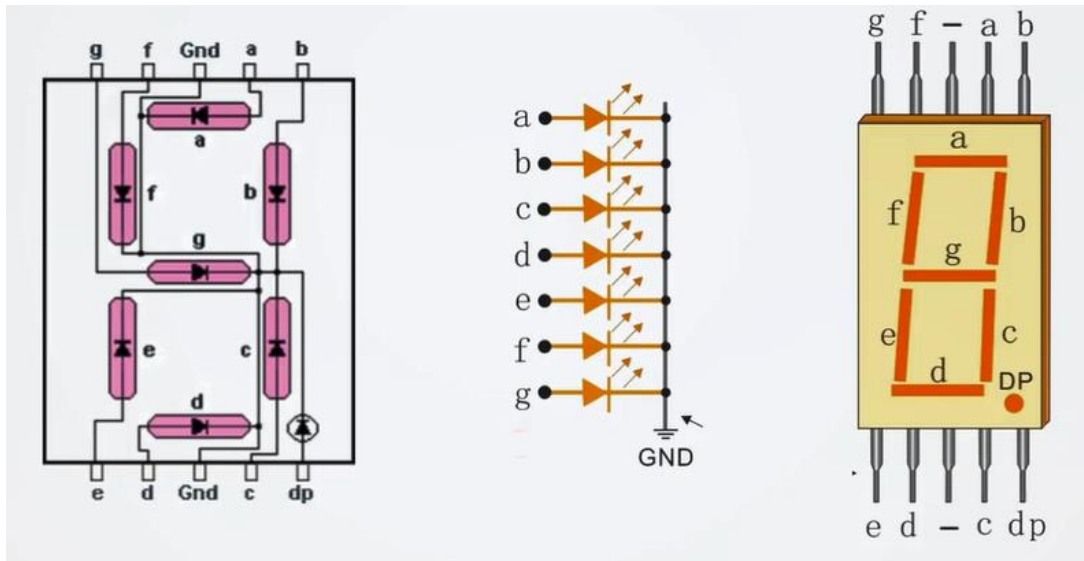


Figure: Internal Setup Diagram of Common Cathode SSD

To use this display, connect GROUND to pins 3 and pin 8 and connect +5V to the other pins to make the individual segments light up.

Common Anode: In common anode seven-segment display, the positive terminal or anode of all the eight LEDs are connected together and then connected to pin 3 and pin 8.

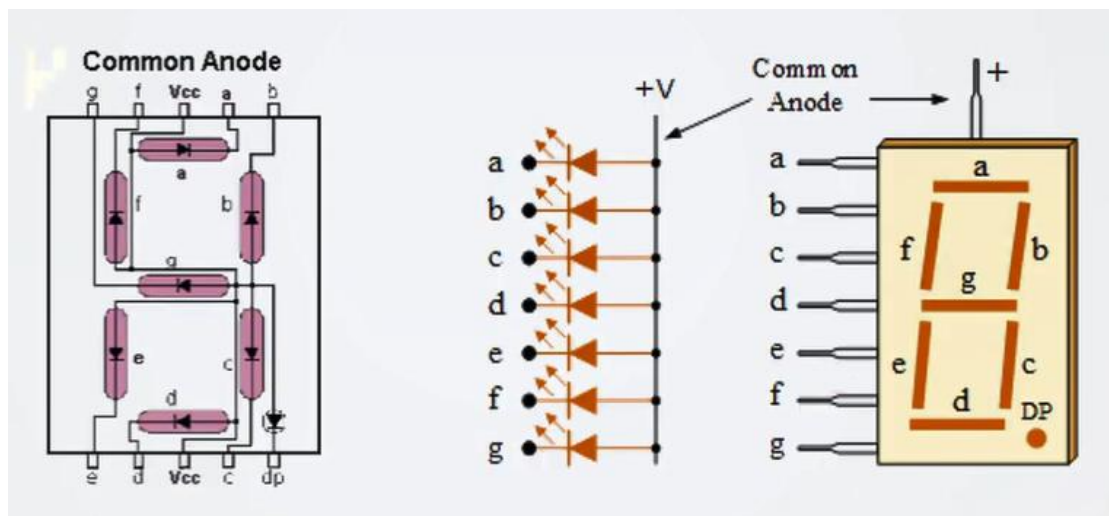
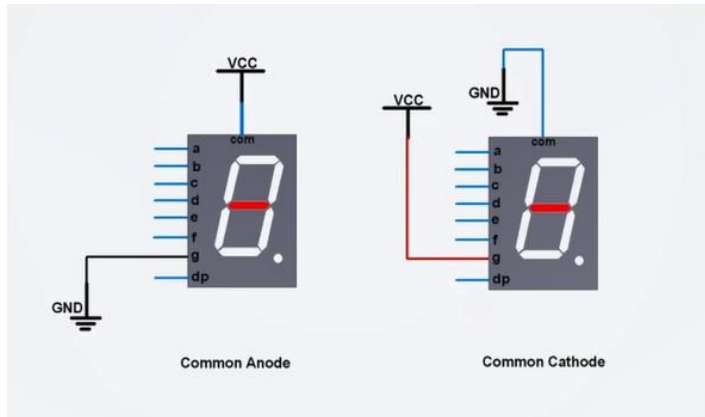


Figure: Internal Setup Diagram of Common Anode SSD

To use this display, connect +5v to pins 3 and pin 8 and connect GROUND to the other pins to make the individual segments light up.



There are total $2^7 = 128$ states possible for the seven-segment display (ignoring dot point). The frequently used values are the Hexadecimal numbers (0 – F).

To display a particular number, you turn on the individual segments as shown in the following table (for CC):

Digit	pgfedcba	p	g	f	e	d	c	b	a
0	0x3F	off	off	on	on	on	on	on	on
1	0x06	off	off	off	off	off	on	on	off
2	0x5B	off	on	off	on	on	off	on	on
3	0x4F	off	on	off	off	on	on	on	on
4	0x66	off	on	on	off	off	on	on	off
5	0x6D	off	on	on	off	on	on	off	on
6	0x7D	off	on	on	on	on	on	off	on
7	0x07	off	off	off	off	off	on	on	on
8	0x7F	off	on	on	on	on	on	on	on
9	0x6F	off	on	on	off	on	on	on	on
A	0x77	off	on	on	on	off	on	on	on
B	0x7C	off	on	on	on	on	on	off	off
C	0x39	off	off	on	on	on	off	off	on
D	0x5E	off	on	off	on	on	on	on	off
E	0x79	off	on	on	on	on	off	off	on
F	0x71	off	on	on	on	off	off	off	on

Interfacing Arduino with Seven segment display

To interface the Arduino UNO with a seven segment display the components required are:

- 1 x seven segment display (common cathode)
- 1 x Arduino UNO
- 1 x Breadboard
- Resistors
- Jumper wires

The pins of the seven-segment display should be connected to Arduino pins (0-7). Common pins (pin 3 and pin 8) are connected to GROUND.

7 Segment Display (CC)	Arduino UNO Board
Segment a (Pin no. 7)	PD0 (Digital Pin 0)
Segment b (Pin no. 6)	PD1 (Digital Pin 1)
Segment c (Pin no. 4)	PD2 (Digital Pin 2)
Segment d (Pin no. 2)	PD3 (Digital Pin 3)
Segment e (Pin no. 1)	PD4 (Digital Pin 4)
Segment f (Pin no. 9)	PD5 (Digital Pin 5)
Segment g (Pin no. 10)	PD6 (Digital Pin 6)
Segment p (Pin no. 5)	PD7 (Digital Pin 7)
Common Cathode (CC) Pin (3 or 8)	GND or Port B Pin (Through Resistor)

Figure: Connection Table

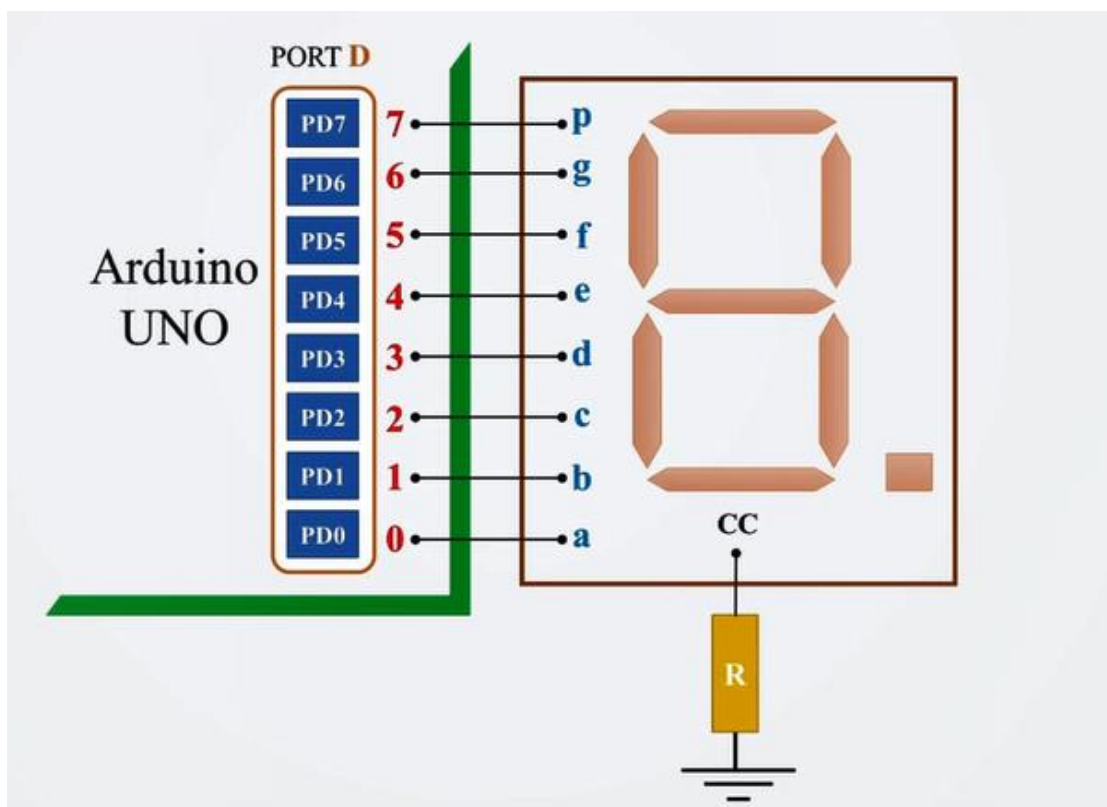
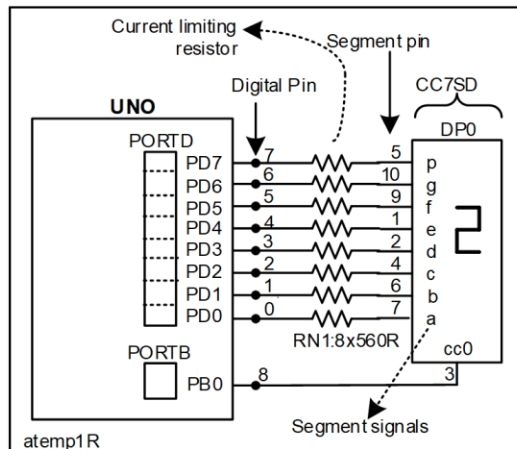


Figure: Connection Diagram

Code

Example-1

```
void setup() {  
    DDRD = 0b11111111;  
    PORTD = 0xFF;  
}  
  
void loop() {  
}
```



Look up Table (LUT): lupTable[16]

Array Location	cc-code in hex format	digit
lupTable[0]	3F	0
lupTable[1]	06	1
lupTable[2]	5B	2
lupTable[3]	4F	3
lupTable[4]	66	4
lupTable[5]	6D	5
lupTable[6]	7D	6
lupTable[7]	07	7
lupTable[8]	7F	8
lupTable[9]	6F	9
lupTable[A]	77	A
lupTable[B]	7C	B
lupTable[C]	39	C
lupTable[D]	5E	D
lupTable[E]	79	E
lupTable[F]	71	F

lutd2ccq

Figure: Interfacing Arduino with Seven Segment Display

Example-2

Printing 0 – F in seven segment display (common cathode)

```
byte lupTable[16] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71};  
  
void setup() {  
    DDRD = 0xFF; //all port lines of PORTD are output  
    pinMode(8, OUTPUT); //PB0 is output
```

```

digitalWrite(8,LOW); // Activating 7SD CC pin

// Printing from '0' to 'F'
for (int i = 0; i <16; i++)
{
    PORTD = lupTable[i]; //Assigning the Values
    delay(1000);
}
digitalWrite(8,HIGH);
}

void loop() {
}

```

Example-3

Printing 0 – F in seven segment display(common anode)

```

byte lupTable[16] = {0x3F, 0x06, 0x5B, 0x4F,0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79,
0x71};

void setup() {
    DDRD = 0xFF; //all port lines of PORTD are output
    pinMode(8, OUTPUT); //PB0 is output
    digitalWrite(8,HIGH); // Activating 7SD CC pin

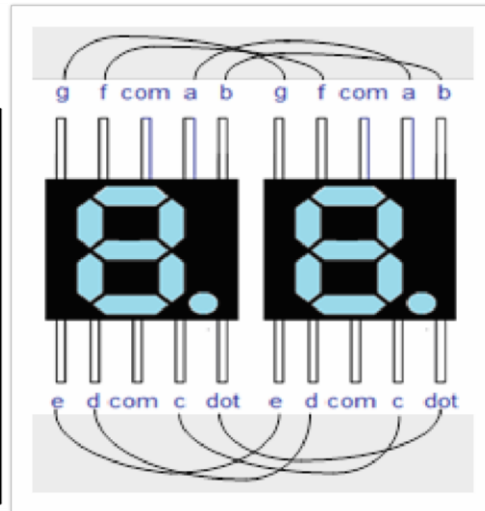
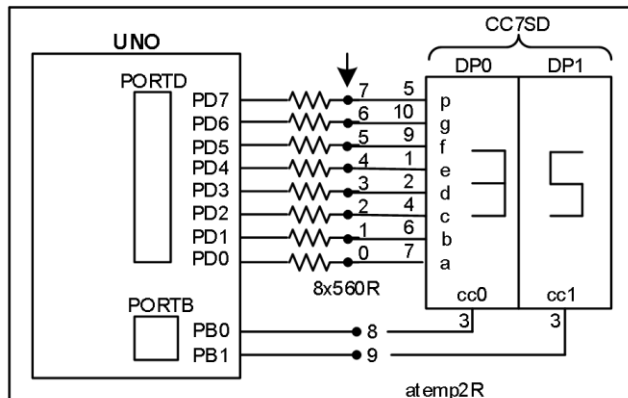
    // Printing from '0' to 'F'
    for (int i = 0; i <16; i++)
    {
        PORTD = ~lupTable[i]; //Assigning the Values. ~ = bit-wise NOT operand
        delay(1000);
    }
    digitalWrite(8,LOW);
}

void loop() {}

```

Multiple Seven Segment Display

Two seven segment displays can be cascaded for showing multiple digits simultaneously.



Example-4

```
byte lupTable[16] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71};
```

```
void setup() {
```

```
    DDRD = 0xFF; //all port lines of PORTD are output
```

```
    pinMode(8, OUTPUT); //PB0 is output
```

```
    pinMode(9, OUTPUT); //PB1 is output
```

```
}
```

```
void loop() {
```

```
    digitalWrite(8, LOW);
```

```
    digitalWrite(9, HIGH);
```

```
    PORTD = lupTable[3];
```

```
    delay(10);
```

```
    digitalWrite(8, HIGH);
```

```
    digitalWrite(9, LOW);
```

```
    PORTD = lupTable[5];
```

```
    delay(10);
```

```
}
```

Example-5

```
byte lupTable[16] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71};
```

```
void setup()
{
  DDRD = 0xFF;           //all port lines of PORTD are output
  pinMode(8, OUTPUT);    //PB0 is output
  pinMode(9, OUTPUT);    //PB1 is output

  byte x1 = 0x23;
  byte x2 = 0x1A;
  byte z = x1 + x2;      //z = result = 0x3D

  byte indexU = z && 0x0F; //indexU = 0x0D; it will be used as an index to point ccArray[]
  byte indexT = (z && 0xF0) >> 4; //indexT = 0x03; index of the Ten position digit
}
```

```
void loop()
{
  PORTD = lupTable[indexT]; //show Ten positional digit (3)
  digitalWrite(8, LOW);
  digitalWrite(9, HIGH);

  delay(1);              //delay to synchronize multiplexing of digits

  PORTD = lupTable[indexU]; //show Unit positional digit (D)
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);

  delay(1);

}
```

Lab NO:5

LAB NAME: ADC and Analog temperature sensor using Arduino

ADC

An Analog to Digital Converter (ADC) is a very useful feature that converts an analog voltage on a pin to a digital number. Not every pin on a microcontroller has the ability to do analog to digital conversions. On the Arduino board, these pins have an 'A' in front of their label (A0 through A5) to indicate these pins can read analog voltages. ADCs can vary greatly between microcontroller. The ADC on the Arduino is a 10-bit ADC meaning it has the ability to detect 1,024 (2^{10}) discrete analog levels. Some microcontrollers have 8-bit ADCs ($2^8 = 256$ discrete levels) and some have 16-bit ADCs ($2^{16} = 65,536$ discrete levels).

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

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

Analog to digital conversions is dependent 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:

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

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?

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

Rearranging:

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

$$x = 434$$

So, the ADC value is 434.

LM35 analog temperature sensor using Arduino UNO

LM35 is an inexpensive, precision Centigrade temperature sensor. It provides an output voltage that is linearly proportional to the Centigrade temperature and is, therefore, very easy to use with the Arduino. The sensor does not require any external calibration or trimming to provide accuracies of $\pm 0.5^\circ\text{C}$ at room temperature and $\pm 1^\circ\text{C}$ over the -50°C to $+155^\circ\text{C}$ temperature range.

The output scale factor of the LM35 is 10 mV/°C and it provides an output voltage of 250 mV at 25°C.

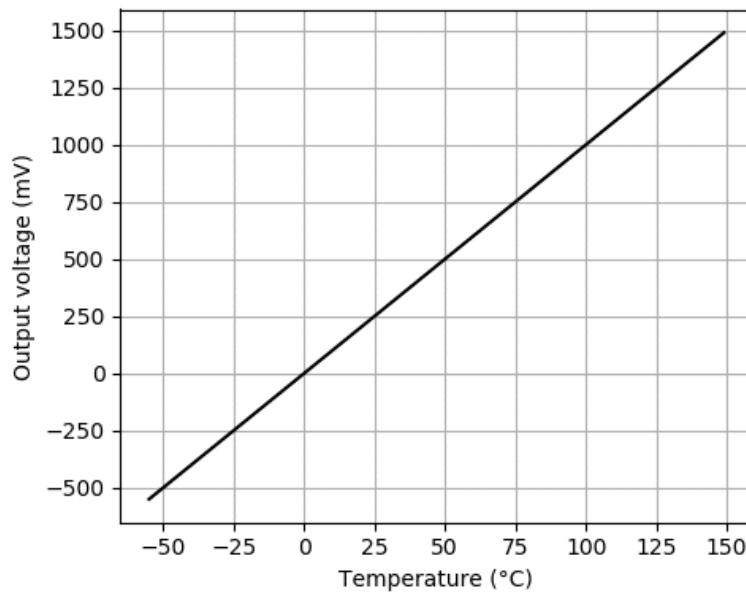


Figure: LM35 output voltage in mV versus temperature

The pinout of the LM35 sensor is as follows:

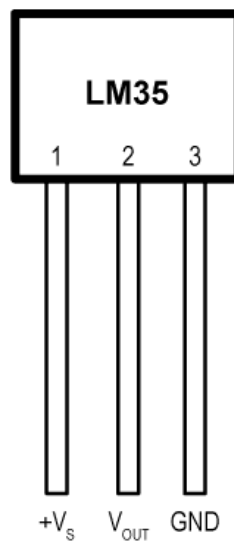


Figure: Pinout of LM35 sensor

Here Pin-1 (V_s) is the positive supply pin, Pin-2 (V_{out}) is the temperature sensor analog output and Pin-3 (GND) is the device ground pin.

Interfacing the LM35 Temperature sensor with Arduino

To interface LM35 temperature sensor with Arduino UNO the components required are:

- LM35 analog temperature sensor
- Arduino UNO
- Breadboard
- Jumper wires
- USB cable

Circuit Diagram:

Start by connecting the +V_s pin to the 5 V output of the Arduino and the GND pin to the ground.Next, connect the middle pin (V_{OUT}) to any of the analog inputs of the Arduino. In this case, the analog input pin A0 is used.

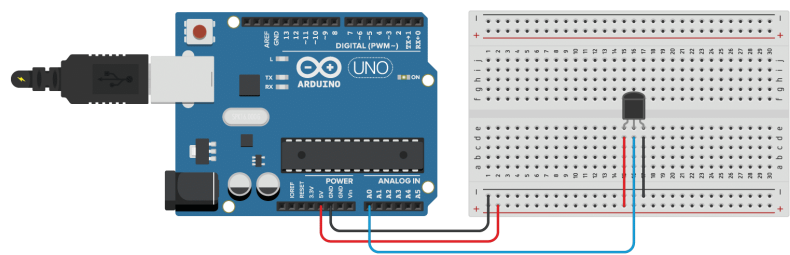


Figure: Connection Diagram

LM35	Arduino
Pin 1 (+V _S)	5 V
Pin 2 (V _{OUT})	Pin A0
PIN 3 (GND)	GND

Figure: Connection Table

To convert the output voltage of the sensor into the temperature in degree Celsius, you can use the following formula:

$$Temperature\ (^{\circ}C) = V_{OUT} / 10$$

with V_{OUT} in millivolt (mV). So, if the output of the sensor is 750 mV, the temperature is 75°C. The output of the LM35 is connected to one of the analog inputs of the Arduino. The value of this analog

input can be read with the function ***analogRead()***. However, this function will not actually return the output voltage of the sensor. Arduino boards contain a multichannel, 10-bit analog to digital converter (ADC), which will map input voltages between 0 and the operating voltage (5 V or 3.3 V) 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. So, if you use ***analogRead()*** to read the voltage at one of the analog inputs of the Arduino, you will get a value between 0 and 1023.

To convert this value back into the output voltage of the sensor, you can use:

$$V_{OUT} = \text{reading from ADC} * (V_{ref} / 1024)$$

Code

Example 1

LM35 analog temperature sensor with Arduino example code

```
// Define to which pin of the Arduino the output of the LM35 is connected
#define sensorPin A0

void setup() {
  // Begin serial communication at a baud rate of 9600:
  Serial.begin(9600);
}

void loop() {
  // Get a reading from the temperature sensor:
  int reading = analogRead(sensorPin);

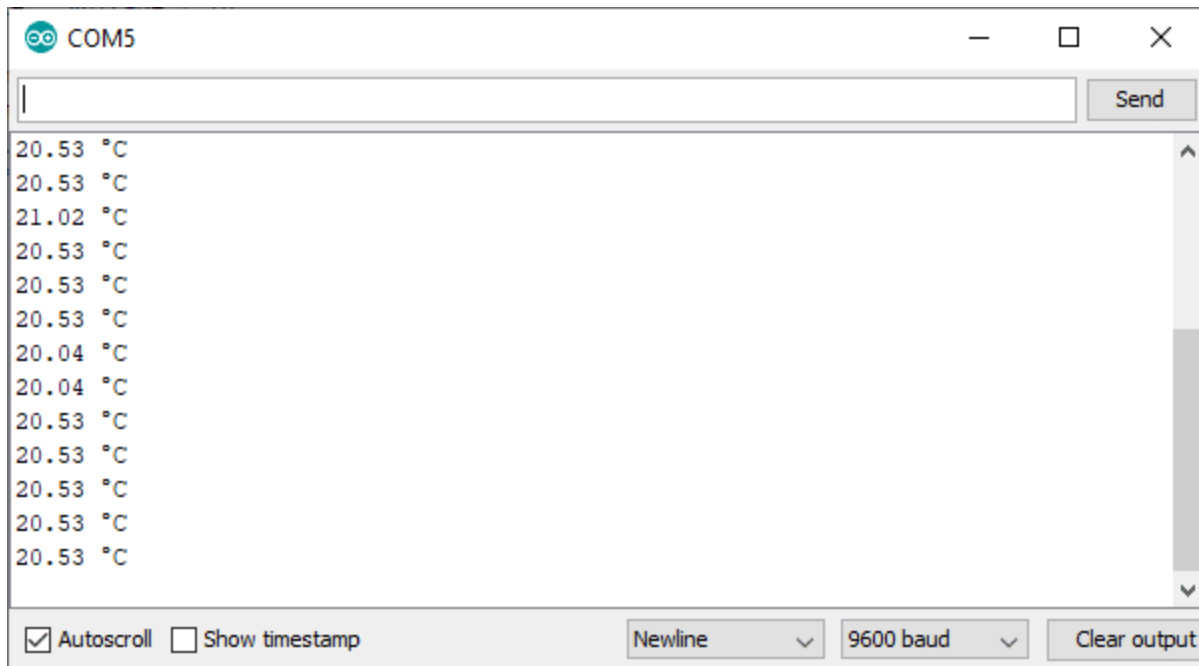
  // Convert the reading into voltage:
  float voltage = reading * (5000 / 1024.0);

  // Convert the voltage into the temperature in degree Celsius:
  float temperature = voltage / 10;

  // Print the temperature in the Serial Monitor:
  Serial.print(temperature);
  Serial.print(" \xC2\xB0"); // shows degree symbol
  Serial.println("C");

  delay(1000); // wait a second between readings
}
```

You should see the following output in the Serial Monitor:



Because we used the default reference voltage of the Arduino for analog input (i.e. the value used as the top of the input range), the maximum resolution we get from the ADC is $5000/1024 = 4.88$ mV or 0.49°C . If we want a higher precision, we can use the built-in 1.1 V reference from the Arduino instead. This reference voltage can be changed using the function ***analogReference()***.

With 1.1 V as the reference voltage, we get a resolution of $1100/1024 = 1.07$ mV or 0.11°C . This limits the temperature range that we can measure to 0 to 110 degrees Celsius.

Example 2

```
#define sensorPin A0
```

```
void setup() {
```

```
  // Begin serial communication at a baud rate of 9600:
```

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

```
  // Set the reference voltage for analog input to the built-in 1.1 V reference:
```

```
  analogReference(INTERNAL);
```

```
}
```

```
void loop() {
```

```
  // Get a reading from the temperature sensor:
```

```
  int reading = analogRead(sensorPin);
```

```
  // Convert the reading into voltage:
```

```
  float voltage = reading * (1100 / 1024.0);
```

```
  // Convert the voltage into the temperature in degree Celsius:
```

```
  float temperature = voltage / 10;
```

```

// Print the temperature in the Serial Monitor:
Serial.print(temperature);
Serial.print(" \xC2\xB0"); // shows degree symbol
Serial.println("C");

```

```

delay(1000); // wait a second between readings
}

```

Example 3

Displaying Temperature in Serial Monitor

```

void setup() {

  Serial.begin(9600);
  analogReference(DEFAULT);
  //analogReference(INTERNAL);
  //analogReference(EXTERNAL);
  pinMode(A0,INPUT);
}

void loop(){
  int x=analogRead(A0);
  float T=(float)((100*(5/1023.0))*x); // DEFAULT

  //float T=(float)((100*(1.1/1023.0))*x); // nothing is connected to AREF (INTERNAL)

  //float T=(float)((100*(3.3/1023.0))*x); // 3.3V connected to AREF (INTERNAL and EXTERNAL)

  Serial.println(T);

  delay(1000);
}

```

Example 4

Displaying Temperature in CC Type 7 Segment Display

```

byte lookuptable[16]={

  0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71 };

void setup() {

  analogReference(DEFAULT);

  pinMode(A0,INPUT);

  DDRD=0xFF;

  DDRB=0xFF;}

void loop(){

  int x=analogRead(A0);

  float T=(float)((100*(5/1023.0))*x);

```

```

for (int j=0;j<100;j++){
int T2=round(T*100);
for (int i=0; i<4; i++){
PORTB=0xFF;
PORTD=lookuptable[T2%10];
if (i==2){
digitalWrite(7,HIGH);}
bitClear(PORTB,i);
delay(5);
T2=T2/10;
}}}

```

Example 5

Displaying Temperature in LCD

```

#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
void setup() {
lcd.init();
lcd.backlight();
analogReference(DEFAULT);
pinMode(A0,INPUT);
}
void loop(){
int x=analogRead(A0);
float T = (100*(5/1023.0))*x;
lcd.setCursor(0,0);
lcd.print(T);
delay(1000); }

```

LAB NO: 6

LAB NAME: Sending measurement of temperature to another Microcontroller based device using software based UART

Objective: The objective of this experiment is to learn about Universal Asynchronous Receiver-Transmitter (UART) and to send measurement of temperature from one microcontroller-based device to another using software based UART

UART: Universal Asynchronous Receiver-Transmitter

UART is a hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end. Embedded systems, microcontrollers, and computers mostly use UART as a form of device-to-device hardware communication protocol. Among the available communication protocols, UART uses only two wires for its transmitting and receiving ends. UART can work with many different types of serial protocols that involve transmitting and receiving serial data. In serial communication, data is transferred bit by bit using a single line or wire. In two-way communication, we use two wires for successful serial data transfer.

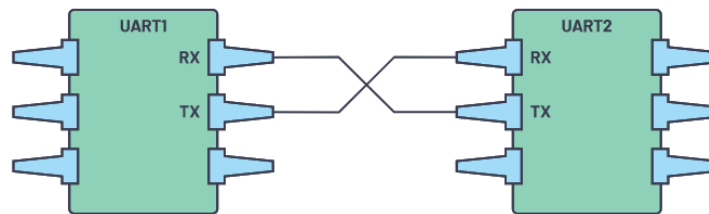


Figure: Two UART directly communicating with each other

The two signals of each UART device are named:

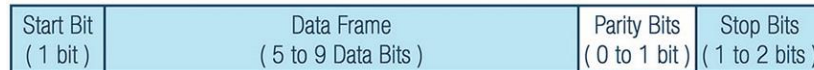
- Transmitter (Tx)
- Receiver (Rx)

The main purpose of a transmitter and receiver line for each device is to transmit and receive serial data intended for serial communication. UART lines serve as the communication medium to transmit and receive one data to another.

For UART and most serial communications, the baud rate needs to be set the same on both the transmitting and receiving device. The baud rate is the rate at which information is transferred to a communication channel. The UART interface does not use a clock signal to synchronize the transmitter and receiver devices; it transmits data asynchronously. Instead of a clock signal, the transmitter generates a bitstream based on its clock signal while the receiver is using its internal clock signal to sample the incoming data. The point of synchronization is managed by having the same baud rate on both devices.

Data Transmission in UART

In UART, the mode of transmission is in the form of a packet. The piece that connects the transmitter and receiver includes the creation of serial packets and controls those physical hardware lines. A packet consists of a start bit, data frame, a parity bit, and stop bits.



Start Bit: The UART data transmission line is normally held at a high voltage level when it's not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for one (1) clock cycle. When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.



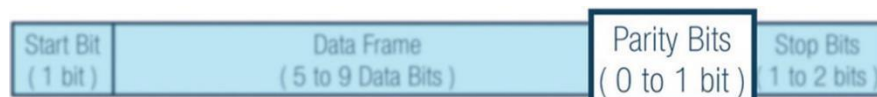
Data Frame: The data frame contains the actual data being transferred. It can be five (5) bits up to eight (8) bits long if a parity bit is used. If no parity bit is used, the data frame can be nine (9) bits long. In most cases, the data is sent with the least significant bit first.



Parity: Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers.

After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the parity bit is a 0 (even parity), the 1 or logic-high bit in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bit or logic highs in the data frame should total to an odd number.

When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd, or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.

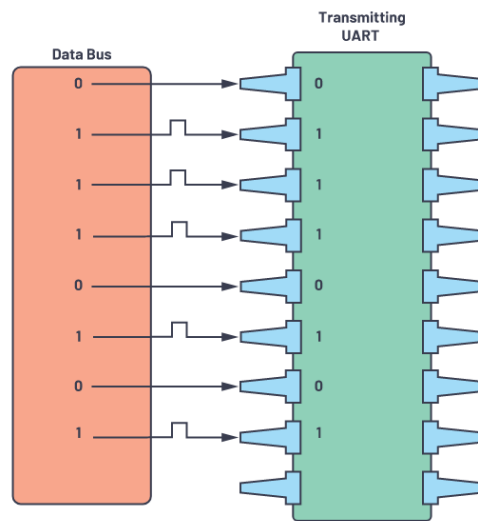


Stop Bits: To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for one (1) to two (2) bit(s) duration.

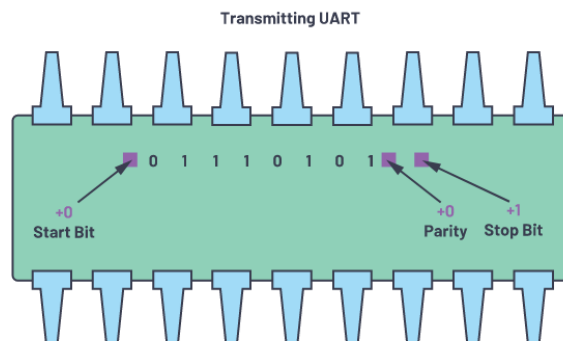


Steps of UART Transmission

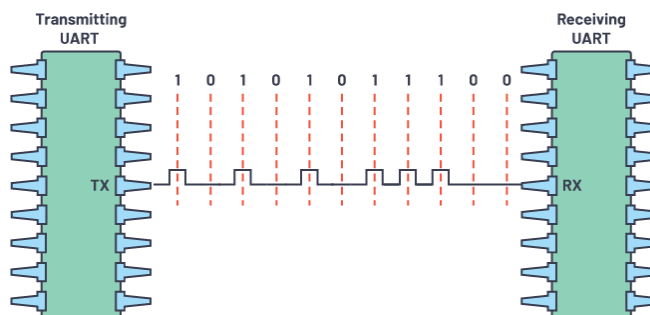
First: The transmitting UART receives data in parallel from the data bus.



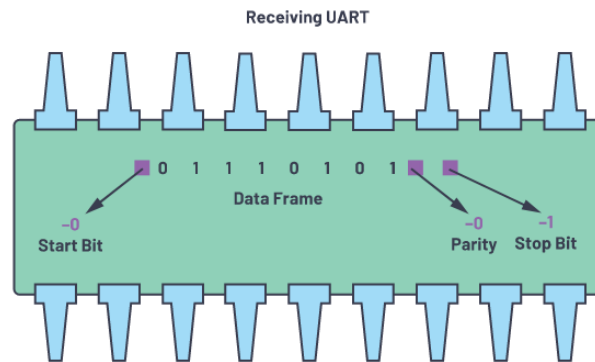
Second: The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame.



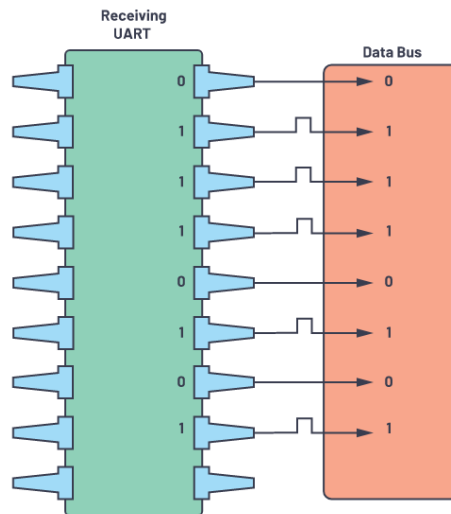
Third: The entire packet is sent serially starting from start bit to stop bit from the transmitting UART to the receiving UART. The receiving UART samples the data line at the preconfigured baud rate.



Fourth: The receiving UART discards the start bit, parity bit, and stop bit from the data frame.



Fifth: The receiving UART converts the serial data back into parallel and transfers it to the data bus on the receiving end.



When properly configured, UART can work with many different types of serial protocols that involve transmitting and receiving serial data.

Sending Temperature Measurement Data from Arduino UNO to NANO using SUART Communication

To interface LM35 temperature sensor with Arduino UNO and for sending the data to nano the components required are:

- LM35 analog temperature sensor
- Arduino UNO
- NANO
- Breadboard
- Jumper wires
- USB cable

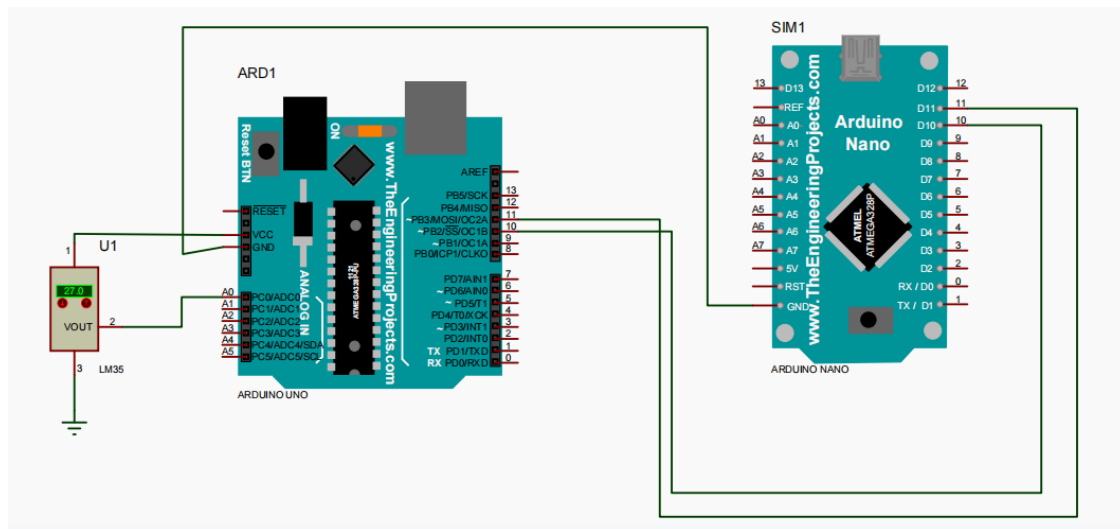


Figure: Connection Diagram

LM 35	Arduino UNO	NANO
Pin 1	5V	
Pin 2	Pin A ₀	
Pin 3	GND	
	10	11
	11	10

Figure: Connection Table

Code

Example 1

Sending LM35 temperature value as string via SUART communication

UNO

```
#include <SoftwareSerial.h>

SoftwareSerial SUART(10, 11);

void setup() {

  Serial.begin(9600);

  SUART.begin(9600);

  Serial.println("UNO");

  analogReference(INTERNAL); //Full Scale of ADC = 1.1V from internal source

  DDRC = 0xF0; // Declaring A0-A3 as input (0000 1111)
```



```
}
```

```
void loop() {  
  float myTemp = 100*(1.1/1023)*analogRead(A0);  
  Serial.println(myTemp, 1);  
  SUART.print(myTemp, 1);  
  delay(1000);
```

NANO

```
#include <SoftwareSerial.h>  
SoftwareSerial SUART(10, 11);  
  
void setup() {  
  Serial.begin(9600);  
  SUART.begin(9600);  
  SUART.println("Nano");  
}  
  
void loop() {  
  if (Serial.available() > 0) {  
    SUART.println(Serial.readString());  
  }  
}
```

Example 2

Sending LM35 temperature value as byte via SUART communication

UNO

```
#include <SoftwareSerial.h>  
SoftwareSerial SUART(10, 11);  
  
void setup() {  
  Serial.begin(9600);  
  SUART.begin(9600);  
  Serial.println("UNO");
```

```

    analogReference(INTERNAL); //Full Scale of ADC = 1.1V from internal source

    DDRC = 0xF0; // Declaring A0-A3 as input (0000 1111)
}

void loop() {
    float myTemp = 100*(1.1/1023)*analogRead(A0);

    Serial.println(myTemp, 1);
    SUART.print(myTemp, 1);

    SUART.write(0x0A);    // for byte transmission
    delay(1000);
}

```

NANO

```

#include <SoftwareSerial.h>

char myTemp[20]; //array of dimension 20

SoftwareSerial SUART(10, 11);

void setup() {
    Serial.begin(9600);

    SUART.begin(9600);

    SUART.println("Nano");
}

void loop() {
    if (Serial.available() > 0) {
        byte m = Serial.readBytesUntil(0x0A, myTemp, 20); //reading bytes

        SUART.println(myTemp);

        memset(myTemp, 0x00, 20); //reset myData[] array to 0s
    }
}

```

LAB NO: 06

LAB Name: I2C communication protocol and interfacing a 16x2 LCD with Arduino UNO.

The I2C (Inter Integrated Circuit) communication bus is very popular and broadly used by many electronic devices because it can be easily implemented in many electronic designs which require communication between a master and multiple slave devices or even multiple master devices. The easy implementations come with the fact that only two wires are required for communication between up to almost 128 (112) devices when using 7 bits addressing and up to almost 1024 (1008) devices when using 10 bits addressing. I2C is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial communication bus. It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short distance, intra-board communication which can be easily controlled with just two general purpose I/O pins and software.

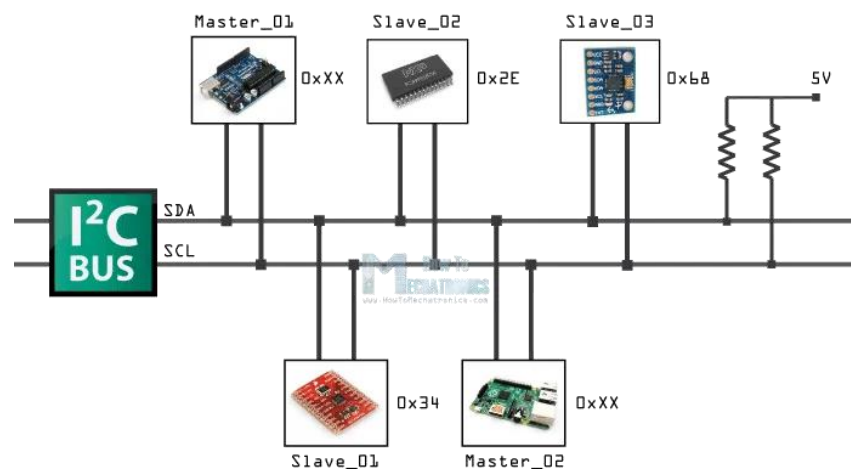
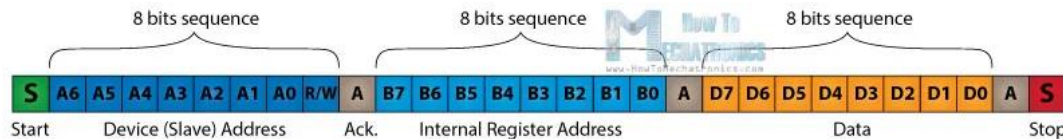


Figure: I2C communication protocol

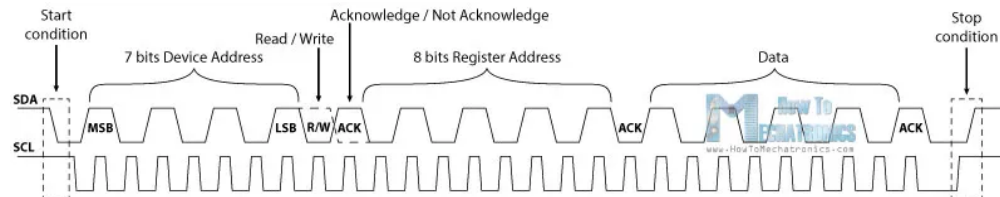
Working of I2C

Each device has a preset ID or a unique device address so the master can choose with which devices will be communicating. The two wires, or lines are called Serial Clock (or SCL) and Serial Data (or SDA). The SCL line is the clock signal which synchronizes the data transfer between the devices on the I2C bus and it's generated by the master device. The other line is the SDA line which carries the data. The two lines are "open-drain" which means that pull up resistors need to be attached to them so that the lines are high because the devices on the I2C bus are active low.

The data signal is transferred in sequences of 8 bits. So after a special start condition occurs comes the first 8 bits sequence which indicates the address of the slave to which the data is being sent. After each 8 bits sequence follows a bit called Acknowledge. After the first Acknowledge bit in most cases comes another addressing sequence but this time for the internal registers of the slave device. Right after the addressing sequences follows the data sequences as many until the data is completely sent and it ends with a special stop condition.



The start condition occurs when data line drops low while the clock line is still high. After this the clock starts and each data bit is transferred during each clock pulse. The device addressing sequence starts with the most significant bit (MSB) first and ends with the least significant bit (LSB) and it's actually composed of 7 bits because the 8th bit is used for indicating whether the master will write to the slave (logic low) or read from it (logic high).

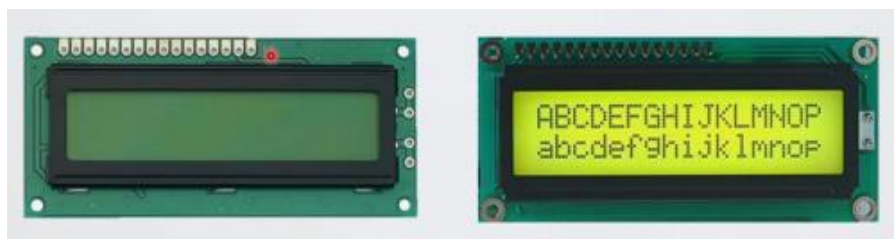


The next bit ACK/ NACK is used by the slave device to indicate whether it has successfully received the previous sequence of bits. So, at this time the master device hands the control of the SDA line over to the slave device and if the slave device has successfully received the previous sequence, it will pull the SDA line down to the condition called Acknowledge. If the slave does not pull the SDA line down, the condition is called Not Acknowledge, and means that it didn't successfully receive the previous sequence which can be caused by several reasons. For example, the slave might be busy, might not understand the received data or command, cannot receive any more data and so on. In such a case the master device decides how it will proceed. Next is the internal registers addressing. The internal registers are locations in the slave's memory containing various information or data. After the addressing, the data transfer sequences begin either from the master or the slave depending of the selected mode at the R/W bit. After the data is completely sent, the transfer will end with a stop condition which occurs when the SDA line goes from low to high while the SCL line is high.

Interfacing 16x2 LCD with Arduino

A **Liquid-Crystal Display (LCD)** is a flat-panel display or other electronically modulated optical device that uses the light-modulating properties of liquid crystals combined with polarizers. Liquid crystals do not emit light directly, instead uses a backlight or reflector to produce images in color or monochrome.

A **16x2** character LCD, for example, has an LED backlight and can display 32 ASCII characters in two rows with 16 characters on each row. There are little rectangles for each character on display and the pixels that make up a character. Each of these rectangles is a grid of 5x8 pixels.



The LCD pinout is as follows

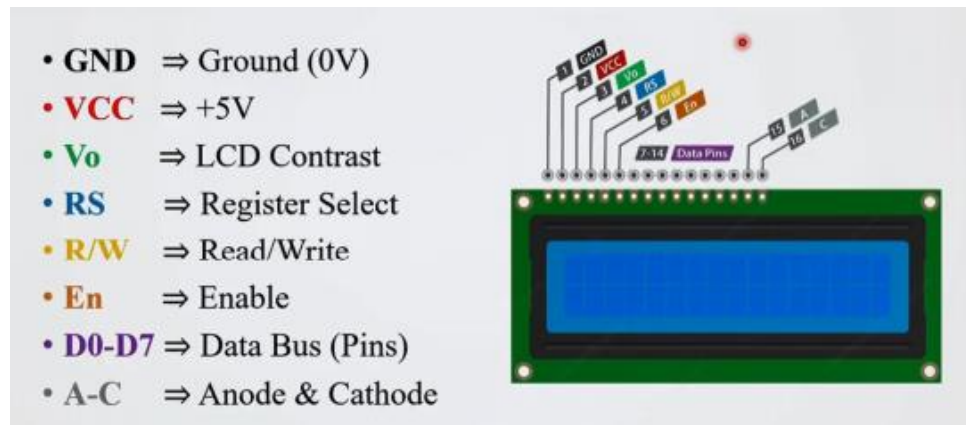
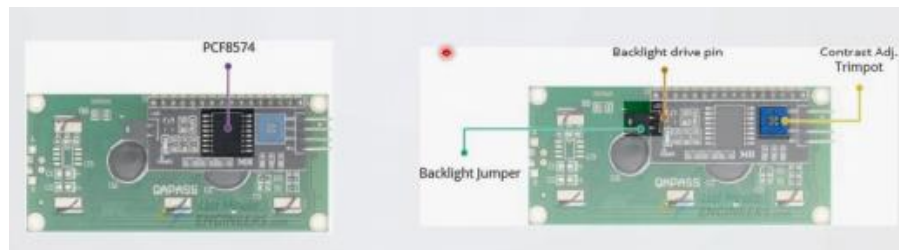


Figure: LCD pinout diagram

To interface Arduino UNO with LCD, I2C LCD adapter can be used. At the heart of the adapter is an 8-bit I/O expander chip-PCF8574. This chip converts the I2C data from an Arduino into the parallel data required by the LCD display. The board also comes with a small trim-pot to make fine adjustments to the contrast of the display.



Two variations of I/O extender are available namely- **PCF8574** & **PCF8574A**. It provides general-purpose remote I/O expansion via the two-wire bidirectional I2C-bus [serial clock (SCL), serial data (SDA)]. Both of these variations are 8-bit I/O expander with interrupt capability.

The PCF8574 and PCF8574A are identical, except for the different fixed portion of the slave address.

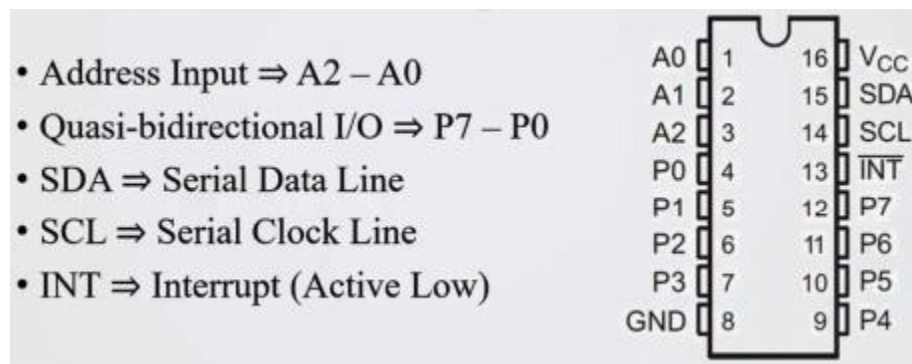
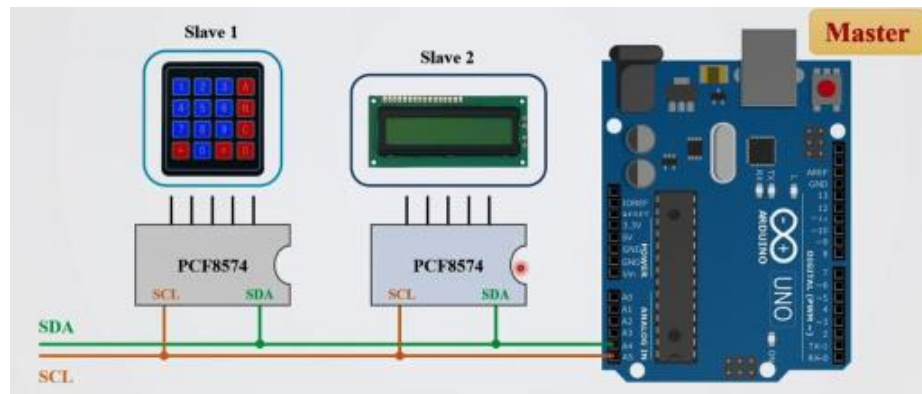


Figure: PCF8574/A pinout

The following diagram shows how the I/O extender can be used for i2C communication:



Device Address

Slave Address is 7-bit (Fixed 4 bit and Variable 3 bit).



For PCF8574, Slave address range would be 20h-27h and for PCF8574A, Slave address range would be 38h-3Fh.

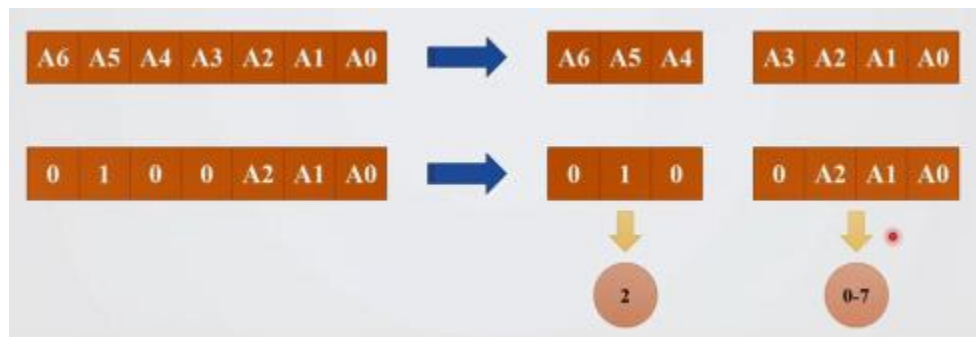


Figure: PCF8574 Device Address

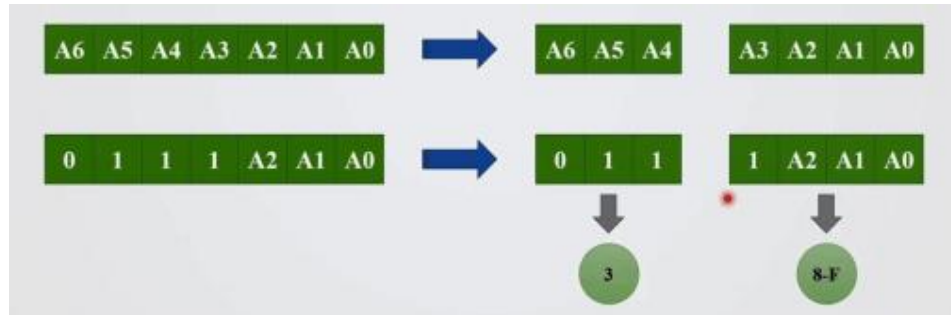


Figure: PCF8574A Device Address

We can identify the Address Maps from the table below

Pin Connectivity			Address Bits							7 bit HEX Address
A2	A1	A0	A6	A5	A4	A3	A2	A1	A0	
GND	GND	GND	0	1	0	0	0	0	0	20h
GND	GND	VCC	0	1	0	0	0	0	1	21h
GND	VCC	GND	0	1	0	0	0	1	0	22h
GND	VCC	VCC	0	1	0	0	0	1	1	23h
VCC	GND	GND	0	1	0	0	1	0	0	24h
VCC	GND	VCC	0	1	0	0	1	0	1	25h
VCC	VCC	GND	0	1	0	0	1	1	0	26h
VCC	VCC	VCC	0	1	0	0	1	1	1	27h

Figure: PCF8574 Address Map

Pin Connectivity			Address Bits							7 bit HEX Address
A2	A1	A0	A6	A5	A4	A3	A2	A1	A0	
GND	GND	GND	0	1	1	1	0	0	0	38h
GND	GND	VCC	0	1	1	1	0	0	1	39h
GND	VCC	GND	0	1	1	1	0	1	0	3Ah
GND	VCC	VCC	0	1	1	1	0	1	1	3Bh
VCC	GND	GND	0	1	1	1	1	0	0	3Ch
VCC	GND	VCC	0	1	1	1	1	0	1	3Dh
VCC	VCC	GND	0	1	1	1	1	1	0	3Eh
VCC	VCC	VCC	0	1	1	1	1	1	1	3Fh

Figure: PCF8574A Address Map

To interface Arduino with LCD the components required are:

- Arduino UNO
- USB Cable
- Jumper Wires
- LCD
- PCF8574/ PCF8574A

- Breadboard

Connection Diagram:

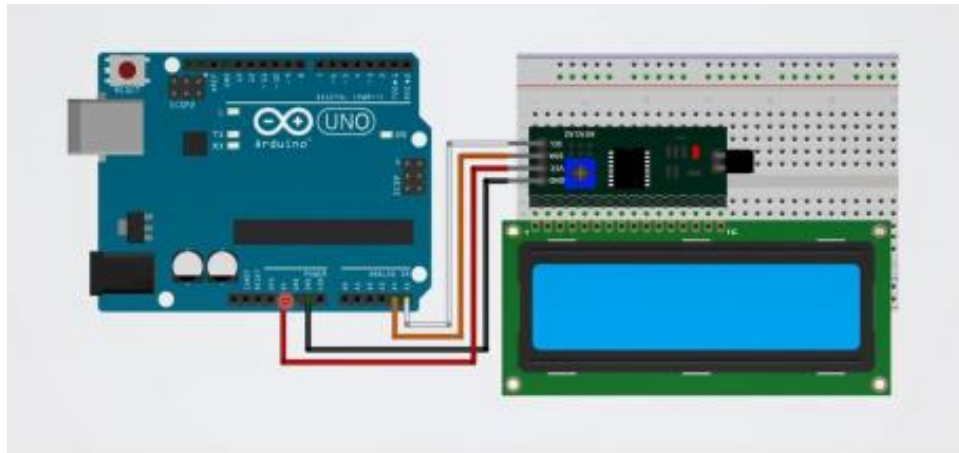


Figure: Connection Diagram

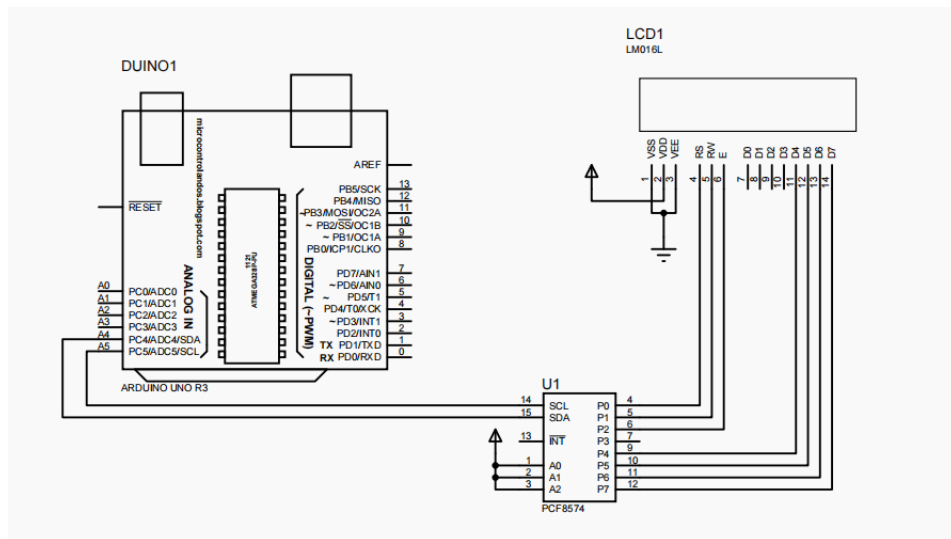


Figure: Connection Diagram (Proteus)

Example 1

Printing 'AUST' on LCD

```
#include<Wire.h> // Library header file; Needed for the I2C controller
```

```
#include <LiquidCrystal_I2C.h> // this header is needed for I2C controller-based LCD
```

```
// Declaring LCD_I2C based Object
```

```
LiquidCrystal_I2C lcd(0x27, 16, 2); //lcd is an object; I2C address, 16 characters, 2 line
```

```
void setup()
```

```
{
```



```

lcd.begin(16, 2); // initializes the 16x2 LCD
}

void loop()
{
  lcd.setCursor(0,0);    //sets the cursor at row 0 column 0
  lcd.print("16x2 LCD MODULE"); // prints 16x2 LCD MODULE
  lcd.setCursor(2,1);    //sets the cursor at row 1 column 2
  lcd.print("AUST"); // prints AUST
}

```

Example 2

Adding two bytes and printing the sum on LCD

```

#include<Wire.h> // Library header file; Needed for the I2C controller
#include <LiquidCrystal_I2C.h> // this header is needed for I2C controller based LCD
// Declaring LCD_I2C based Object
LiquidCrystal_I2C lcd(0x27, 16, 2); //lcd is an object; I2C address, 16 characters, 2 line
byte sum = 0x00; // Variable for storing Sum result

void setup()
{
  // Next 3 lines are for I2C based LCD
  lcd.init();

  lcd.backlight(); //this function brings light at the back of LCD
  lcd.setCursor(0,0); //1st argument refers display block 0; 2nd arg = 0 refers topline
  // Adding two numbers
  byte x1 = 5;
  byte x2 = 6;
  sum = x1 + x2; // result = A7
  // Show result at DP0Dp1 positions of the TopLine of LCD
  lcd.print(sum);
}

```

```
void loop(){
```

```
}
```

I2C Communication

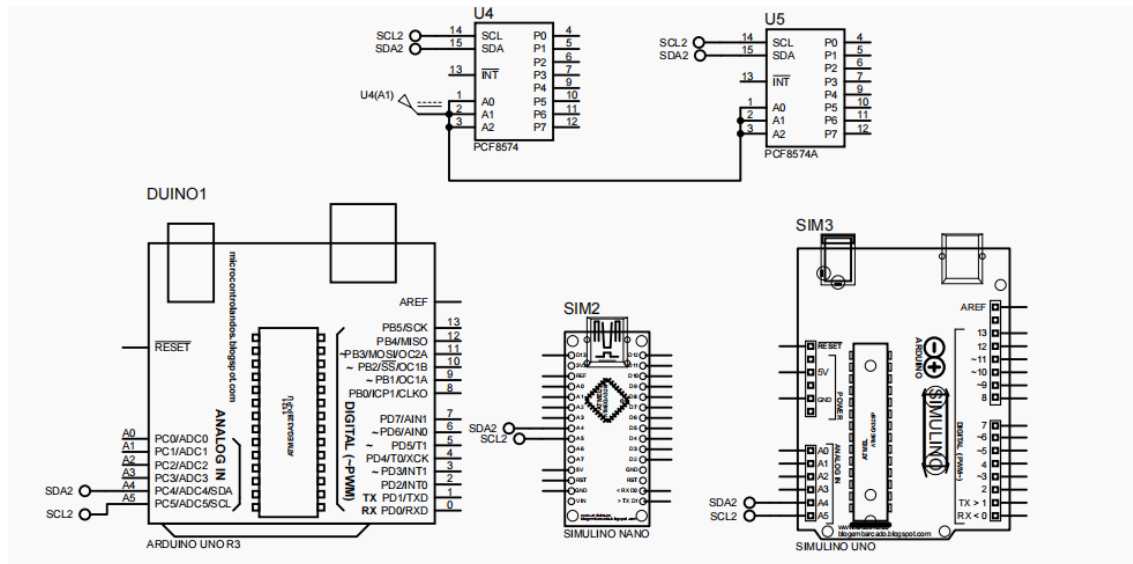


Figure: I2C communication

Example 3

Finding Slave's addresses

```
//slave-1//
#include<Wire.h>

void setup()
{
  Wire.begin(0x13);
}

void loop()
{
}

//slave-2//
```

```

#include<Wire.h>

void setup()
{
    Wire.begin(0x15);
}

void loop()
{
}

//master//

#include <Wire.h>

void setup() {
    Serial.begin(9600);
    Wire.begin();
}

void loop() {
    for (byte i = 0x00; i <= 0x3F; i++){
        Wire.beginTransmission(i);
        if (Wire.endTransmission() == 0){
            Serial.println("Slave address: 0x"+String(i, HEX));
            delay(1000);
        }
    }
    exit(0);
}

```

Example 4

Transmit data (integer) from master to slave

```
//master//
```

```
#include<Wire.h>
```

```
void setup()
```

```
{
```

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

```
  Wire.begin();
```

```
  Wire.beginTransmission(0x13);
```

```
  byte busStatus1 = Wire.endTransmission();
```

```
  if(busStatus1 !=0)
```

```
  {
```

```
    Serial.println("Slave1 is not found...!");
```

```
    while(1);
```

```
  }
```

```
    Serial.println("Slave1 is found.");
```

```
}
```

```
void loop()
```

```
{
```

```
  Wire.beginTransmission(0x13);
```

```
  Wire.write(highByte(0x1234));
```

```
  Wire.write(lowByte(0x1234));
```

```
  Wire.endTransmission();
```

```
  Serial.println(0x1234,HEX);
```

```
  delay(1000);
```

```
}
```

```
//slave//
```

```
#include<Wire.h>
```

```
int p;
```

```
bool flag;
```

```

void setup()
{
    Serial.begin(9600);
    Wire.begin(0x13);
    Wire.onReceive(receiveEvent);
}

void loop()
{
    if(flag == true)
    {
        Serial.println(p, HEX);
        flag = false;
    }
}

void receiveEvent(int howMany)
{
    Serial.println(howMany);
    byte y = Wire.read();
    byte z = Wire.read();
    p = y<<8 | z;
    flag = true;
}

```

Example 5

Transmit data (integer) from slave to master

```

//master//
#include<Wire.h>

```

```

String readData;

```

```

void setup()

```

```

{

```

```

    Serial.begin(9600);

```

```

Wire.begin();

Wire.beginTransmission(0x13);

byte busStatus1 = Wire.endTransmission();

if(busStatus1 !=0)
{
    Serial.println("Slave1 is not found...!");
    while(1);
}

Serial.println("Slave1 is found.");
}

void loop()
{
    readData = "";
    byte m = Wire.requestFrom(0x13, 2);
    while(Wire.available())
    {
        readData = readData+String(Wire.read(),HEX);
        readData.toUpperCase();
    }

    Serial.println(readData);

    delay(1000);
}

```

//slave//

```
#include<Wire.h>
```

```
float a;
```

```
void setup()
```

```
{
```

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

```
    while(!Serial.available()){}
```

```
    a = Serial.parseFloat();
```

```
Wire.begin(0x13);  
Wire.onRequest(sendEvent);  
}
```

```
void loop()  
{  
  
}  
void sendEvent()  
{  
Wire.write(highByte(a));  
}
```

Example 6

Transmit data (string) from slave to master

```
#include<Wire.h>  
  
void setup()  
{  
Serial.begin(9600);  
Wire.begin();  
Wire.beginTransmission(0x13);  
byte busStatus1 = Wire.endTransmission();  
if(busStatus1 !=0)  
{  
Serial.println("Slave1 is not found...!");  
while(1);  
}  
Serial.println("Slave1 is found.");  
}
```

```

void loop()
{
  byte m = Wire.requestFrom(0x13, 6);
  while(Wire.available())
  {
    char c = Wire.read();
    Serial.print(c);
  }
  Serial.println();
  delay(1000);
}

```

Example 7

Transmit data (float) from slave to master

```

//master//
#include<Wire.h>

bool flag;
int p;
char t[20];

void setup()
{
  Serial.begin(9600);
  Wire.begin();
  Wire.beginTransmission(0x13);
  byte busStatus1 = Wire.endTransmission();
  if(busStatus1 !=0)
  {
    Serial.println("Slave1 is not found...!");
    while(1);
  }
  Serial.println("Slave1 is found.");
}

```



```

void loop()
{
    byte m = Wire.requestFrom(0x13, 5);
    for(byte i = 0; i<m; i++)
    {
        t[i] = Wire.read();
    }
    Serial.println(atoi(t),4);
    delay(1000);
}

```

```

//slave//
#include<Wire.h>

float a,b;
char t[10];
void setup()
{
    Serial.begin(9600);
    Serial.println("Enter you value:");
    while(!Serial.available()){ }
    a = Serial.parseFloat();

```

```

    Wire.begin(0x13);
    Wire.onRequest(sendEvent);
    Serial.println(a);
}

```

```

void loop()
{

```

```

    dtostrf(a,3,2,t);
}

void sendEvent()
{
    Wire.write(highByte(t));
}

```

Example 8

Communication among 2 masters and 2 slaves

```

//master_1//
#include<Wire.h>

String readData;

void setup()
{
    Serial.begin(9600);
    Wire.begin();
    Wire.beginTransmission(0x13);
    byte busStatus1 = Wire.endTransmission();
    if(busStatus1 !=0)
    {
        Serial.println("Slave1 is not found...!");
    }
    else{
        Serial.println("Slave1 is found.");
    }
    Wire.beginTransmission(0x15);
    byte busStatus2 = Wire.endTransmission();
    if(busStatus2 !=0)
    {
        Serial.println("Slave2 is not found...!");
    }
}

```

```

    }

    else{

        Serial.println("Slave2 is found.");

    }

    if(busStatus1 | busStatus2){

        while(1);

    }

}

void loop()

{

    dataExchange(0x13);

    dataExchange(0x15);

}

void dataExchange(int x){

    readData = "";

    Wire.beginTransmission(x);

    Wire.write(highByte(0x1234));

    Wire.write(lowByte(0x1234));

    Wire.endTransmission();

    byte m = Wire.requestFrom(x, 2);

    while(Wire.available())

    {

        readData = readData+String(Wire.read(),HEX);

        readData.toUpperCase();

    }

    Serial.println(readData);

    delay(1000);

}

//master_2//

#include<Wire.h>

```

```

String readData;

void setup()
{
    Serial.begin(9600);
    Wire.begin();
    Wire.beginTransmission(0x13);
    byte busStatus1 = Wire.endTransmission();
    if(busStatus1 !=0)
    {
        Serial.println("Slave1 is not found...!");
    }
    else{
        Serial.println("Slave1 is found.");
    }
    Wire.beginTransmission(0x15);
    byte busStatus2 = Wire.endTransmission();
    if(busStatus2 !=0)
    {
        Serial.println("Slave2 is not found...!");
    }
    else{
        Serial.println("Slave2 is found.");
    }
    if(busStatus1 | busStatus2){
        while(1);
    }
}

void loop()
{
    dataExchange(0x15);
    dataExchange(0x13);
}

```

```
}
```

```
void dataExchange(int x){
```

```
    readData = "";
```

```
    Wire.beginTransmission(x);
```

```
    Wire.write(0x43);
```

```
    Wire.write(0x21);
```

```
    Wire.endTransmission();
```

```
    byte m = Wire.requestFrom(x, 2);
```

```
    while(Wire.available())
```

```
    {
```

```
        readData = readData+String(Wire.read(),HEX);
```

```
        readData.toUpperCase();
```

```
    }
```

```
    Serial.println(readData);
```

```
    delay(1000);
```

```
}
```

```
//slave_1//
```

```
#include<Wire.h>
```

```
int p;
```

```
bool flag;
```

```
void setup()
```

```
{
```

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

```
    Wire.begin(0x13);
```

```
    Wire.onReceive(receiveEvent);
```

```
    Wire.onRequest(sendEvent);
```

```
}
```

```
void loop()
```

```
{
```

```
    if(flag == true)
    {
        Serial.println(p, HEX);
        flag = false;
    }
}
```

```
void receiveEvent(int howMany)
{
    byte y = Wire.read();
    byte z = Wire.read();
    p = y<<8 | z;

    flag = true;
}
```

```
void sendEvent()
{
    Wire.write(highByte(0xABCD));
    Wire.write(lowByte(0xABCD));
}
```

```
//slave_2//
#include<Wire.h>

int p;
bool flag;

void setup()
{
    Serial.begin(9600);
    Wire.begin(0x15);
    Wire.onReceive(receiveEvent);
    Wire.onRequest(sendEvent);
}
```

```
void loop()

{
  if(flag == true)
  {
    Serial.println(p, HEX);
    flag = false;
  }
}

void receiveEvent(int howMany)
{
  byte y = Wire.read();
  byte z = Wire.read();
  p = y<<8 | z;
  flag = true;
}

void sendEvent()
{
  Wire.write(highByte(0x5678));
  Wire.write(lowByte(0x8765));
}
```