EXPERIMENT NO. 4

Aim : To Interface button with Arduino UNO and write a program to count the number of times button was pressed & display its value on SSD.

Apparatus Required : Button, SSD(Seven Segment Display), Arduino UNO Board, 330 Ω Resistor.

Theory :

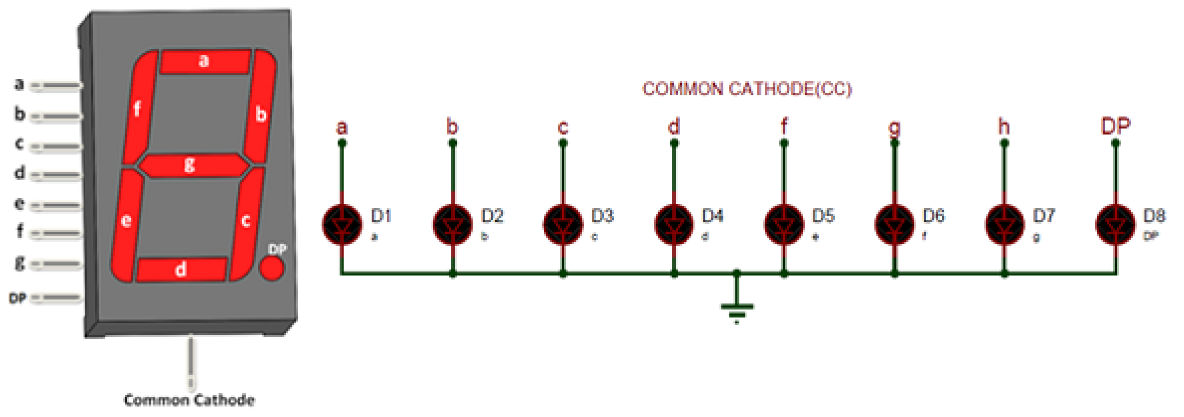
A 7-segment display is commonly used in electronic display devices for decimal numbers from 0 to 9 and in some cases, basic characters. The use of light-emitting diodes (LEDs) in seven-segment displays made it more popular, whereas of late liquid crystal displays (LCD) have also come into use.

Electronic devices like microwave ovens, calculators, washing machines, radios, digital clocks, etc. to display numeric information are the most common applications. Let's take a look at the seven-display pinout to have a better understanding.

A seven-segment display is made of seven different illuminating segments. These are arranged in a way to form numbers and characters by displaying different combinations of segments.

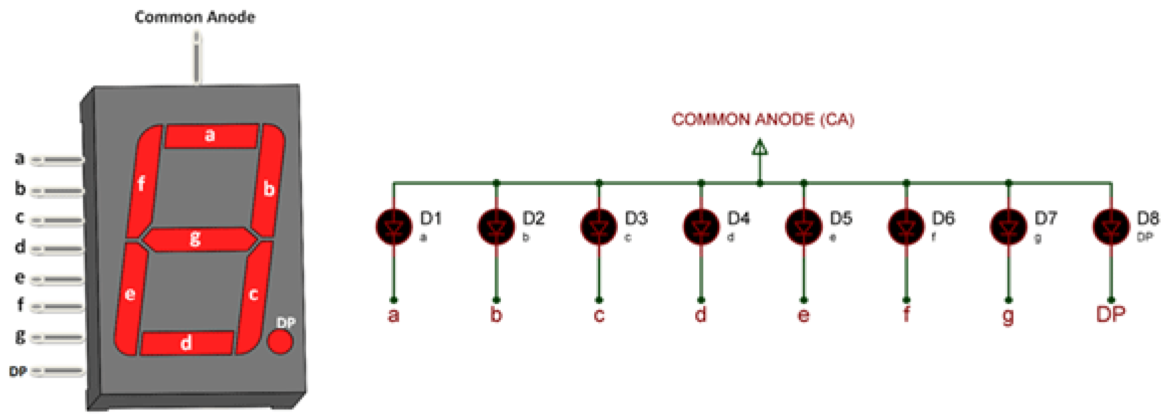
Common Cathode (CC) 7 Segment Display

The common cathode display is commonly called CC display. In this type the common pin on the 7-segment display is connected to all the eight Cathode pins of the LEDs. So In order to make this type of seven segment display to work we should connect he Com pin to the Ground pin and power the other pins with Vcc (+5V typically).



Common Anode (CA) 7 Segment Display

The common anode display is commonly called CA display. In this type the common pin on the **7**-segment display is connected to all the eight Anode pins of the LEDs. So In order to make this type of seven segment display to work we should connect he Com pin to the Vcc (+5V typically) and ground the required segment pin to turn it on.



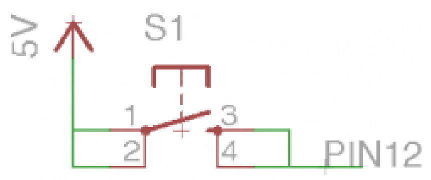
Push Button & Significance of INPUT_PULLUP :

Consider a circuit, which is a simple normally-open push-button on a breadboard. It has been wired so that one side is tied to GND and the other side is connected to Pin 11 of an Arduino Uno
With the following code

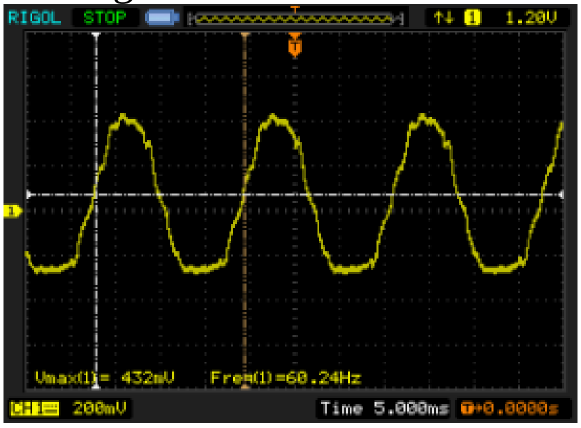
```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(11,INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.println(digitalRead(11));}
```

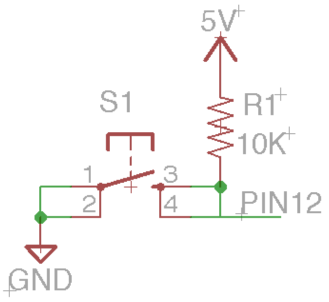
We will find that random garbage values appear on serial monitor and fluctuate even when we bring our fingers closer. The pin is “Floating.” This schematic shows how the button is wired on the breadboard. But PIN 11 is connected to nothing yet the values fluctuate even if the button is unpressed



This means that any signals in the air, such as from nearby electronics, can cause the pin to “float” to either a HIGH or LOW. For example, this waveform from an oscilloscope shows what the pin is doing when nothing is connected.

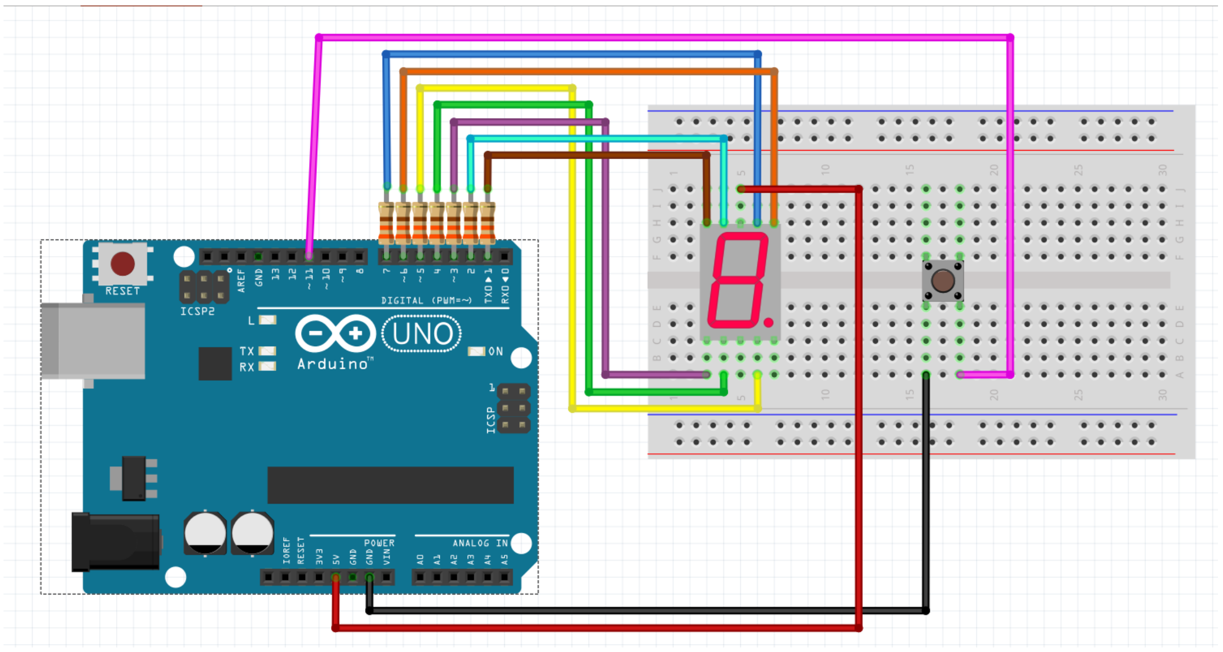


With the oscilloscope waveform visible. As a human finger comes close to the pin, the waveform changes. 60Hz noise from the environment is being coupled into the circuit through the finger, which is what causes the “random” behaviour of the input pin.
The fix for floating pins is to “pull them up” to a known value when the switch is unpressed. This is done with a Pull-Up resistor, as illustrated in the following schematic:



Now when nothing is connected, current cannot flow through the resistor. So, the Voltage on both legs will be the same. This means the same point where the resistor connects to the switch and Pin 11 will be forced to sit at 5V. When the button is pressed, that same point will drop to 0V as current starts flowing through the resistor (ohm's law). Since the resistor is there, you don't have to worry about a short circuit. In this case, 10Kohms was picked as a relatively large value. The exact value of the resistor does not matter. You want something big enough to limit the amount of current wasted when the button is pressed, but not so big (like 10Mohm that you start having noise problems again.) A really cool feature of the ATmega chips used by the Arduino platform is that they have these resistors already built into them. All that needs to be done is turn the Arduino Internal Pull-Up resistor on and you get the previous schematic, for free! Here is the new breadboard circuit. Notice that the red jumper wire has changed to yellow. Also, it is no longer connected to 5V but now is connected to GND.

Circuit Diagram :



Code :

```
int a=7;
int b=6;
int c=5;
int d=4;
int e=3;
int f=2;
int g=1;
int btn=11;
int cc=0;
void setup() {
  // put your setup code here, to run once:
pinMode(a,OUTPUT);
pinMode(b,OUTPUT);
pinMode(c,OUTPUT);
pinMode(d,OUTPUT);
pinMode(e,OUTPUT);
pinMode(f,OUTPUT);
pinMode(g,OUTPUT);
pinMode(btn,INPUT_PULLUP);
//Serial.begin(9600);
}
void loop() {
  // put your main code here, to run repeatedly:
bool btnstate=digitalRead(btn);
if(btnstate==0){
  cc++;
  if(cc==10)
  cc=0;
}
switch(cc){
case 1:
PORTD = B10011110 ;//DISPLAYING 1
break;
case 2:
PORTD = B00100100 ; //DISPLAYING 2
break;
case 3:
PORTD = B00001100 ; //DISPLAYING 3
break;
case 4:
PORTD = B10011000 ; //DISPLAYING 4
break;
case 5:
PORTD = B01001000 ; //DISPLAYING 5
break;
case 6:
PORTD = B01000000 ; //DISPLAYING 6
break;
case 7:
PORTD = B00011110 ; //DISPLAYING 7
break;
case 8:
PORTD = B00000000 ; //DISPLAYING 8
break;
case 9:
PORTD = B00001000 ; //DISPLAYING 9
break;
case 0:
PORTD = B00000010; //DISPLAYING 0
break;
}
//Serial.println(cc);
delay(185);
}
```

Result : Hence, We successfully interfaced button with Arduino UNO and wrote a program to count the number of times button was pressed & displayed its value on SSD.