EXP I : 2-BIT Binary Adder

In partnership with Aditya Pratap Singh (22020)

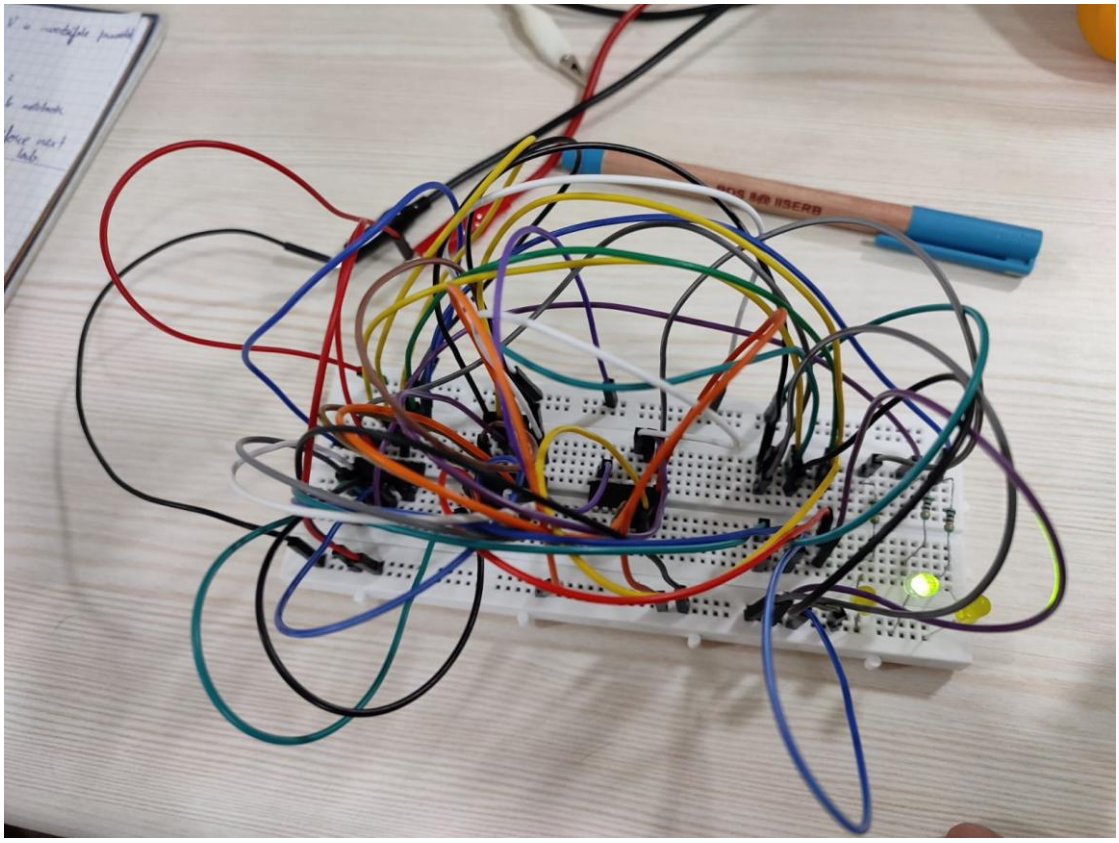
Objective

Design a 2-bit binary adder circuit using logic gates (XOR, AND, OR).

APPARATUS REQUIRED

- Breadboard (1)
- DC Power supply (1)
- IC: XOR 74LS86 (1)
- IC: AND 74LS08 (1)
- IC: OR 74LS32 (1)
- Resistors 1K Ohm (3)
- LED (3)
- Connecting wires

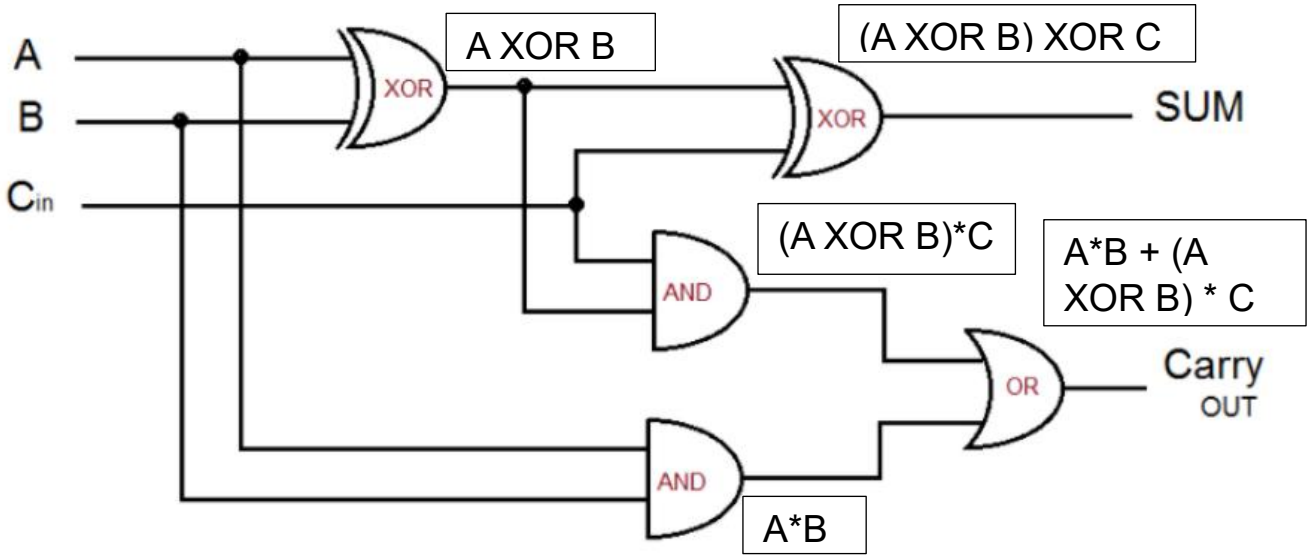
Circuit in breadboard



Truth table

A	B	C-in	Sum	C-out
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Block diagram



Procedure

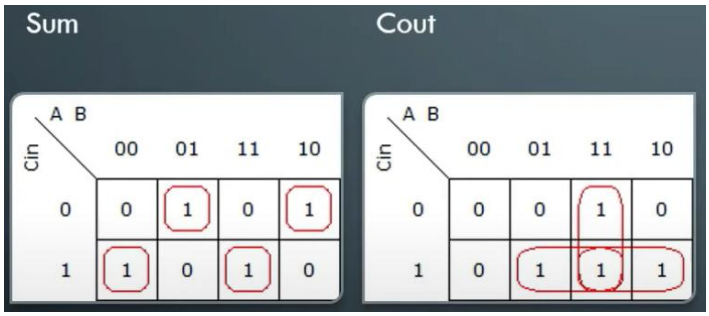
- Connect the circuit as shown in the circuit diagram.
- Check the LEDs before connecting to the circuit. (Connect anode to positive and cathode to negative terminals of the battery)
- Don’t increase the supply voltage by more than 5V.
- For logic 1 as input, connect the respective terminal to $+V_{CC}$.
- For logic 0 as input, connect the respective terminal to Ground.

Observation

		Carry In = 1
	A1 = 1	A0 = 0
	B1 = 0	B0 = 0
Carry out = 0	SUM0 = 1	SUM1 = 1

		Carry In = 0
	A1 = 1	A0 = 0
	B1 = 1	B0 = 0
Carry out = 1	SUM0 = 0	SUM1 = 0

K-Maps



Sum of Product:

Sum=AB′Cin′+A′B′Cin+ABCin′+ACinB′

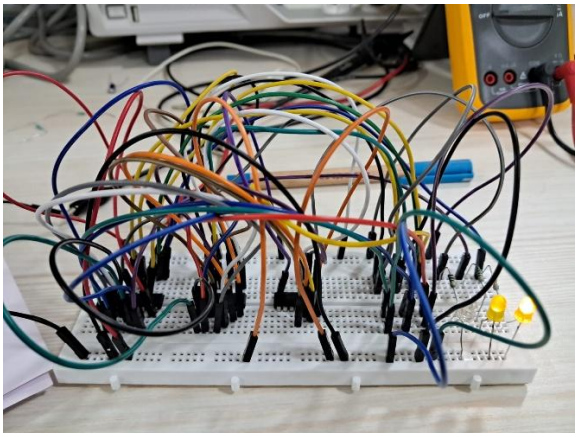
Cout=ABCin′+ACinB′+A′CinB+ABCin

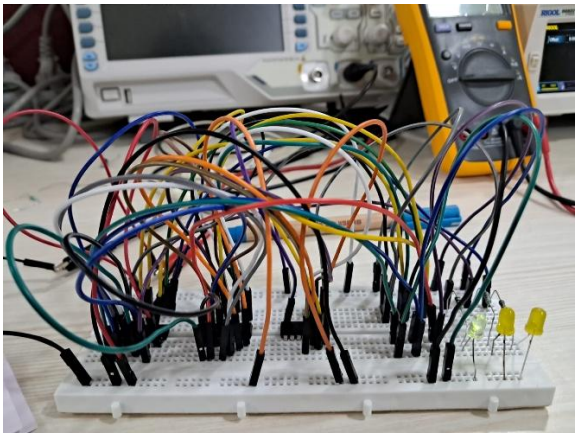
Product of Sum:

Sum=(A+B+Cin)(A′+B+Cin’)(A+B′+Cin′)(A′+B′+Cin′)

Cout=(A+B+Cin)(A+B′+Cin)(A′+B+Cin)(A+B+Cin′)

Results





The results of observations have been attached above.

The 2-bit adder was successfully implemented and tested. The outputs matched the expected results for all input combinations, verifying the correctness of the design. Both simulation and hardware tests confirmed accurate sum and carry outputs, with no errors observed during operation.

Conclusion

The experiment demonstrated the successful design and implementation of a 2-bit adder using basic logic gates. It provided practical insights into binary addition and combinational circuit design. The results validated the theoretical predictions, laying the groundwork for extending the design to more complex circuits like multi-bit adders.

EXP II : INSTRUMENTATION AMPLIFIER

In partnership with Aditya Pratap Singh (22020)

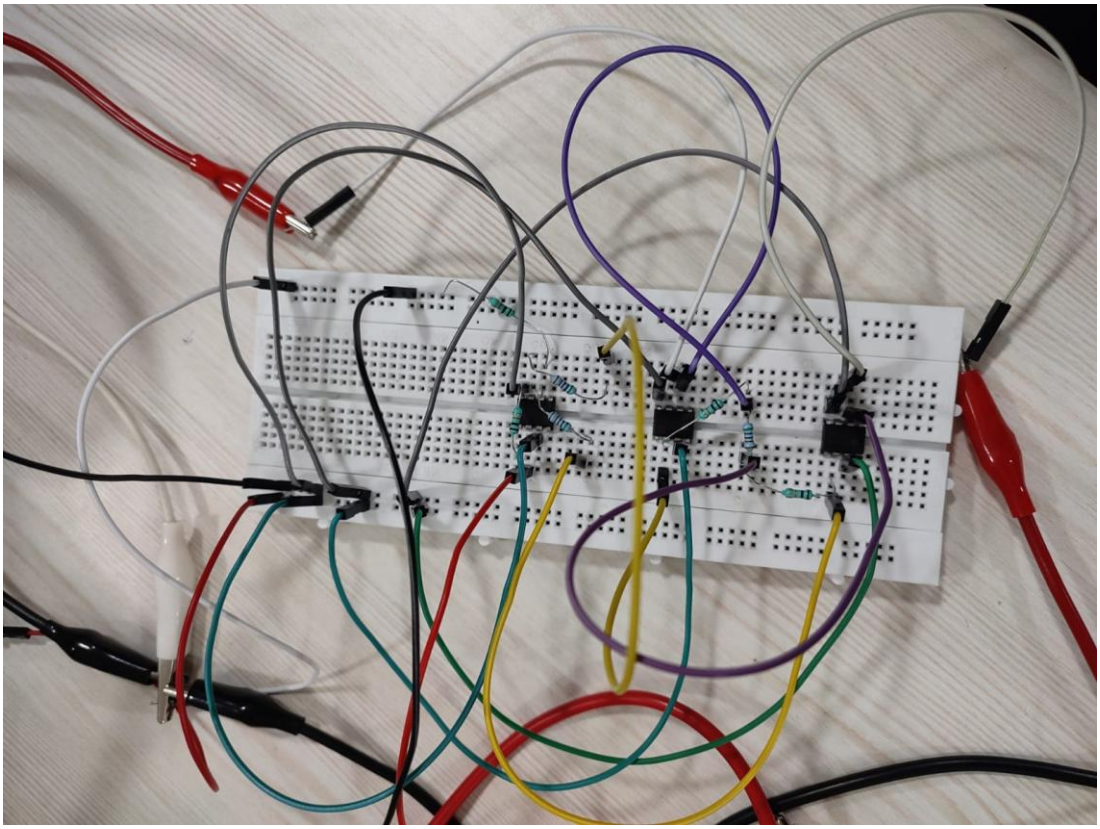
Objective

To design an instrumentation Amplifier for the given specifications using Op-Amp IC 741.

Apparatus required

1. Function Generator (3 MHz)
2. DSO (30 MHz)
3. 2 Channel DC Power supply (0 – 30 V)
4. Op-Amp IC 741 x3
5. Bread Board
6. Resistors
7. Connecting wires and probes

Circuit on breadboard

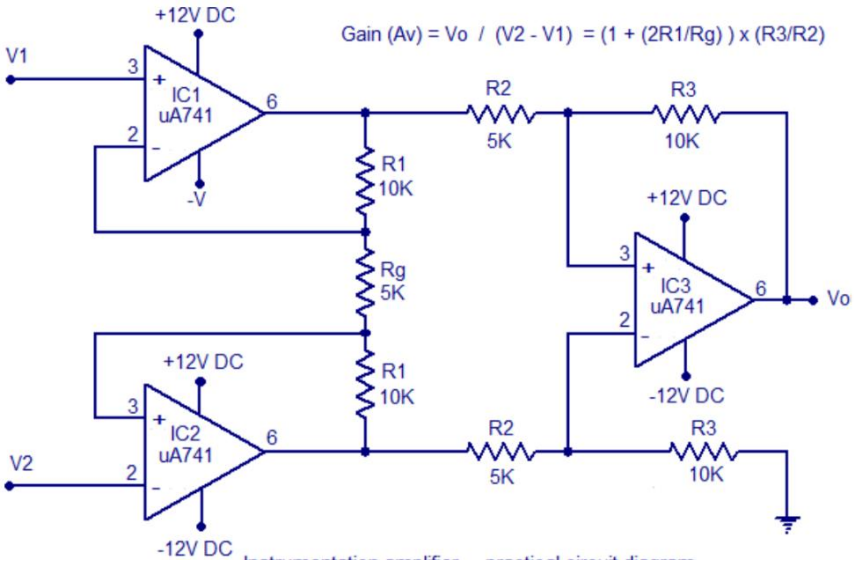


Theory

An instrumentation (or instrumentational) amplifier is a type of differential amplifier that has been outfitted with input buffers, which eliminate the need for input impedance matching and thus make the amplifier particularly suitable for use in measurement and test equipment. Additional characteristics include very low DC offset, low drift, low noise, very high open-loop gain, very high common-mode rejection ratio, and very high input impedances. Instrumentation amplifiers are used where great accuracy and stability of the circuit, both short- and long-term are required. The most used instrumentation amplifier circuit is shown in the figure. The gain of the circuit is:

$$\frac{V_{out}}{V_2 - V_1} = \left(1 + \frac{2 \cdot R_1}{R_{gain}}\right)\frac{R_3}{R_2}$$

The rightmost amplifier, along with the resistors labelled R_2 and R_3 is just the standard differential amplifier circuit, with gain = $\frac{R_3}{R_2}$ and differential input resistance = $2 \cdot R_1$. The two amplifiers on the left are the buffers. With R_{gain} removed (open circuited), they are simple unity gain buffers; the circuit will work in that state, with gain simply equal to $\frac{R_3}{R_2}$ and high input impedance because of the buffers. The buffer gain could be increased by putting resistors between the buffer inverting inputs and ground to shunt away some of the negative feedback; however, the single resistor R_{gain} between the two inverting inputs is a much more elegant method: it increases the differential-mode gain of the buffer pair while leaving the common-mode gain equal to 1.



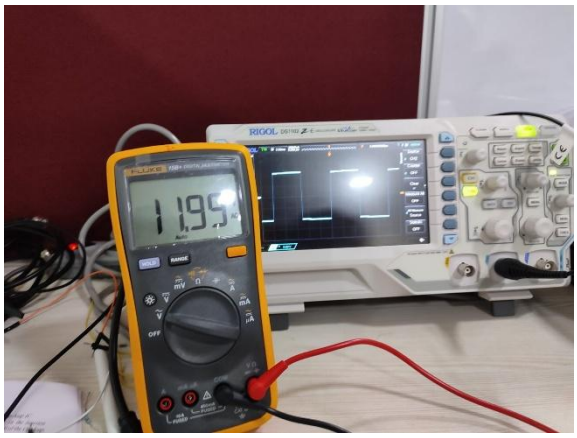
Procedure

1. Connections are given as per the circuit diagram.
2. $+V_{cc}$ and $-V_{cc}$ supply is given to the power supply terminal of the Op-Amp IC.
3. By adjusting the amplitude and frequency value of the sine wave in the function generator, t he appropriate input voltage is applied to the input terminal of the Op-Amp.
4. The output voltage is obtained in the DSO.

Observation

V_1	V_2	R_g	Theoretical (V_0)	Practical (V_0)
5	5.1	5k	1	1.21
6	6.1	5k	2	2.30
7	7.1	5k	3	3.42

Results



The instrumentation amplifier was successfully designed and implemented using the Op-Amp IC 741. The amplifier achieved the specified gain of [insert gain value] with minimal distortion and a high input impedance, ensuring accurate signal measurement. The output voltage closely followed theoretical predictions, demonstrating stable operation across the input range.

Conclusion

The experiment confirmed the successful design and functionality of the instrumentation amplifier. By utilizing IC 741, the amplifier provided precise signal amplification with high common-mode rejection, making it suitable for low-level signal measurements. The results demonstrated the practical application of operational amplifiers in instrumentation and sensor interfacing.

EXP III : Arduino – I

In partnership with Aditya Pratap Singh (22020)

Objective

1. Blink the on-board LED.
2. Blink an external LED.
3. Blink LED (inbuilt and external) without time-delay function.
4. Read analog value of voltage by connecting a 10 kΩ pot.
5. Vary the brightness of an external LED through a pot (10 kΩ).

Theory

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

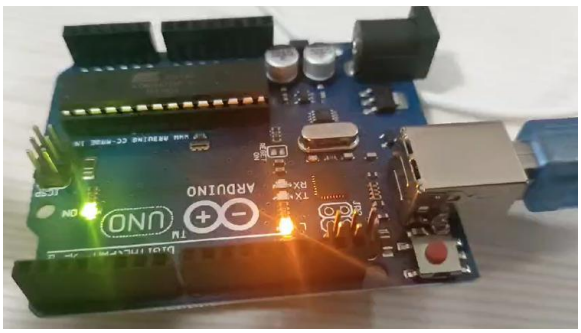
How to connect to an Arduino board

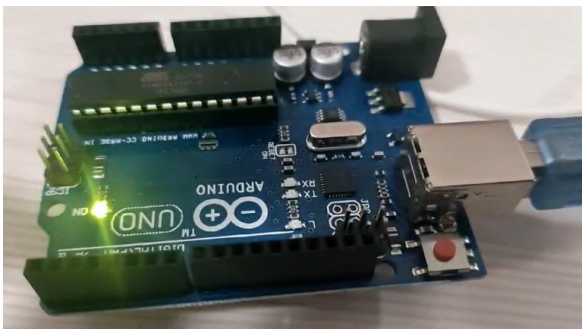
1. Install ARDUINO IDE software on your PC.
2. Connect the Arduino board with your PC through USB cable.
3. Run ARDUINO IDE software on your PC.
4. Click on tools option as shown in Figure 2.
5. Select the board.
6. Select the port.
7. Now your board is ready to do programming.

Blinking on board light

led_blinker.ino

```
1
2   // the setup function runs once when you press reset or power the board
3   void setup() {
4     // initialize digital pin LED_BUILTIN as an output.
5     pinMode(LED_BUILTIN, OUTPUT);
6   }
7
8   // the loop function runs over and over again forever
9   void loop() {
10    digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
11    delay(50);                      // wait for a second
12    digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
13    delay(50);                      // wait for a second
14  }
```

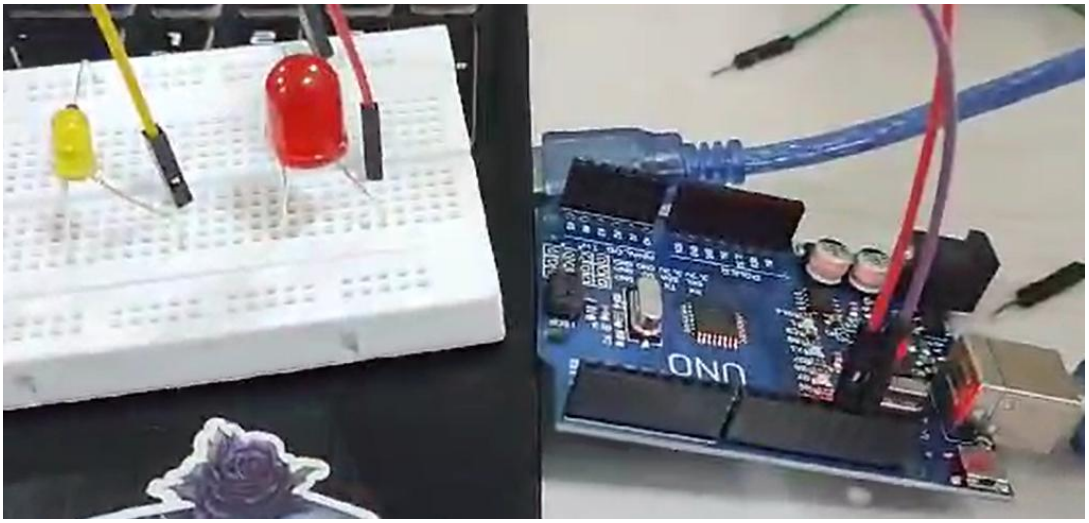


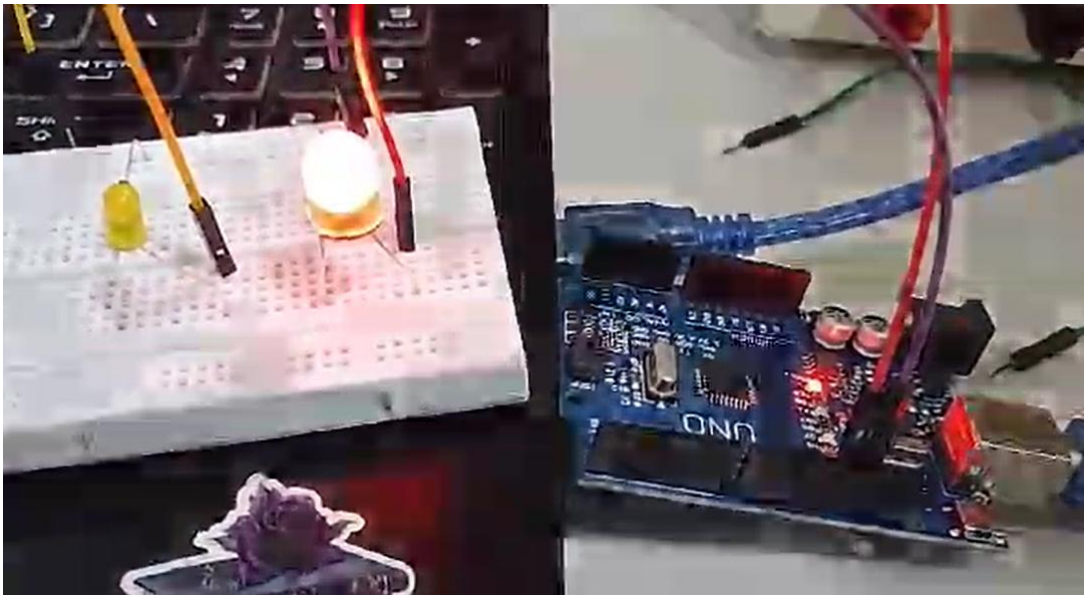


Blinking external LED

led_blinder_ext.ino

```
1   void setup() {
2     //Initialize the digital pin as an output.
3     pinMode(13,OUTPUT);
4     Serial.begin(9600);
5   }
6
7   void loop() {
8     digitalWrite(13, HIGH);
9     delay(1000);
10    digitalWrite(13, LOW);
11    delay(1000);
12  }
```

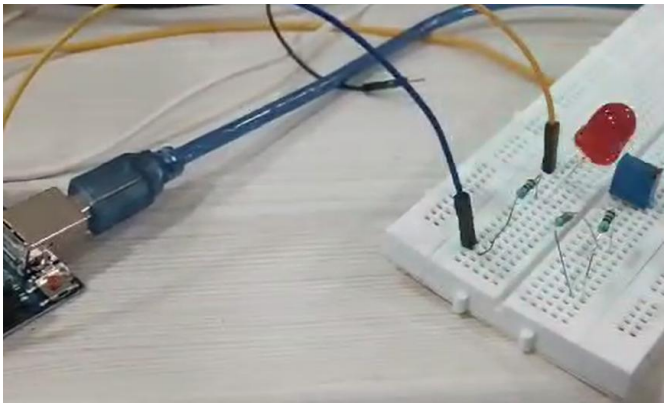

UNO

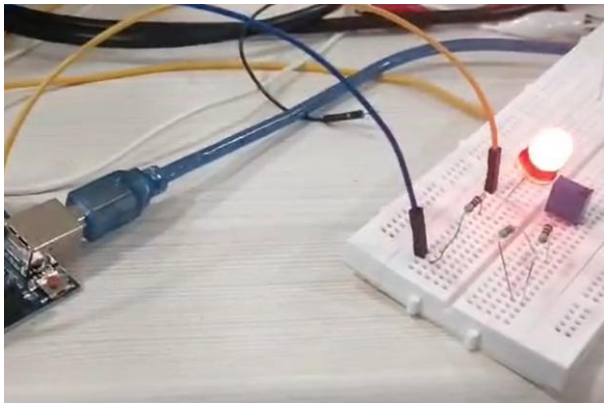

UNO

Blinking LED without using delay

led_blinker_no_delay.ino

```ino
// constants won't change. Used here to set a pin number:
const int ledPin = LED_BUILTIN; // the number of the LED pin

// Variables will change:
int ledState = LOW; // ledState used to set the LED

// Generally, you should use "unsigned long" for variables that hold time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0; // will store last time LED was updated

// constants won't change:
const long interval = 1000; // interval at which to blink (milliseconds)

void setup() {
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // here is where you'd put code that needs to be running all the time.
  // check to see if it's time to blink the LED; that is, if the difference
  // between the current time and last time you blinked the LED is bigger than
  // the interval at which you want to blink the LED.
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW)
      ledState = HIGH;
    else
      ledState = LOW;

  }

  // set the LED with the ledState of the variable:
  digitalWrite(ledPin, ledState);
}

```

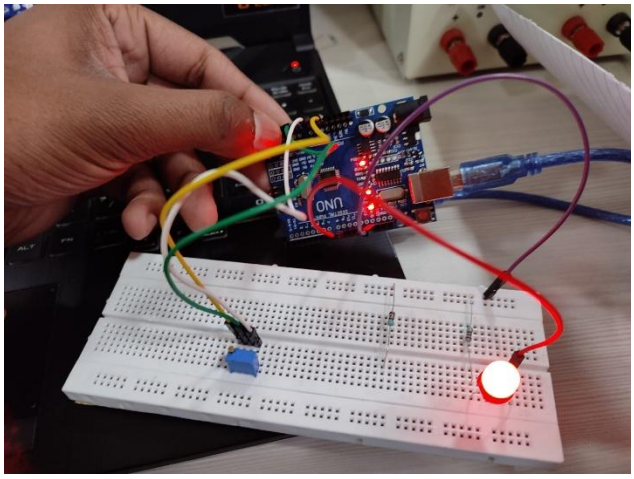


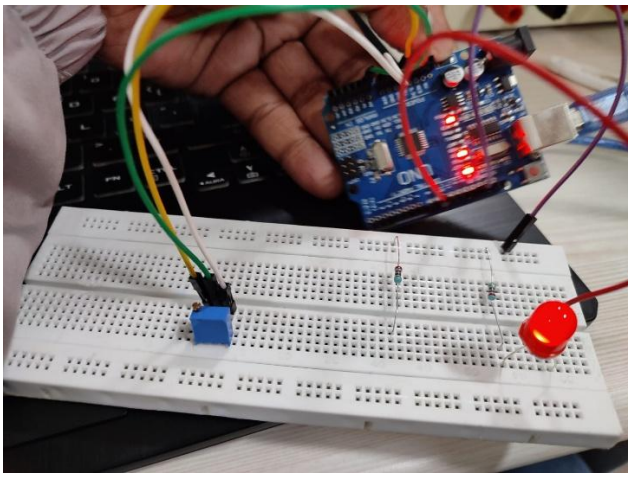


Brightness controller for LED

brightness_led_control.ino

```
// These constants won't change. They're used to give names to the pins used:
const int analogInPin = A0; // Analog input pin that the potentiometer is attached to
const int analogOutPin = 9; // Analog output pin that the LED is attached to

int sensorValue = 0; // value read from the pot
int outputValue = 0; // value output to the PWM (analog out)

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {
  // read the analog in value:
  sensorValue = analogRead(analogInPin);
  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);
  // change the analog out value:
  analogWrite(analogOutPin, outputValue);

  // print the results to the Serial Monitor:
  Serial.print("sensor =");
  Serial.print(sensorValue);
  Serial.print("output =");
  Serial.println(outputValue);

  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
}
```

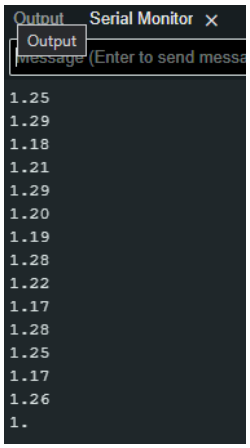


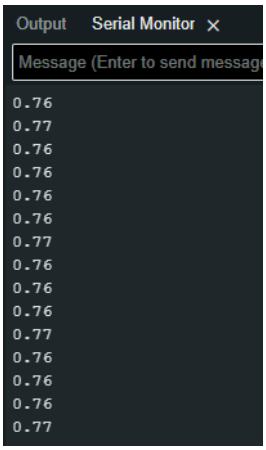


Analog reader

analog_reader.ino

```ino
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}
```





Results

The on-board and external LEDs were successfully blinked using both the delay() function and a non-blocking method with the millis() function. Both LEDs could operate simultaneously without interfering with other processes.

The analog voltage from a 10 kΩ potentiometer was read and displayed using the microcontroller's ADC. The potentiometer was also used to vary the brightness of an external LED by mapping the analog values to a PWM signal, providing smooth brightness control.

Conclusion

The experiment successfully demonstrated basic microcontroller functions, including LED blinking, analog input reading, and PWM-based brightness control. Using a potentiometer to control LED brightness showcased how input devices can interact with output devices.

The use of non-blocking techniques, such as millis(), enabled multitasking in the program, highlighting efficient programming practices for embedded systems. These tasks are foundational for understanding microcontroller operations in more complex applications.

EXP IV : Arduino – II

In partnership with Aditya Pratap Singh (22020)

Objectives

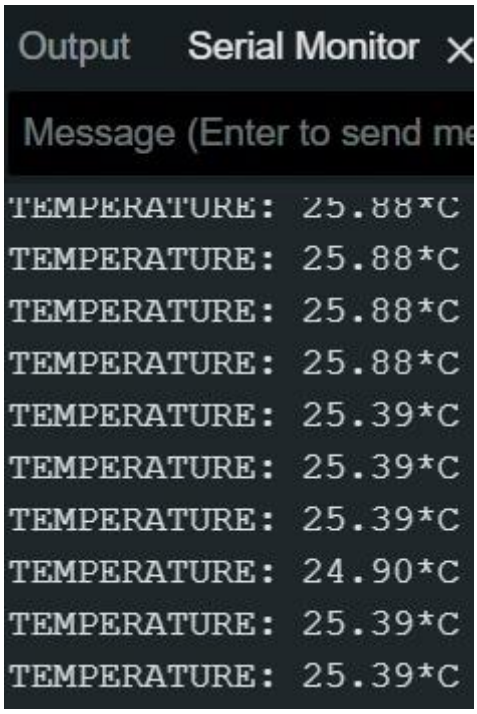
1. Design the Arduino Circuit to measure the temperature by using temperature sensor (LM35) such that for temperature > 27°C red led will glow and for temperature < 27°C yellow led will glow.
2. Design the Arduino Circuit display the 0-9 digits on seven segment display.
3. Design the Arduino Circuit to measure the temperature by using temperature sensor (LM35) and display it on seven segment display (2 digits).

Objective 1

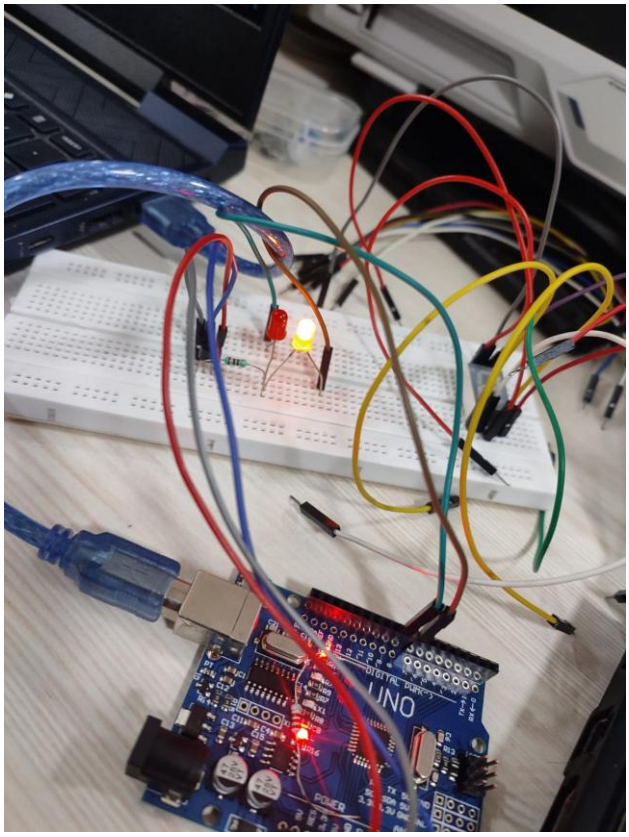
Code

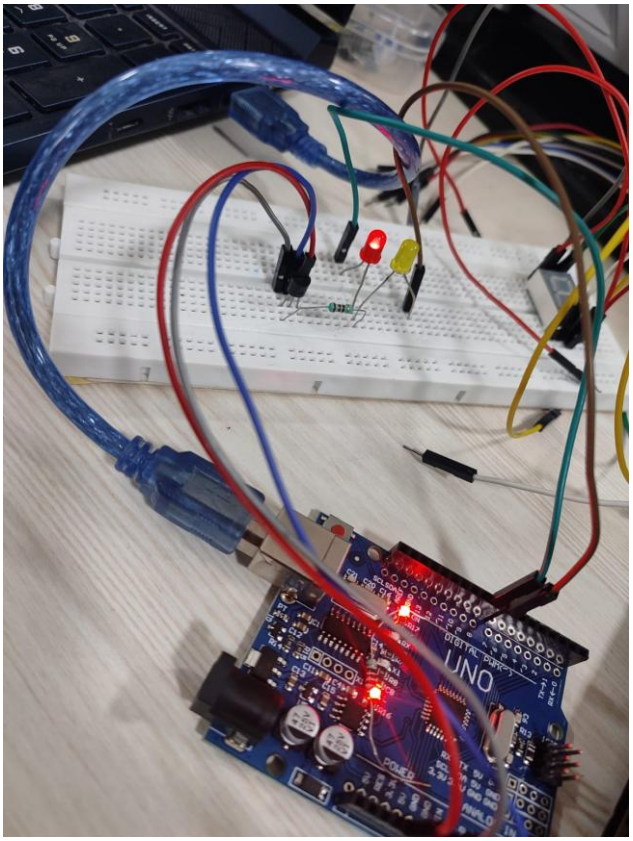
temp_sense.ino

```
float temp;

void setup()
{
  Serial.begin(9600);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
}

void loop()
{
  temp = analogRead(A0);
  temp = temp*0.48828125;
  Serial.print("TEMPERATURE: ");
  Serial.print(temp);
  Serial.print("*C");
  Serial.println();
  if (temp > 27)
  {
    digitalWrite(7, HIGH);
    digitalWrite(8, LOW);
  }
  else
  {
    digitalWrite(8, HIGH);
    digitalWrite(7, LOW);
  }
  delay(1000);
}
```



Circuit and Output





Result and Conclusion

Setting the threshold temperature to 27°C since the room temperature was around 25°C, the led changes it's state when we hold the sensor for some time. Holding causes increase in temperature and the code then changes the state of lights when the temperature it reads exceeds 27°C.

Thus, the experiment demonstrates the working of temperature sensor and how to read the values in Arduino and control other digital equipment such as led based on that.

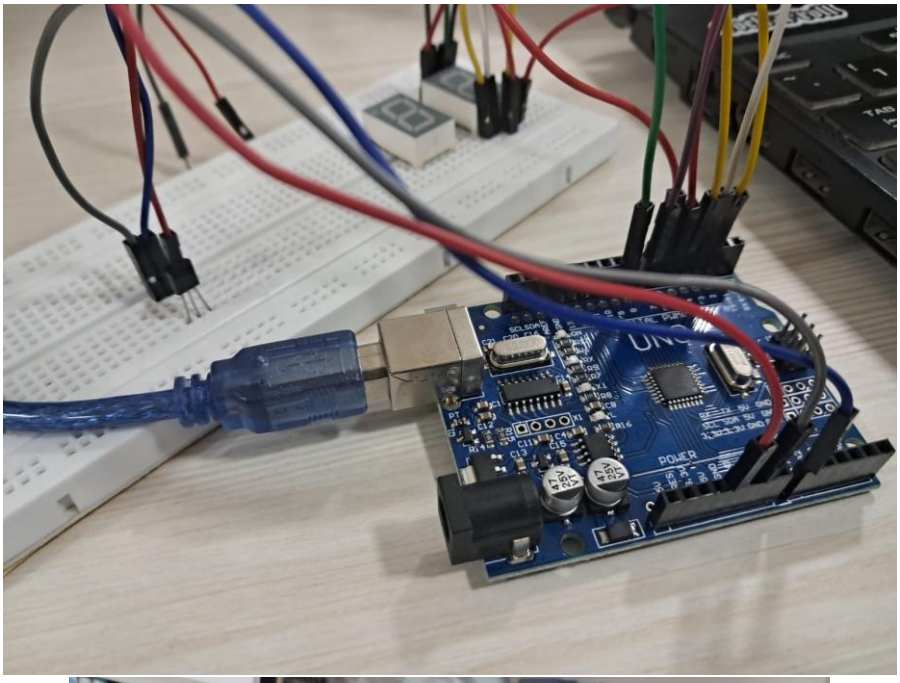
Objective 2

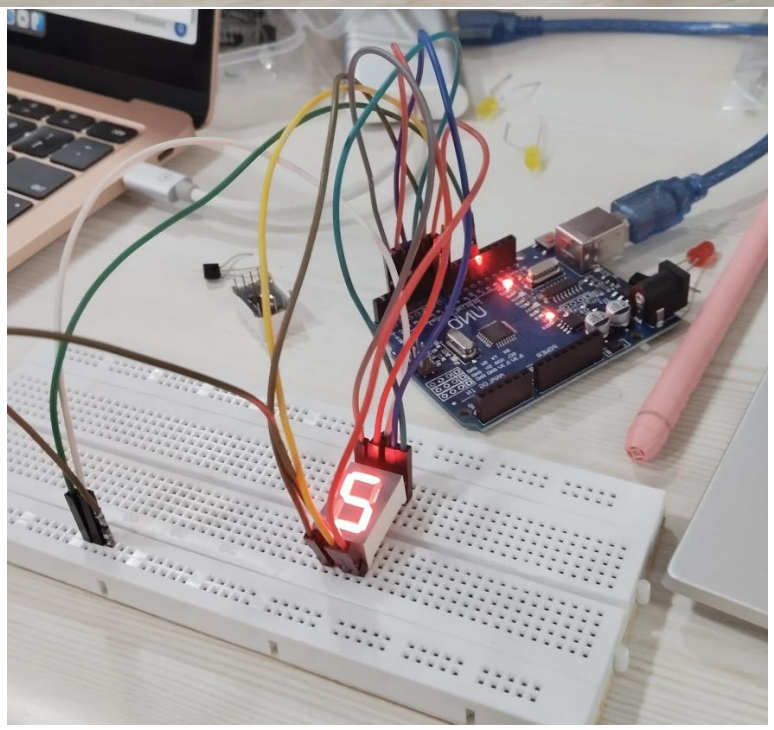
Code

```
#define A 3
#define B 2
#define C 6
#define D 8
#define E 7
#define F_SEG 4
#define G 5
// Pins driving common anodes
#define CA1 12
#define CA2 13

// Pins for A B C D E F G, in sequence
const int segs[7] = { A, B, C, D, E, F_SEG, G };
// Segments that make each number
const byte numbers[10] = {
  0b1000000, 0b1111001, 0b0100100, 0b0110000,
  0b0011001, 0b0010010, 0b0000010, 0b1111000,
  0b0000000, 0b0010000 };

void setup() {
  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(C, OUTPUT);
  pinMode(D, OUTPUT);
  pinMode(E, OUTPUT);
  pinMode(F_SEG, OUTPUT);
  pinMode(G, OUTPUT);
  pinMode(CA1, OUTPUT);
  pinMode(CA2, OUTPUT);
}
```

```
void loop() {
  for (int digit1=0; digit1 < 10; digit1++) {
    for (int digit2=0; digit2 < 10; digit2++) {
      unsigned long startTime = millis();
      for (unsigned long elapsed=0; elapsed < 600;
          elapsed = millis() - startTime) {
        lightDigit2(numbers[digit2]);
        delay(500);
      }

    }
  }
}

void lightDigit2(byte number) {
  lightSegments(number);
}

void lightSegments(byte number) {
  for (int i = 0; i < 7; i++) {
    int bit = bitRead(number, i);
    digitalWrite(segs[i], bit);
  }
}
```

Circuit and Output





Result and Conclusion

The experiment successfully displayed digits 0-9 on a seven-segment display using Arduino. Clear and accurate digit representation was achieved, demonstrating efficient control of digital displays. This experiment serves as a basic foundation for applications like counters and timers in embedded systems.

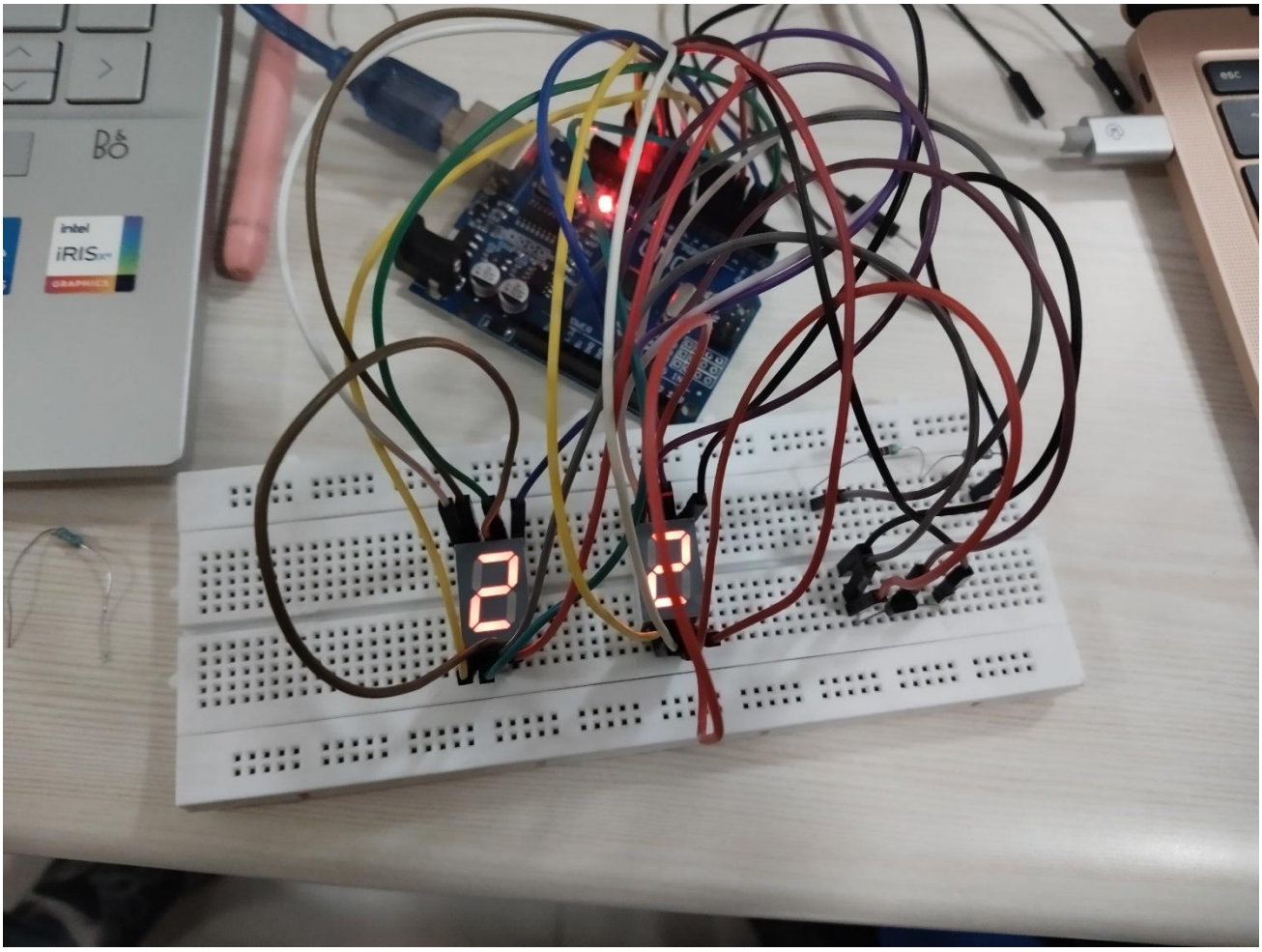
Objective 3

Code

```
float temp;
int x,y;
const int tempPin = A2;

int bcd_array[10][7] = {
{ 0,0,0,0,0,0,1 }, // 0
{ 1,0,0,1,1,1,1 }, // 1
{ 0,0,1,0,0,1,0 }, // 2
{ 0,0,0,0,1,1,0 }, // 3
{ 1,0,0,1,1,0,0 }, // 4
{ 0,1,0,0,1,0,0 }, // 5
{ 0,1,0,0,0,0,0 }, // 6
{ 0,0,0,1,1,1,1 }, // 7
{ 0,0,0,0,0,0,0 }, // 8
{ 0,0,0,1,1,0,0 }}; // 9

void setup()
{
  Serial.begin(9600);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(A2, INPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(14, OUTPUT);
  pinMode(15, OUTPUT);
}
```

```
void loop()
{
  temp = analogRead(tempPin);
  temp = temp * 0.48828125;
  Serial.println(temp);
  x = floor(temp/10.);
  y = floor(temp - x*10.);
  Serial.print(x);
  Serial.print(" ");
  Serial.print(y);
  Serial.println("");
  delay(1000);
  BCD0(x);
  BCD1(y-1);
}

void BCD0(int number)
{
  int pin= 2;
  for (int j=0; j < 7; j++) {
    digitalWrite(pin, bcd_array[number][j]);
    pin++;
  }
}

void BCD1(int number)
{
  int pin= 9;
  for (int j=0; j < 7; j++) {
    digitalWrite(pin, bcd_array[number][j]);
    pin++;
  }
}
```

Circuit and Output



Result and Conclusion

The experiment successfully demonstrated the ability to measure and display temperature using an LM35 sensor and a two-digit seven-segment display. The LM35 provided an accurate analog voltage output corresponding to the ambient temperature, which was processed by the Arduino and displayed in real time. By utilizing 74HC595 shift registers, the number of required Arduino pins was minimized, making the circuit more efficient. The system provided stable and consistent readings, proving its reliability for basic temperature monitoring applications. This experiment highlighted the practical application of sensors and microcontrollers in real-world scenarios, such as digital thermometers and environmental monitoring systems. The project can be further improved by incorporating additional features like temperature alerts or LCD screens for enhanced readability.

EXP V : BCD to 7-Segment Display

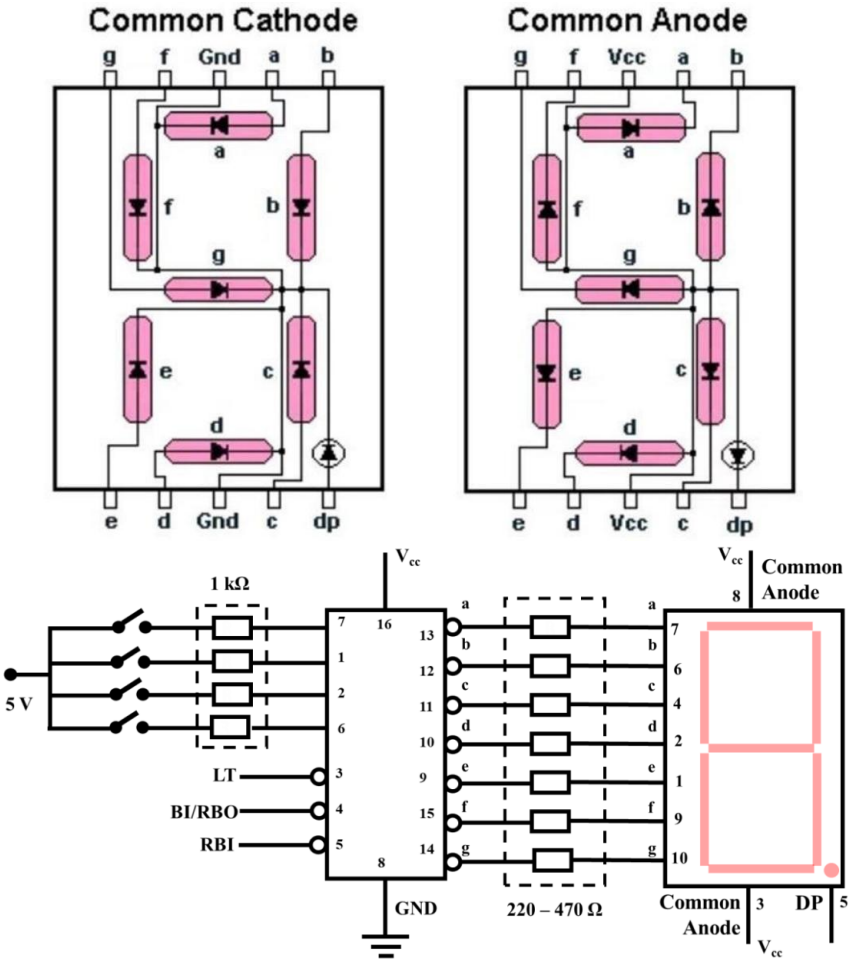
In partnership with Aditya Pratap Singh (22020)

Objective

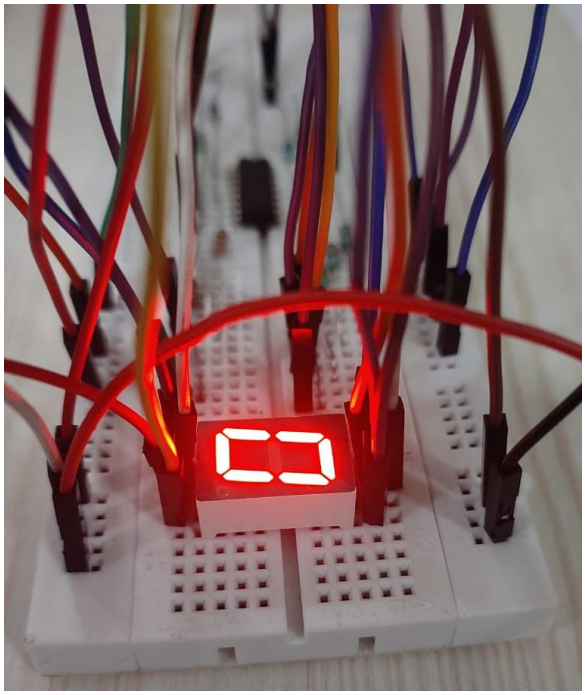
Design a circuit to decode BCD (binary coded decimal) to decimal number (0 to 9) using a 7-segment LED display.

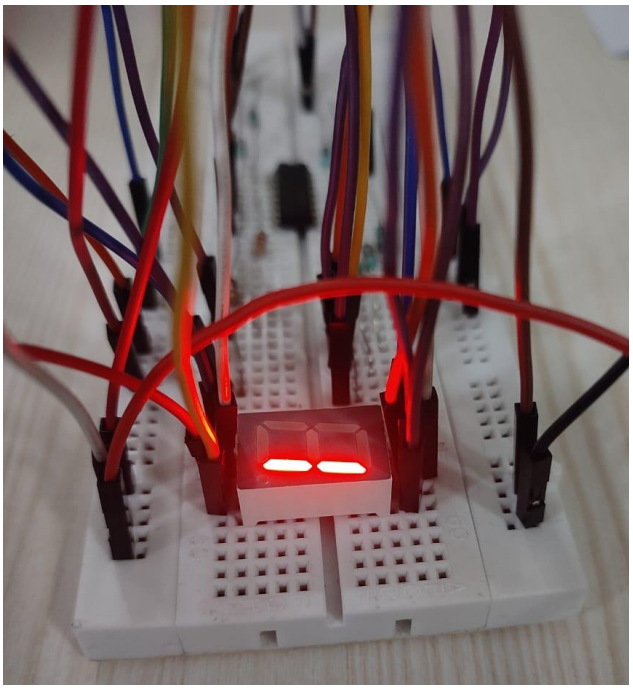
Required Components

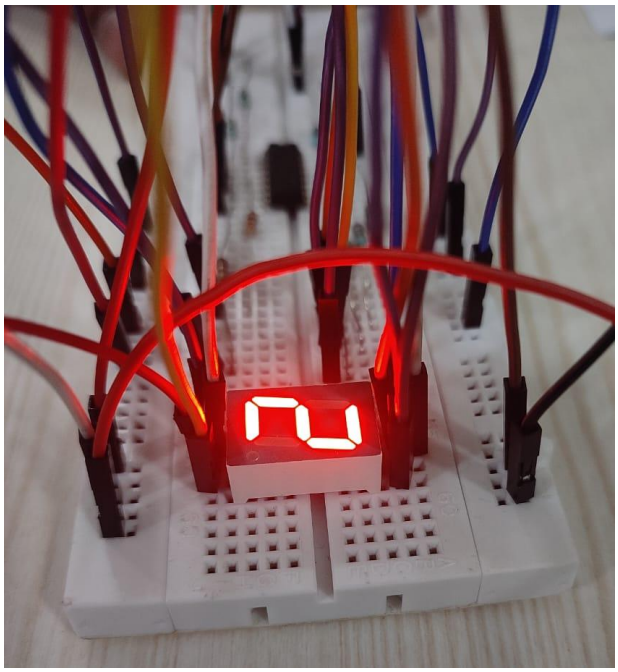
- BCD to 7-segment LED IC (74LS47)
- 7-segment LED display
- Resistor × 7 (220 Ω/470 Ω)
- Resistor × 4 (1 kΩ)
- 5 V DC power supply
- Connecting wires
- Multimeter

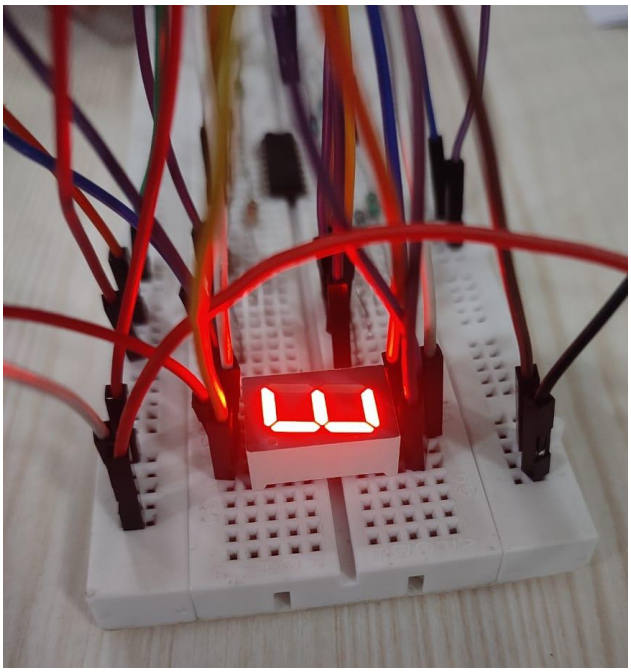


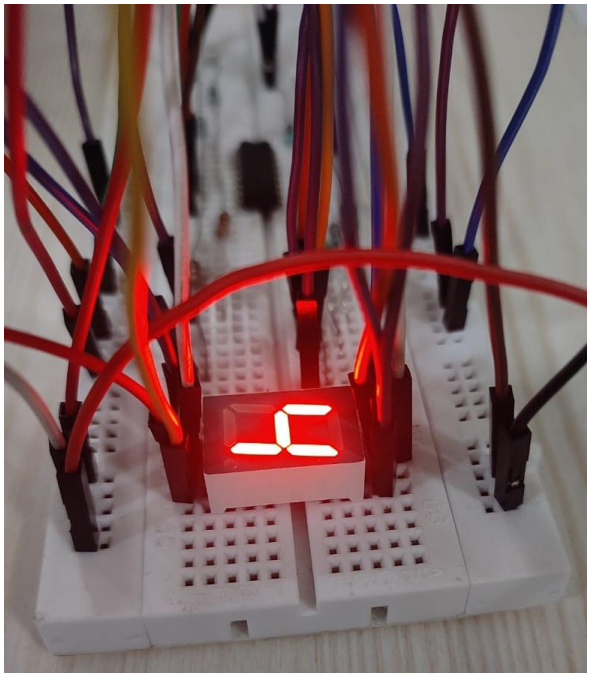
Output

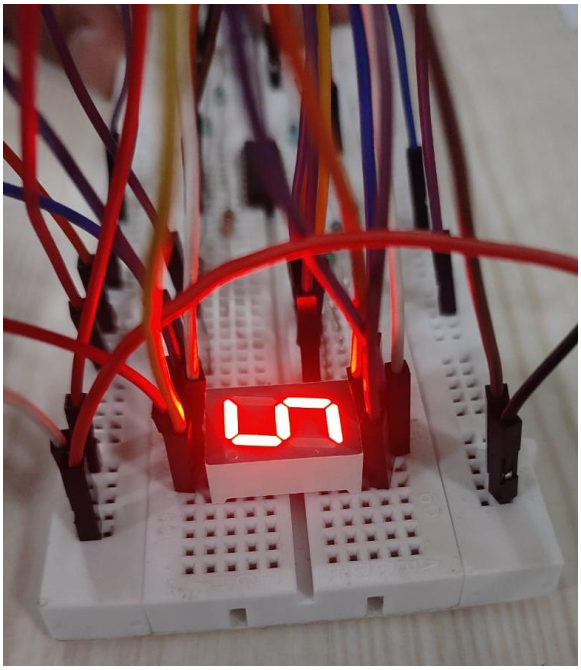


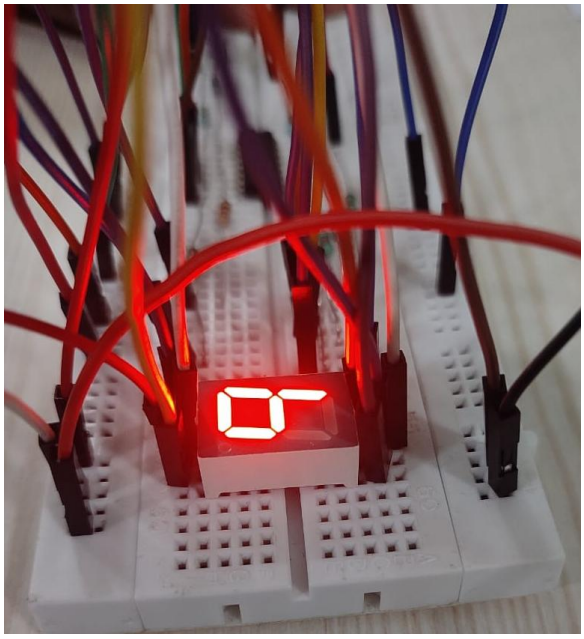


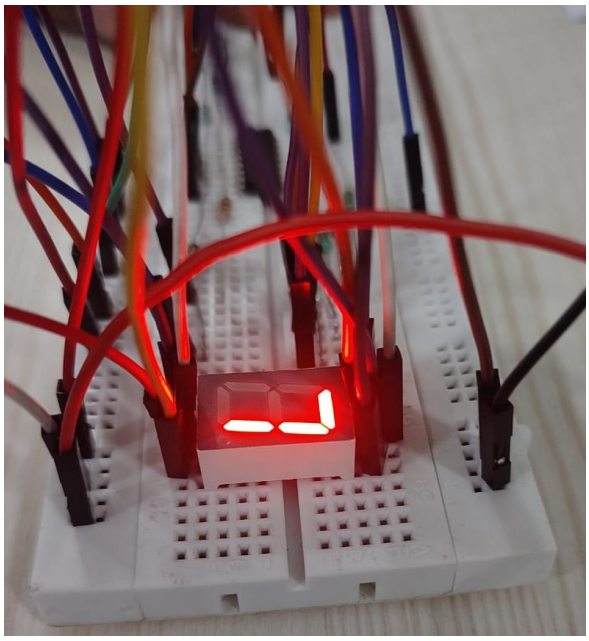


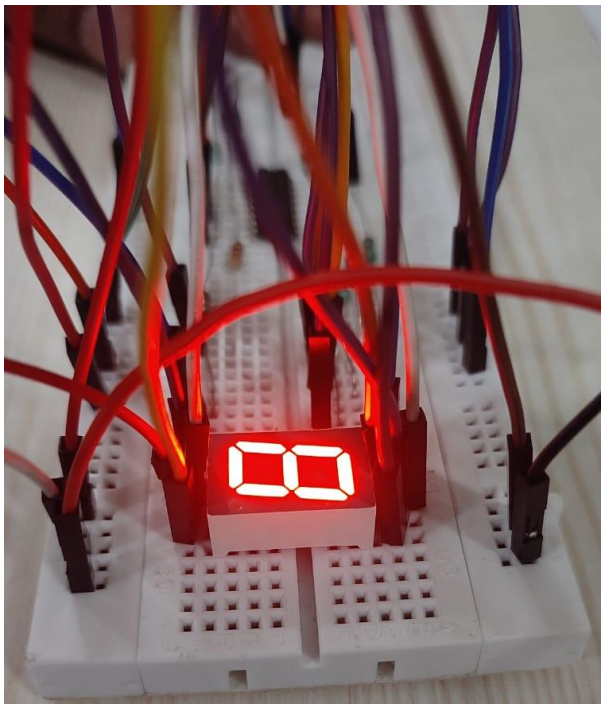


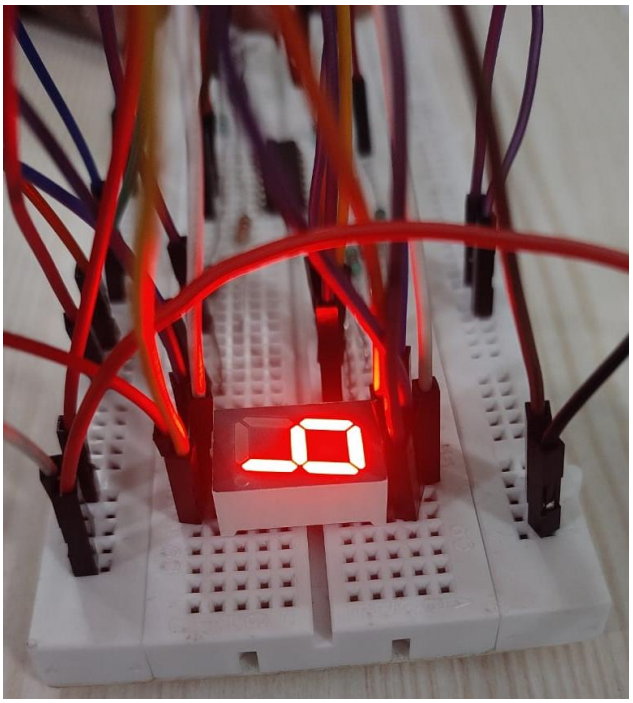












Result and Conclusion

Inputs				Expected Output
A_3 (6)	A_2 (2)	A_1 (1)	A_0 (7)	Display
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

The designed circuit successfully decodes a Binary Coded Decimal (BCD) input (0000 to 1001) into its corresponding decimal representation on a 7-segment LED display. By using a BCD to 7-segment decoder (from 0 to 9), the circuit simplifies the conversion process, ensuring correct segment illumination. The experiment verifies the accurate display of decimal digits, demonstrating the effective working of digital logic in numeric displays.

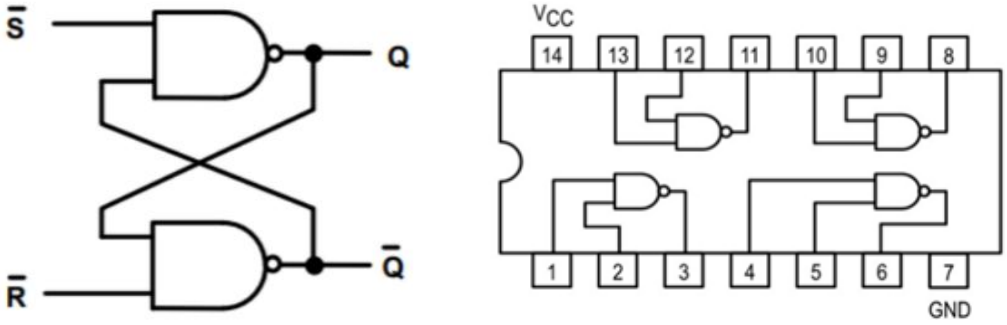
EXP VI : SR Latch

In partnership with Aditya Pratap Singh (22020)

Objectives

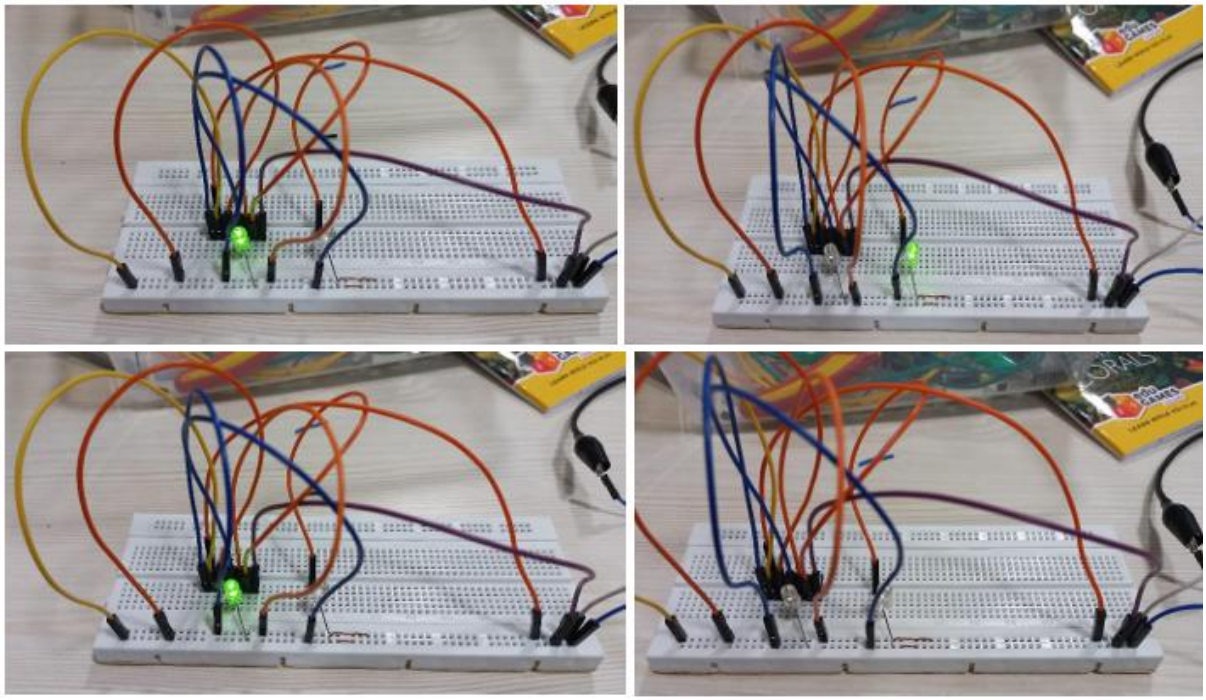
1. Design SR flip flop using NAND gates.
2. Design SR flip flop using NOR gates.
3. Verify and test type1 and type2 SR latch using SR latch IC 74LS279.

SR Flip Flop using NAND Gates

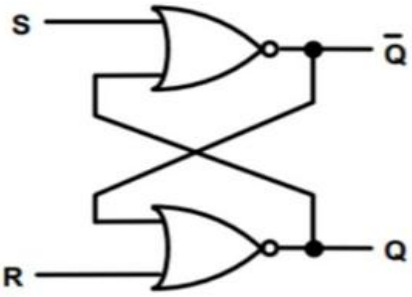


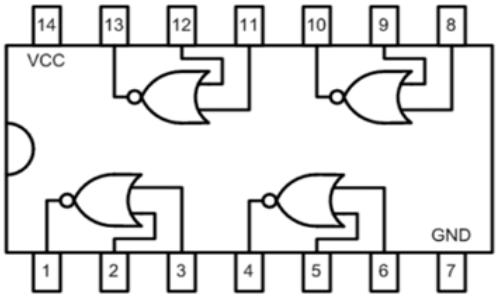
Input		Output	
1	1		
0	1	1	0
1	0	0	1
0	0	x	x

Observation



SR Flip Flop using NOR Gates

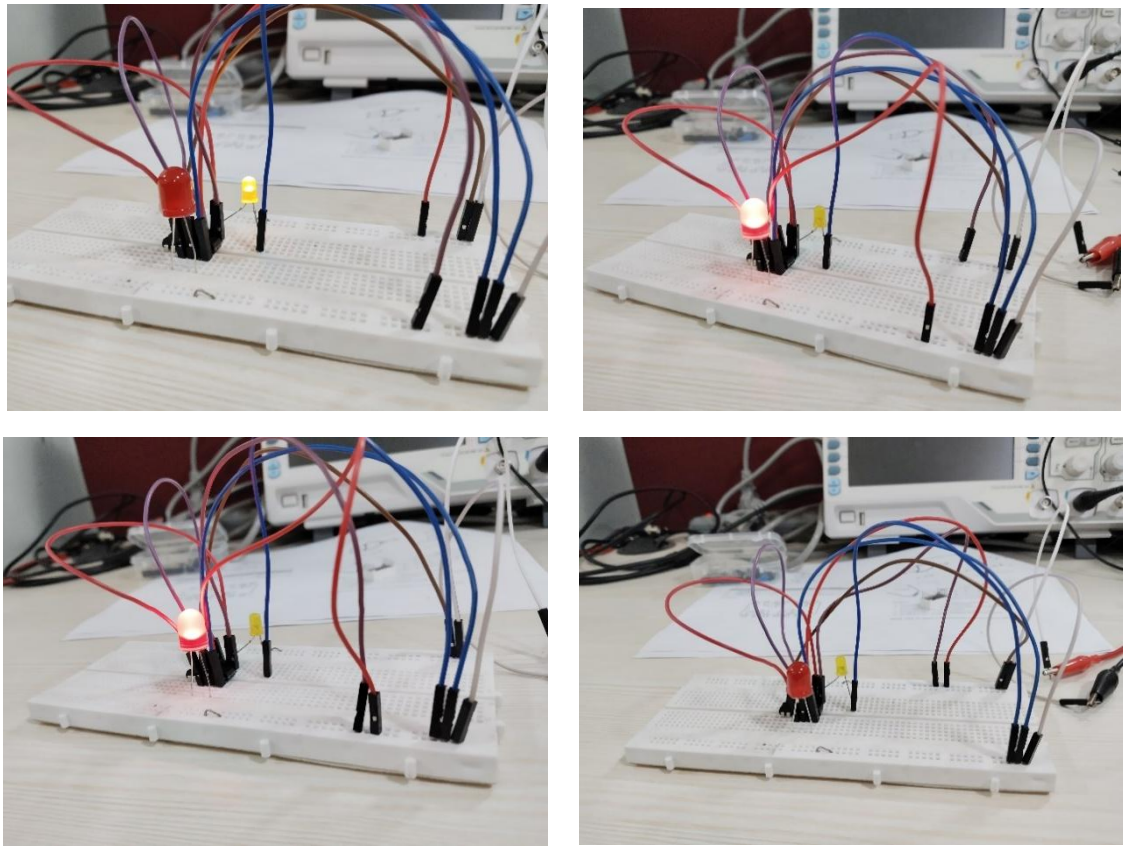




Input		Output	
0	0		
0	1	0	1
1	0	1	0
1	1	x	x

Figure 2 Truth Table for SR Latch using NOR gates

Observations



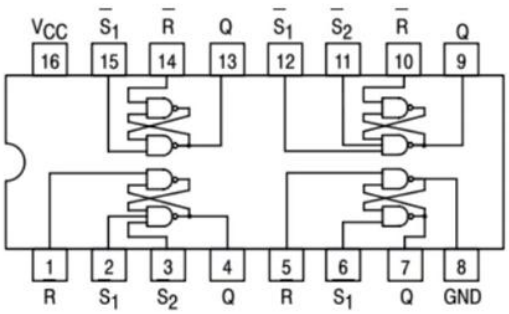
Type 1 and 2 SR Latch using IC

Input			Output
L	L	L	No Change(Q)
L	Don't care	H	H
Don't care	L	H	H
H	H	L	L
H	H	H	Pseudo stable

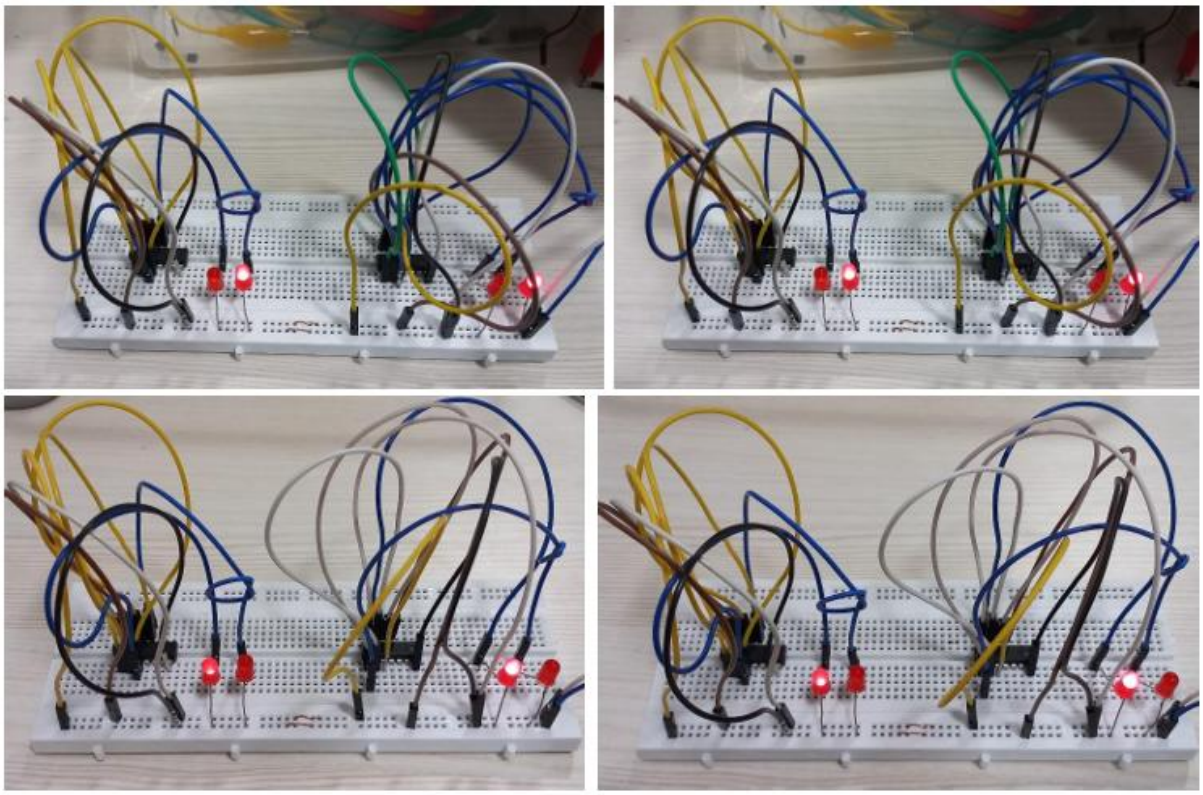
Table 3: Truth table for type1 SR latch

Input		Output
H	H	No Change(Q)
L	H	H
H	L	L
L	L	Pseudo stable

Table 4: Truth table for type2 SR latch



Observations



Results and Conclusion

1. The SR flip-flop using NAND gates works in active-low logic and behaves according to its truth table.
2. The SR flip-flop using NOR gates work in active-high logic, also matching expected behaviour.
3. The IC 74LS279 successfully implements both Type1 and Type2 SR latches and verifies the latch behaviour.

- SR flip-flops can be effectively implemented using basic logic gates (NAND or NOR).
- The logic level of inputs (active-high or active-low) changes how the flip-flop reacts.
- The IC 74LS279 simplifies SR latch implementation and confirms theoretical design via hardware.
- Care must be taken to avoid invalid states where both S and R are active simultaneously.

EXP VII : JK Flip Flop

In partnership with Aditya Pratap Singh (22020)

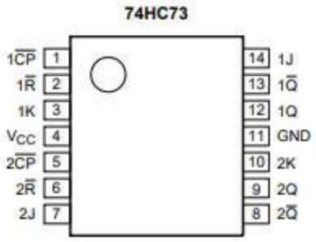
Objectives

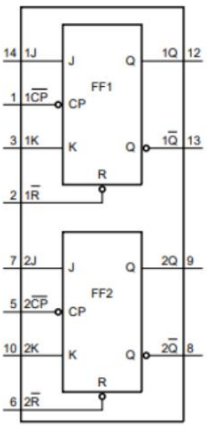
In this lab session we learnt about the working of JK flip flop and used it to create a frequency divider.

1. Familiarize yourself with the connection diagram, function table, and recommended operating conditions for the IC 74HC73N.
2. Verify the function table of the IC 74HC73N. Note down the outputs for various input settings.
3. Create frequency divider for f/4. That is, design a circuit such that, if the input clock frequency to the circuit is f, the circuit generates a square wave of frequency f/4. Record your output for input signals of different frequencies.

Materials Required

- Connecting Wires
- Breadboard
- JK Flip Flop IC (74HC73N)
- Resistor (220Ω x2)
- External LED x2
- Oscilloscope
- Function Generator
- DC supply



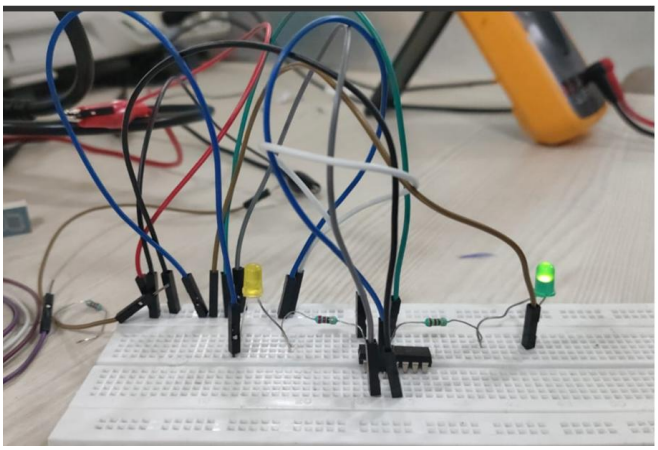


Objective 2

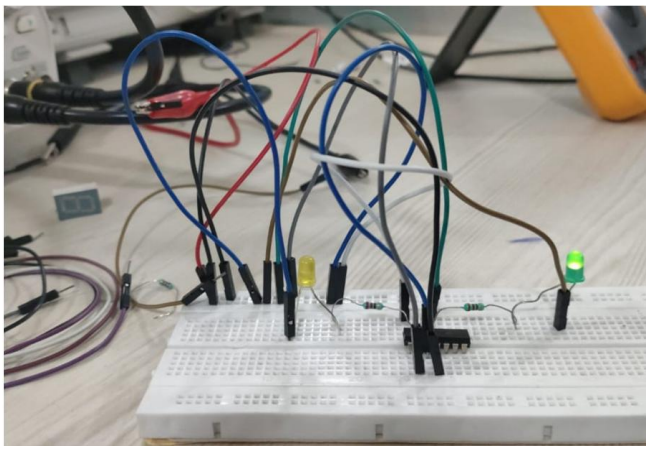
1. Connect the circuit as shown in the circuit diagram.
2. Supply V_{cc} and ground from the power supply.
3. Provide a clock pulse using a square wave from the function generator.
4. Check the output from Q and Q. Verify with the Function table.

H = HIGH voltage level; h = HIGH voltage level one set-up time prior to the HIGH-to-LOW clock transition;
L = LOW voltage level; l = LOW voltage level one set-up time prior to the HIGH-to-LOW clock transition;
q = state of referenced output one set-up time prior to the HIGH-to-LOW clock transition;
X = don't care; ↓ = HIGH-to-LOW clock transition.

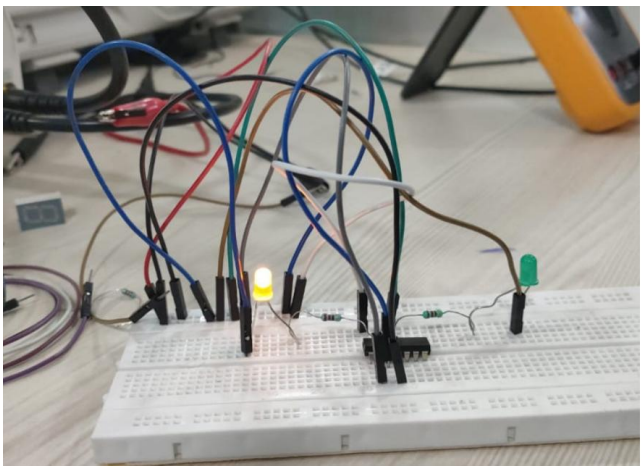
Input				Output		Operating mode
$n\overline{R}$	nCP	nJ	nK	nQ	$n\overline{Q}$	
L	X	X	X	L	H	asynchronous reset
H	↓	h	h	$\overline{q}$	q	toggle
H	↓	l	h	L	H	load 0 (reset)
H	↓	h	l	H	L	load 1 (set)
H	↓	l	l	q	$\overline{q}$	hold (no change)



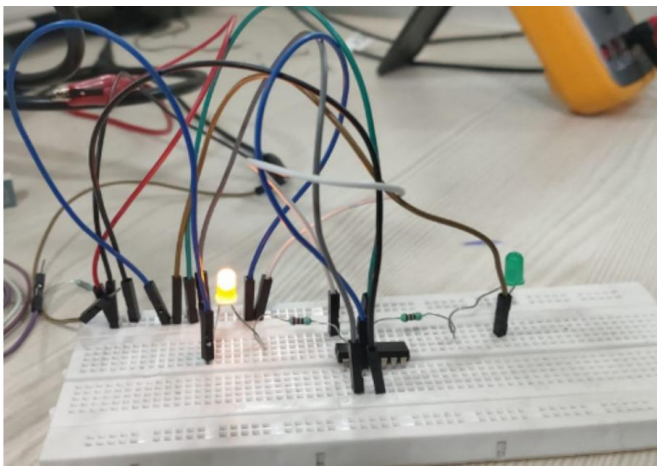
J: High ; K: Low



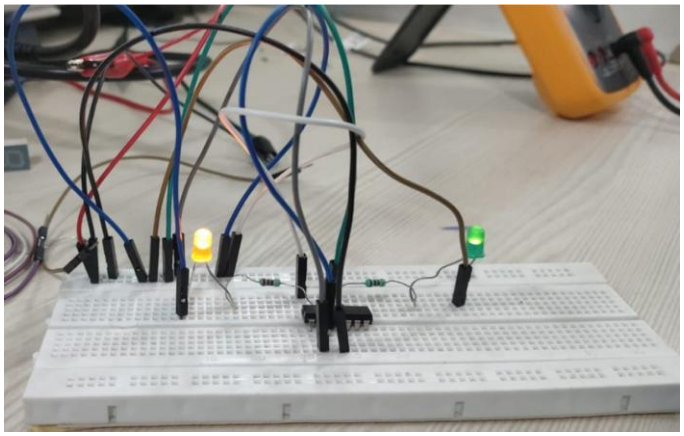
J: Low ; K: Low (Q Hold)



J: Low ; K: High



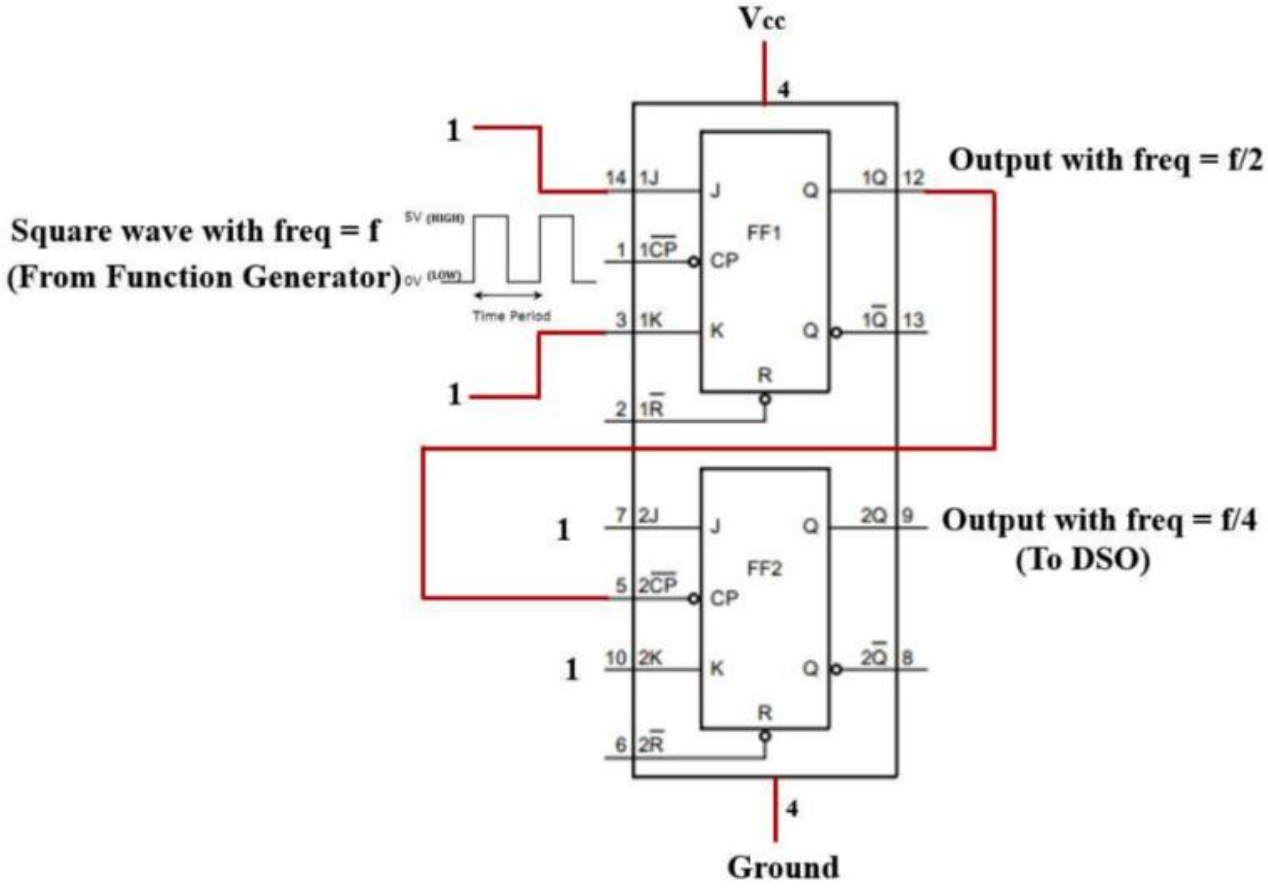
J: Low ; K: Low (Q' Hold)



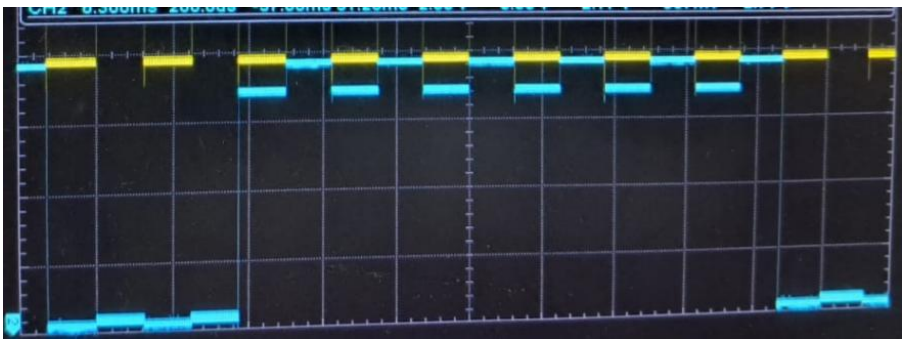
J-> High , K-> High (Toggle)

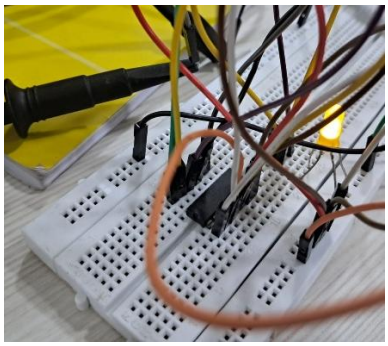
Objective 3

1. Connect the circuit as shown in the circuit diagram.
2. Supply V_{cc} and ground from the power supply.
3. J and K of both the flip flop should be high i.e. 1.
4. Connect 1**Q** and 2**CP**.Provide a clock pulse using a square wave from the function generator to 1**CP** with frequency f.
5. Check the frequency.



	Max	Min	Vpp	Top	Base	Amp	Avg	Vrms	Area	AreaP
CH1	30.0 V	-24.8 V	54.8 V	24.6 V	-19.6 V	44.2 V	2.49 V	22.3 V	298mVs	31.5mVs
CH2	4.12 V	160mV	3.96 V	4.02 V	328mV	3.69 V	1.88 V	2.55 V	225mVs	103mVs
	Freq	Peri	+Duty	-Duty	+Wid	-Wid	+Rate	-Rate	Over	Pre
CH1	79.4 Hz	12.60ms	50.00 %	50.00 %	6.300ms	6.300ms	177kV/s	-177kV/s	12.15 %	11.76 %
CH2	20.0 Hz	50.00ms	50.00 %	50.00 %	25.00ms	25.00ms	469 V/s	-14.8kV/s	2.626 %	4.539 %
	Rise	Fall	tVmax	tVmin	Vari	Vupper	Vmid	Vlower	VrmsP	
CH1	200.0us	200.0us	25.00ms	-56.40ms	493 V²	20.2 V	2.51 V	-15.2 V	22.3 V	
CH2	6.300ms	200.0us	-37.60ms	31.20ms	2.98 V²	3.65 V	2.17 V	697mV	2.71 V	





Results and Conclusion

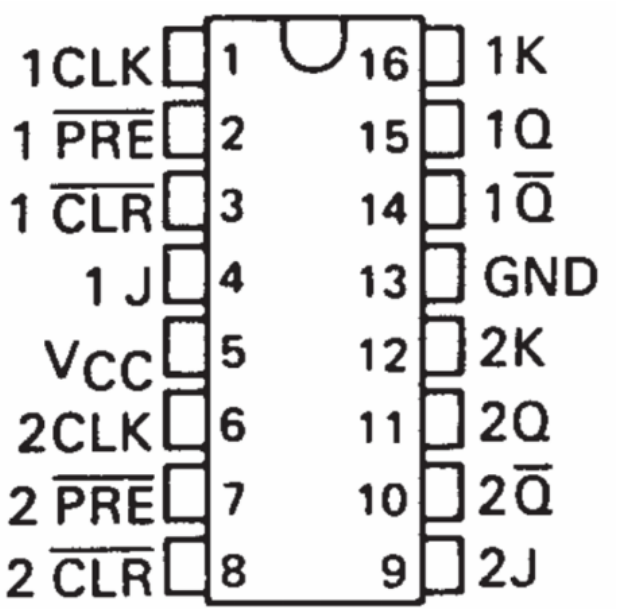
In this lab session, we explored the operation of the JK flip-flop using the 74HC73N IC, a dual JK negative edge triggered flip-flop with clear. We examined its connection diagram and verified the function table by applying various input combinations and observing the corresponding outputs, which matched the expected behaviour. Using this understanding, we constructed a frequency divider circuit that successfully divided the input clock frequency by 4. By cascading two JK flip-flops and configuring them in toggle mode (J = K = 1), we achieved a square wave output of frequency f/4 for a given input clock of frequency f. The output was tested with different input frequencies, and the observed output frequencies confirmed the correct division, demonstrating the practical application of JK flip-flops in frequency division and digital timing systems.

EXP VIII : Counters using JK Flip Flop

In partnership with Aditya Pratap Singh (22020)

Objective

Design counters using JK flip-flops



Materials Required

- Breadboard
- DC power supply
- JK Flip-Flop IC (74HC73N)
- LED
- Signal Generator
- Connecting wires

Procedure

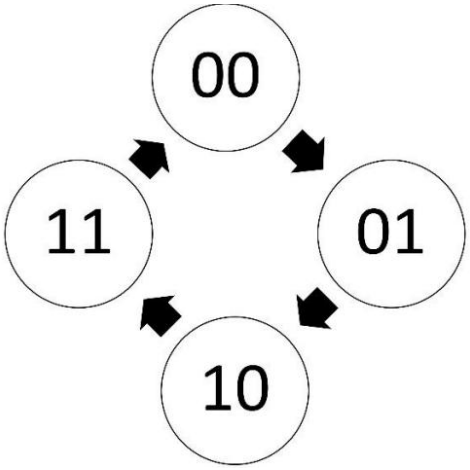
1. Connect the circuit as shown in the circuit diagram.
2. Check the LEDs before connecting to the circuit. (Connect anode to positive and cathode to negative terminals of the battery)
3. Don't increase the voltage by more than 5V.
4. For logic 1 as input, connect the respective terminal to $+V_{cc}$.
5. For logic 0 as input, connect the respective terminal to the Ground.
6. Connect the PRE pins to logic 1.

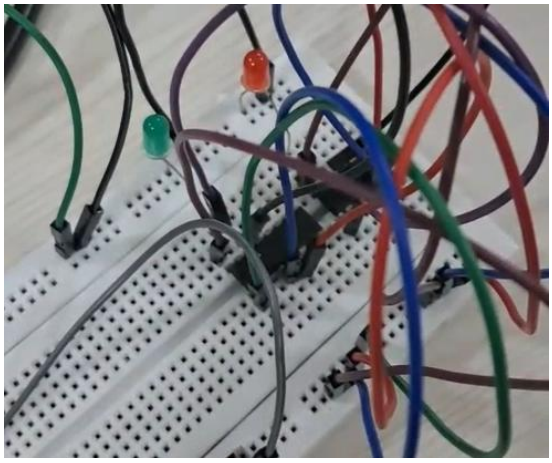
Truth table				
Clock	J	K	Q	$\bar{Q}$
1	0	0	Q	$\bar{Q}$
1	0	1	0	1
1	1	0	1	0
1	1	1	Toggle state	

Characteristic table				
Clock	J	K	Q_n	Q_{n+1}
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

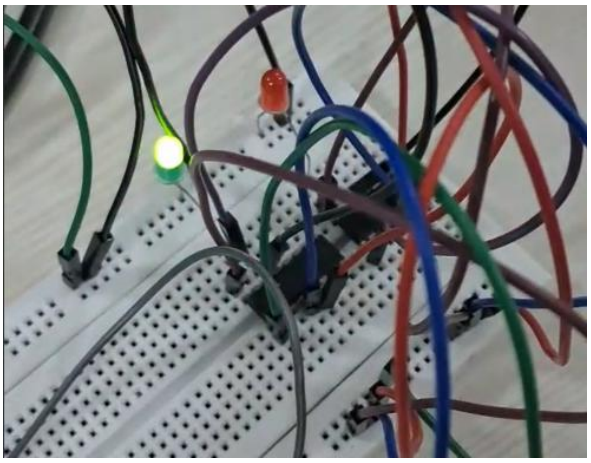
Excitation table			
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0
X=Don't care			

Experimental circuit excitation table							
Present state		Next state		Output			
Q_1	Q_0	Q_1	Q_0	J_1	K_1	J_0	K_0
0	0	0	1	0	X	1	X
0	1	1	0	1	X	X	1
1	0	1	1	X	0	1	X
1	1	0	0	X	1	X	1

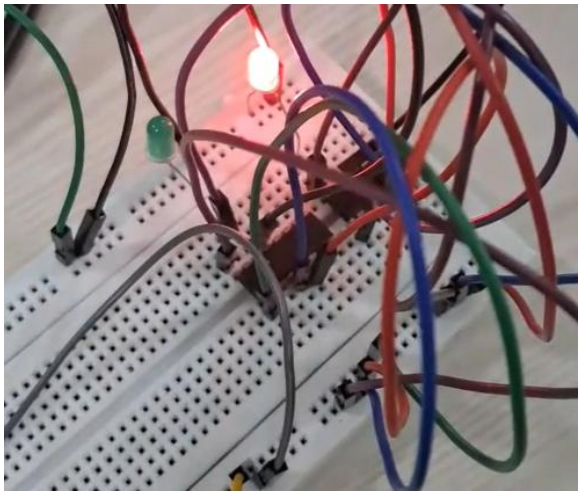




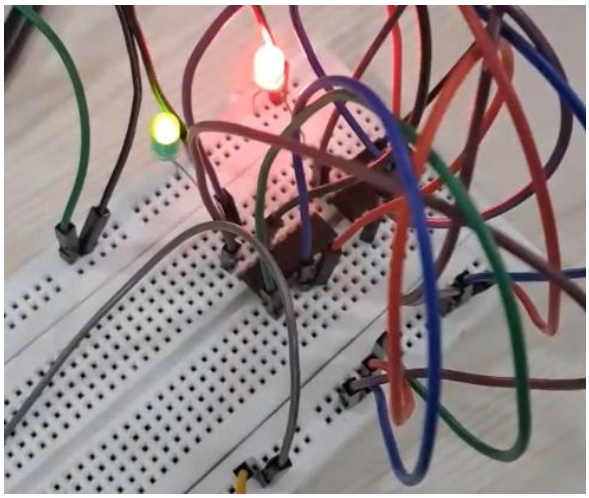
00



01



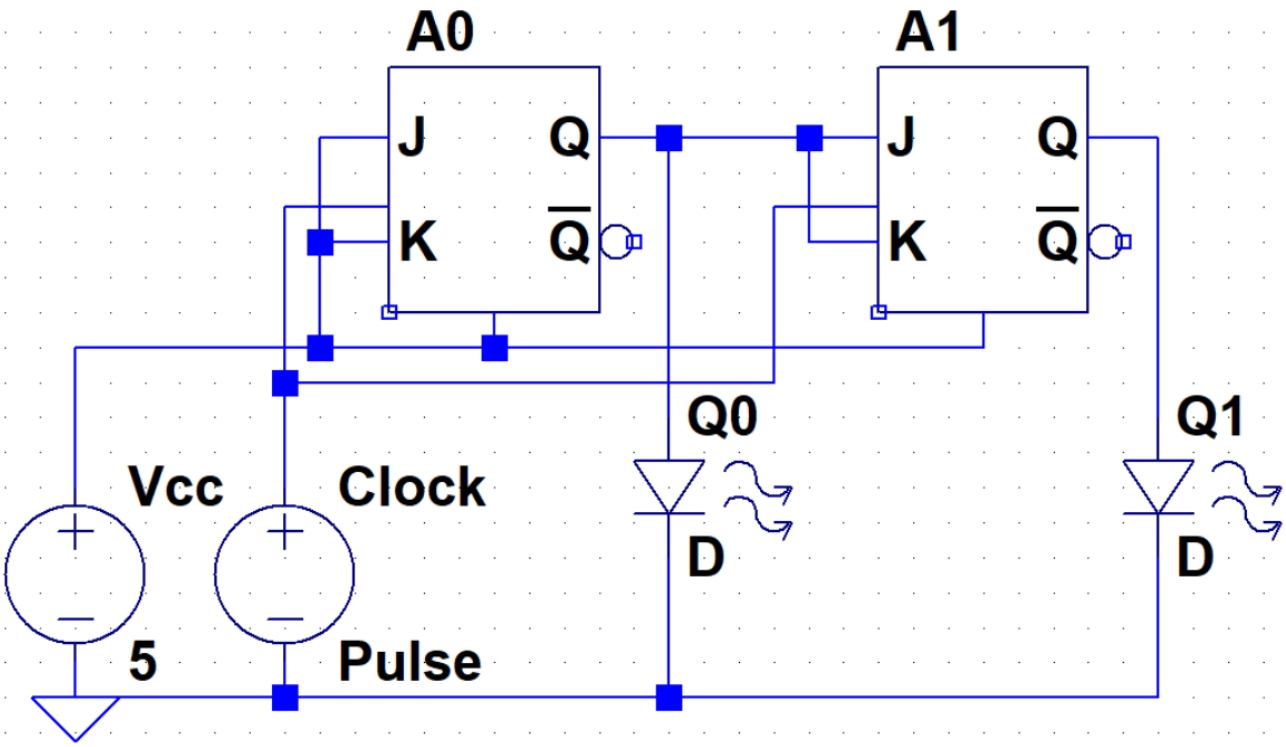
10



11

Result and Conclusion

In this experiment, we successfully designed and implemented a 2-bit binary counter using JK flip-flops. By configuring two JK flip-flops in a cascaded manner with both J and K inputs tied high (logic 1), we enabled toggle functionality on the negative edge of the clock. The first flip-flop toggled with every clock pulse, while the second flip-flop toggled with the output of the first, effectively counting in binary from 00 to 11 (0 to 3 in decimal) and then repeating. The outputs were observed and verified for each clock cycle, and the counter behaved as expected, accurately cycling through the 2-bit binary sequence. This demonstrated the utility of JK flip-flops in sequential logic design, particularly in implementing simple synchronous counters.



EXP IX : Simulation of MOSFET and CMOS

In partnership with Aditya Pratap Singh (22020)

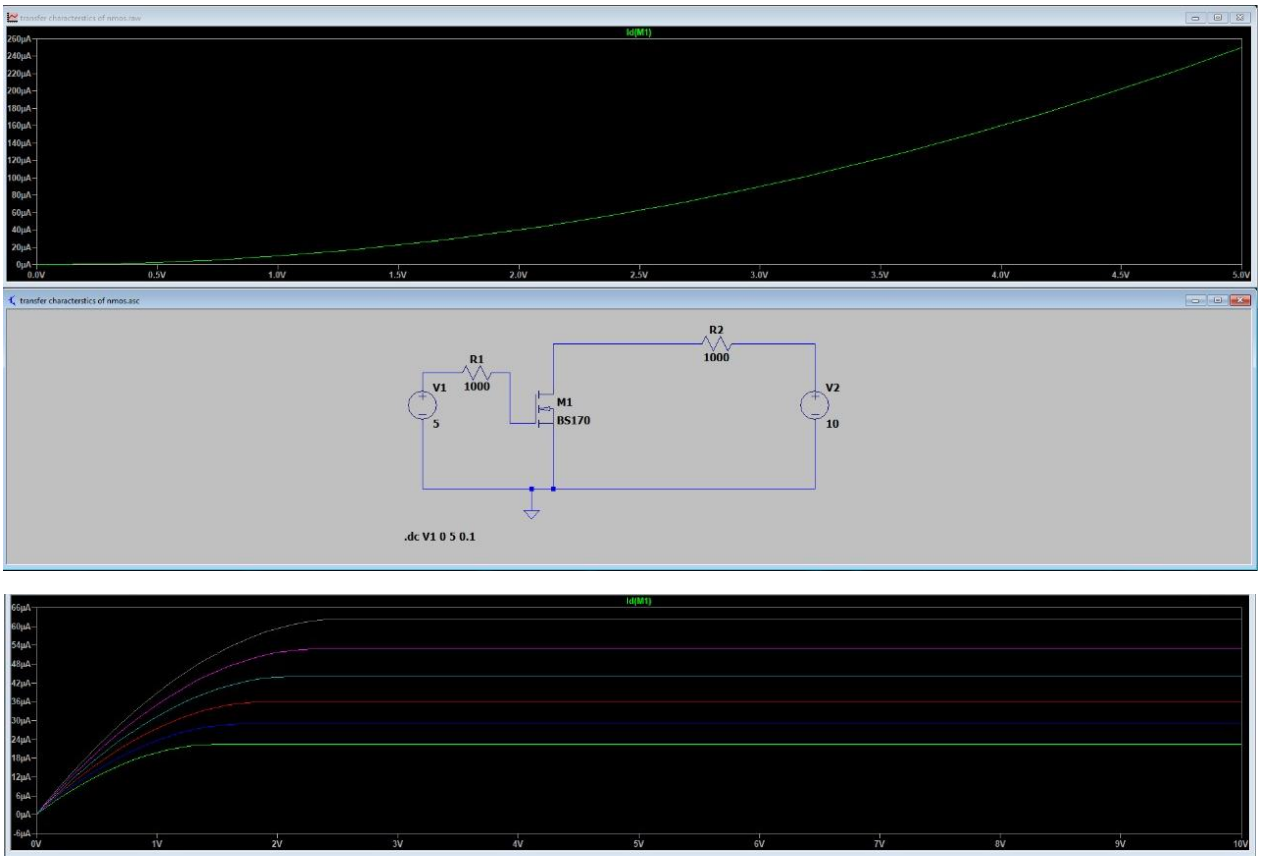
Objective

This experiment aims to understand the LTspice simulation with experimental plots:

1. the transfer characteristics and output characteristics of n-channel enhancement-type MOSFETs
2. the transfer characteristics and output characteristics of p-channel enhancement-type MOSFETs, and
3. the voltage transfer characteristics (VTC) of a CMOS inverter by connecting the n-MOSFET and p-MOSFET.

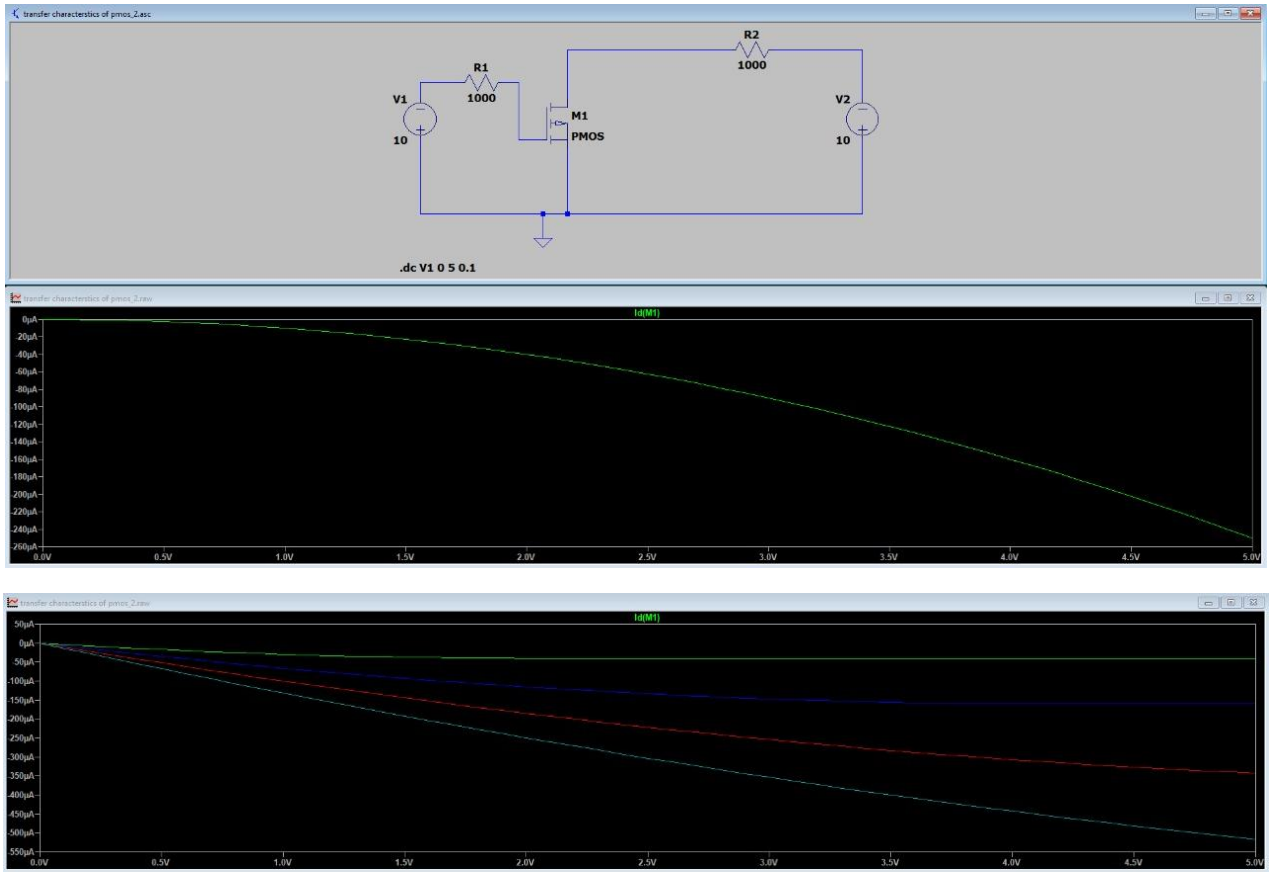
Requirement is only a PC with LTSpice installed.

Objetive 1 (NMOS)



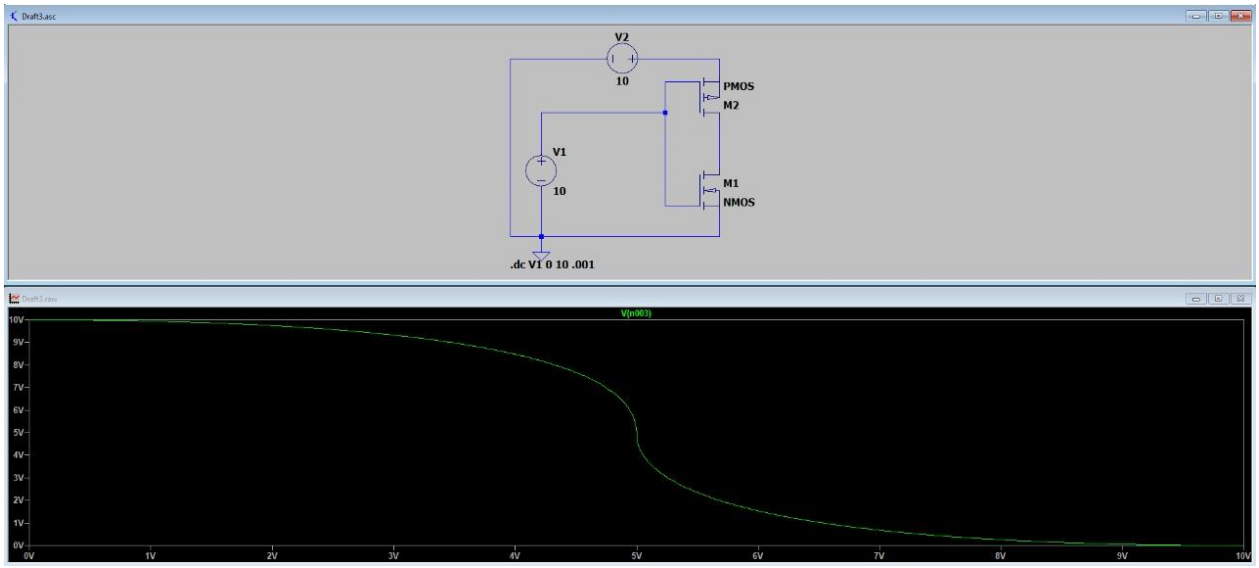
Above is the transfer and below are the output characters of the NMOS.

Objective 2 (PMOS)



Above is the transfer and below are the output characters of the PMOS.

Objective 3 (CMOS)



Voltage Transfer characteristics of CMOS.

Results and Conclusion

In this experiment, we used LTspice to simulate and analyse the electrical behaviour of enhancement-type n-channel and p-channel MOSFETs, as well as a CMOS inverter. The transfer characteristics (Id vs. V_{gs}) for both NMOS and PMOS were plotted, revealing the threshold voltages and showcasing the gradual increase in drain current with increasing gate voltage beyond the threshold. The output characteristics (I_d vs. V_{ds}) were also obtained for different gate voltages, clearly illustrating the distinct linear and saturation regions of operation. Finally, we constructed a CMOS inverter using both NMOS and PMOS transistors and plotted its voltage transfer characteristics (V_{out} vs. V_{in}). The VTC curve confirmed the inverter action with a sharp transition near the midpoint of the supply voltage, demonstrating its switching behaviour. These simulations validated theoretical expectations and provided deeper insights into MOSFET operation and CMOS logic design.