AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Engineering

Lab Report

Experiment # 06

Experiment Title: Communication between two Arduino Boards using SPI.

Date of Perform:	28 April 2025	**Date of Submission:**	5 May 2025
Course Title:	Microprocessor and Embedded Systems Lab		
Course Code:	EE4103	**Section:**	P
Semester:	Spring 2024-25	**Degree Program:**	BSc in CSE
Course Teacher:	**Prof. Dr. Engr. Muhibul Haque Bhuyan**		

Group # 01

Sl No	Name	ID	PROGRAM	SIGNATURE
1.	Md.Saikat Hossain	23-51242-1	BSc in CSE	
2.	Md.Mosharof Hossain Khan	23-51259-1	BSc in CSE	
3.	Rimal Banik	23-51260-1	BSc in CSE	
4.	Md.Rahidul Islam	23-51269-1	BSc in CSE	
5.	Rahat Ahmed	21-44911-2	BSc in CSE	

Faculty use only

FACULTY COMMENTS	**Marks Obtained**	
	Total Marks	

Contents

Objectives:

The objectives of this experiment are to-

a) Study the SPI protocol used in Arduino.
b) Write assembly language programming code for SPI communication with Arduinos.
c) Use SPI protocol for communication between two Arduinos.
d) Build a circuit to control the master side LED by the push button at the slave side and vice versa using the SPI Serial communication protocol.
e) Know the working principles of the SPI used in Arduino.

Apparatus:

1. Arduino IDE 2.3.5
2. Arduino Microcontroller board (2)
3. LED lights (2)
4. Push Button (2)
5. Resistors 10 kΩ, 2.2 kΩ(2 + 2)
6. Breadboard and
7. Jumper wires

Circuit Diagram:

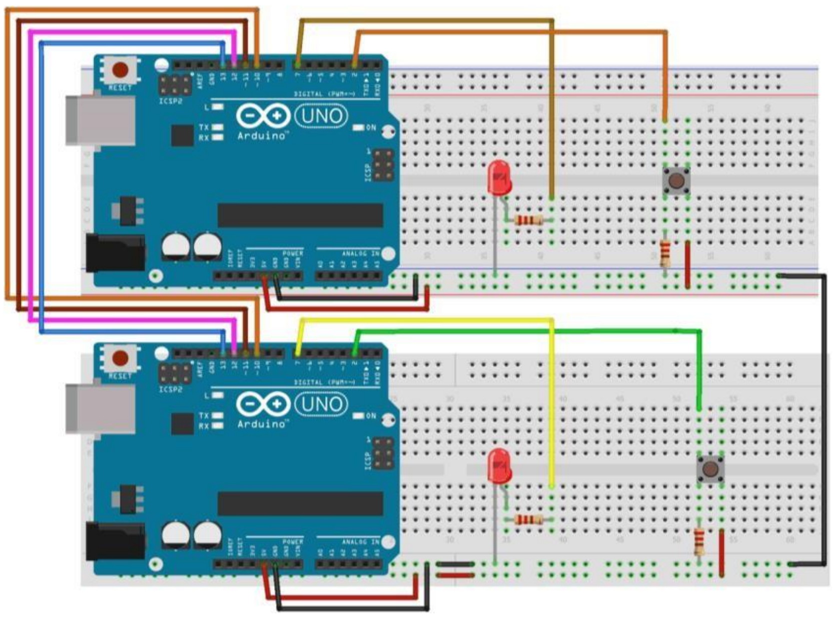


Figure-01: Two Arduino board's pin connections for SPI communications

Code Explanation:

PART 1: Master Code:

```
//SPI MASTER (ARDUINO)
//SPI COMMUNICATION BETWEEN TWO ARDUINO CIRCUIT DIGEST

#include<SPI.h>            //Library for SPI
 #define LED 7
#define ipbutton 2
 int buttonvalue;
 int x;

void setup (void){
  Serial.begin(115200); //Starts Serial Communication at Baud Rate 115200
  pinMode(ipbutton,INPUT);       //Sets pin 2 as input
  pinMode(LED,OUTPUT);           //Sets pin 7 as Output
  SPI.begin();                   //Begins the SPI communication
  SPI.setClockDivider(SPI_CLOCK_DIV8); //Sets clock for SPI communication at  8 (16/8 = 2 MHz)
  digitalWrite(SS,HIGH); //Setting SS to HIGH do disconnect master from slave
}

void loop(void){
  byte Mastersend, Mastereceive;
  buttonvalue = digitalRead(ipbutton);   //Reads the status of the pin 2

  if(buttonvalue == HIGH) //Setting x for the slave based on input at pin 2
  {
    x = 1;
  }
  else
  {
    x = 0;
  }

  digitalWrite(SS, LOW); //Starts communication with Slave from the Master
  Mastersend = x;
  Mastereceive = SPI.transfer(Mastersend); //Sends the Mastersend value to
//the slave and also receives value from the slave
if(Mastereceive == 1) //To set the LED based on the value received from slave
  {
    digitalWrite(LED,HIGH); //Sets pin 7 HIGH
    Serial.println("Master LED is ON");
  }
 else
  {
   digitalWrite(LED,LOW);  //Sets pin 7 LOW
   Serial.println("Master LED is OFF");
  }
  delay(1000);
}
```

Explanation: This code demonstrates SPI (Serial Peripheral Interface) communication where the Arduino board operates as an SPI **Master**. It reads input from a **push button** connected to pin 2 and controls an **LED** on pin 7 based on feedback received from an SPI **Slave** device. The **SPI.h** library is used to manage the SPI communication protocol.In the setup() function, **serial communication** is initialized at a baud rate of **115200** for debugging and monitoring. Pin 2 is configured as an **input** to detect button presses, while pin 7 is set as an **output** to control the LED. The SPI communication is initiated using SPI.begin(), and the SPI clock speed is set to **2 MHz** using SPI.setClockDivider(SPI_CLOCK_DIV8). Initially, the **Slave Select (SS)** pin is set to **HIGH**, effectively disabling the slave until communication is needed. Within the loop() function, the code continuously monitors the state of the button. If the button connected to pin 2 is pressed, the variable x is set to 1; otherwise, it is set to 0. The master then starts SPI communication by setting SS to **LOW**, sends the value of x to the slave using SPI.transfer(x), and simultaneously receives a byte from the slave. The received value (Mastereceive) determines whether to turn the LED ON or OFF. If the master receives a 1, it turns the LED ON and prints "Master LED is ON" via the Serial Monitor. Otherwise, the LED remains OFF, and it prints "Master LED is OFF."

PART 2: Slave Code:

```
//SPI SLAVE (ARDUINO)
//SPI COMMUNICATION BETWEEN TWO ARDUINO

#include<SPI.h>
 #define LEDpin 7
#define buttonpin 2

volatile boolean received;
volatile byte Slavereceived, Slavesend;
int buttonvalue;
int x;

void setup(){
  Serial.begin(115200);
  pinMode(buttonpin,INPUT);    // Setting pin 2 as INPUT
  pinMode(LEDpin,OUTPUT);      // Setting pin 7 as OUTPUT
  pinMode(MISO,OUTPUT);     //Sets MISO as OUTPUT to send data to

Master In
  SPCR |= _BV(SPE);          //Turn on SPI in Slave Mode
  received = false;
  SPI.attachInterrupt();   //Interrupt ON is set for SPI communication
}
 ISR(SPI_STC_vect)          //Interrupt routine function
{
  Slavereceived = SPDR; // Value received from Master stored in
Slavereceived
  received = true;      //Sets received as True
}
```

```
void loop() {
if(received) //To set LED ON/OFF based on the value received from Master
   {
  if (Slavereceived == 1)
      {
        digitalWrite(LEDpin, HIGH);   //Sets pin 7 as HIGH to turn on LED
       Serial.println("Slave LED is ON");
       }
 else
      {
        digitalWrite(LEDpin,LOW);    //Sets pin 7 as LOW to turn off
LED
        Serial.println("Slave LED is OFF");
      }

   buttonvalue = digitalRead(buttonpin); //Reads the status of the pin 2
   if (buttonvalue == HIGH)  //To set the value of x to send to Master
      {
        x = 1;
      }
 else
      {
        x=0;
      }
Slavesend = x;
  SPDR = Slavesend;    //Sends the x value to the Master via SPDR
  delay(1000);
    }
}
```

Explanation: This code demonstrates SPI (Serial Peripheral Interface) communication where the Arduino functions as an SPI Slave device. It receives input from the SPI Master, controls an LED on pin 7, and sends the state of its own push button (connected to pin 2) back to the master. The **SPI.h** library is included to handle all SPI-related functions efficiently.In the **setup()** function, serial communication is initialized at a baud rate of **115200** for debugging and monitoring via the Serial Monitor. The slave's push button pin (pin 2) is configured as input, and the LED pin (pin 7) is configured as output. The **MISO pin** is also set as output because, in SPI, the slave uses it to send data to the master. The line **SPCR|=_BV(SPE)** enables SPI in Slave Mode, and **SPI.attachInterrupt()** enables an interrupt that is triggered when data is received from the master. When the master sends data, the SPI interrupt service routine **ISR(SPI_STC_vect)** is executed. It stores the received byte into the variable **Slavereceived** and sets a received flag to true.In the **loop()** function, if data has been received **(received == true),** the slave checks the value of Slavereceived. If it's 1, it turns the LED ON and prints "Slave LED is ON" to the Serial Monitor. If it's 0, the LED is turned OFF and it prints "Slave LED is OFF". Simultaneously, the slave reads its own button (on pin 2), and based on its state, sets a value (x) which is then loaded into the SPI Data Register (SPDR). This value will be sent back to the master during the next SPI transfer, enabling two-way communication.

Hardware Implementation and Results:

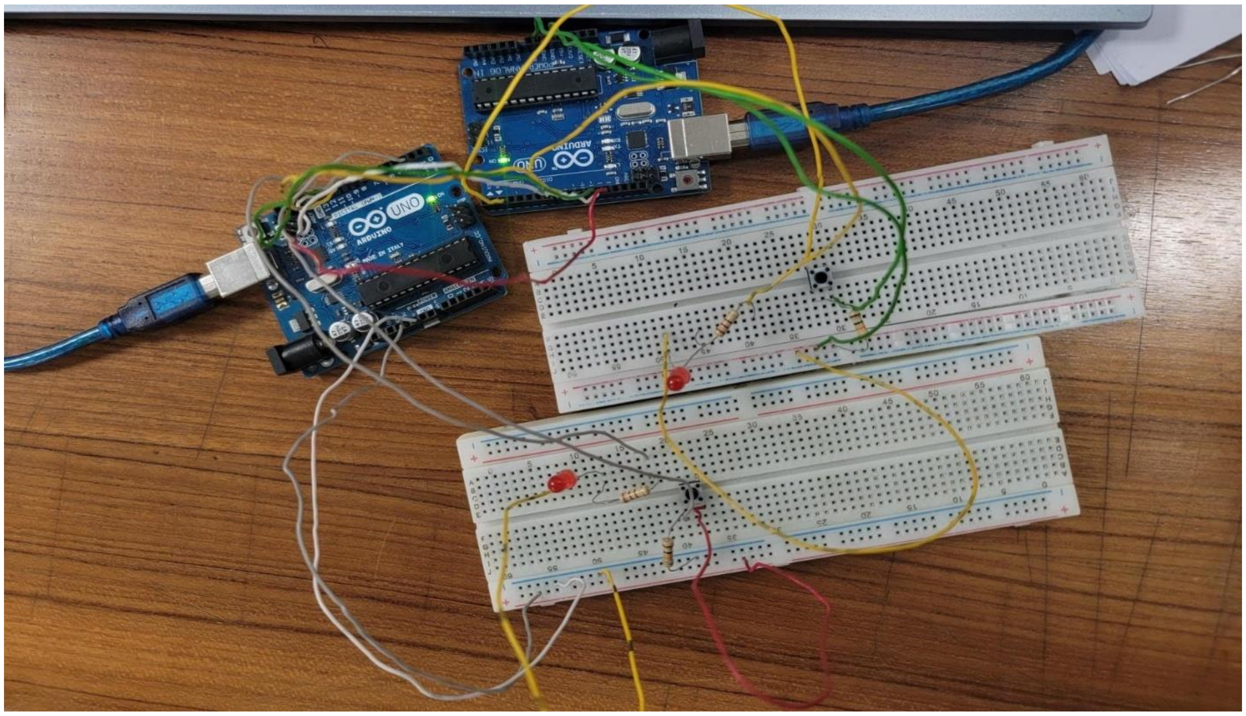
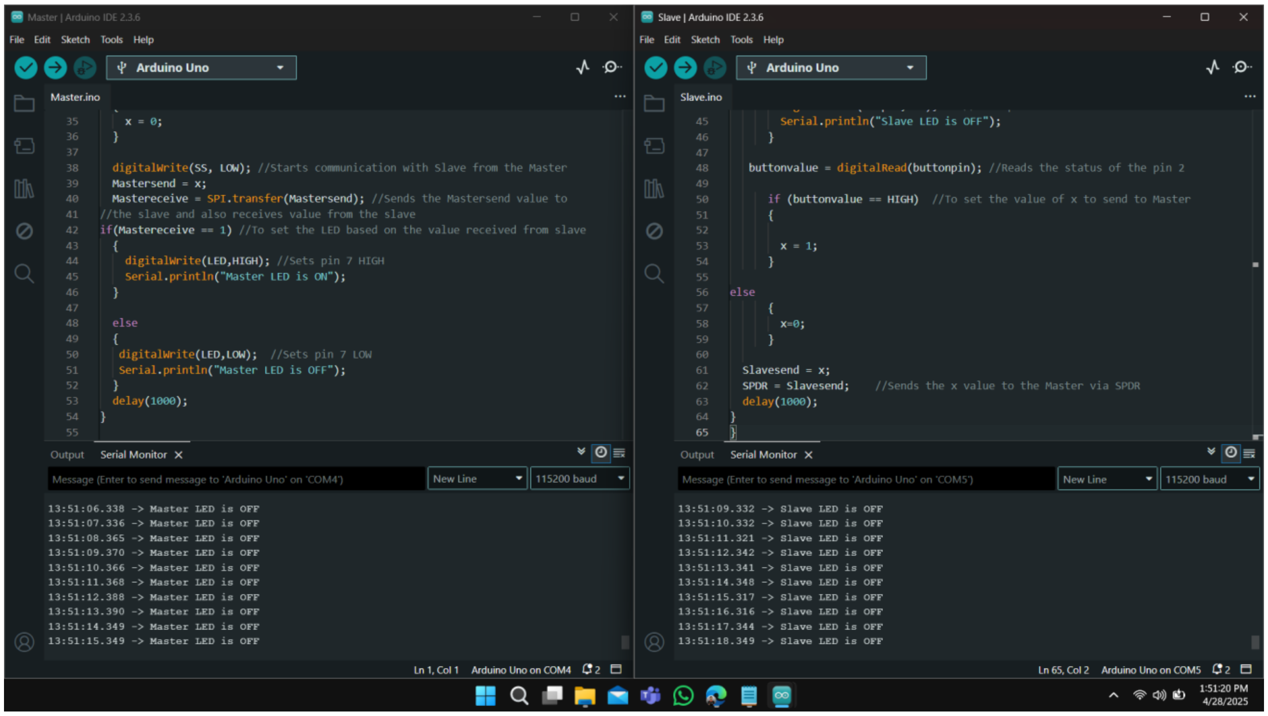


Figure-02: Both Master and Slave LEDs remain OFF when switches are not pressed



```
    x = 0;
  }

  digitalWrite(SS, LOW); //Starts communication with Slave from the Master
  Mastersend = x;
  Mastereceive = SPI.transfer(Mastersend); //Sends the Mastersend value to
//the slave and also receives value from the slave
if(Mastereceive == 1) //To set the LED based on the value received from slave
  {
    digitalWrite(LED,HIGH); //Sets pin 7 HIGH
    Serial.println("Master LED is ON");
  }

  else
  {
   digitalWrite(LED,LOW);  //Sets pin 7 LOW
   Serial.println("Master LED is OFF");
  }
  delay(1000);
}
```

Message (Enter to send message to 'Arduino Uno' on 'COM4')

```
13:51:06.338 -> Master LED is OFF
13:51:07.336 -> Master LED is OFF
13:51:08.365 -> Master LED is OFF
13:51:09.370 -> Master LED is OFF
13:51:10.366 -> Master LED is OFF
13:51:11.368 -> Master LED is OFF
13:51:12.388 -> Master LED is OFF
13:51:13.390 -> Master LED is OFF
13:51:14.349 -> Master LED is OFF
13:51:15.349 -> Master LED is OFF
```

```
        Serial.println("Slave LED is OFF");
      }

   buttonvalue = digitalRead(buttonpin); //Reads the status of the pin 2

      if (buttonvalue == HIGH)  //To set the value of x to send to Master
      {

        x = 1;
      }

else
      {
        x=0;
      }

  Slavesend = x;
  SPDR = Slavesend;    //Sends the x value to the Master via SPDR
  delay(1000);
}
}
```

Message (Enter to send message to 'Arduino Uno' on 'COM5')

```
13:51:09.332 -> Slave LED is OFF
13:51:10.332 -> Slave LED is OFF
13:51:11.321 -> Slave LED is OFF
13:51:12.342 -> Slave LED is OFF
13:51:13.341 -> Slave LED is OFF
13:51:14.348 -> Slave LED is OFF
13:51:15.317 -> Slave LED is OFF
13:51:16.316 -> Slave LED is OFF
13:51:17.344 -> Slave LED is OFF
13:51:18.349 -> Slave LED is OFF
```

Figure-03: Both Master and Slave LEDs remain OFF when switches are not pressed

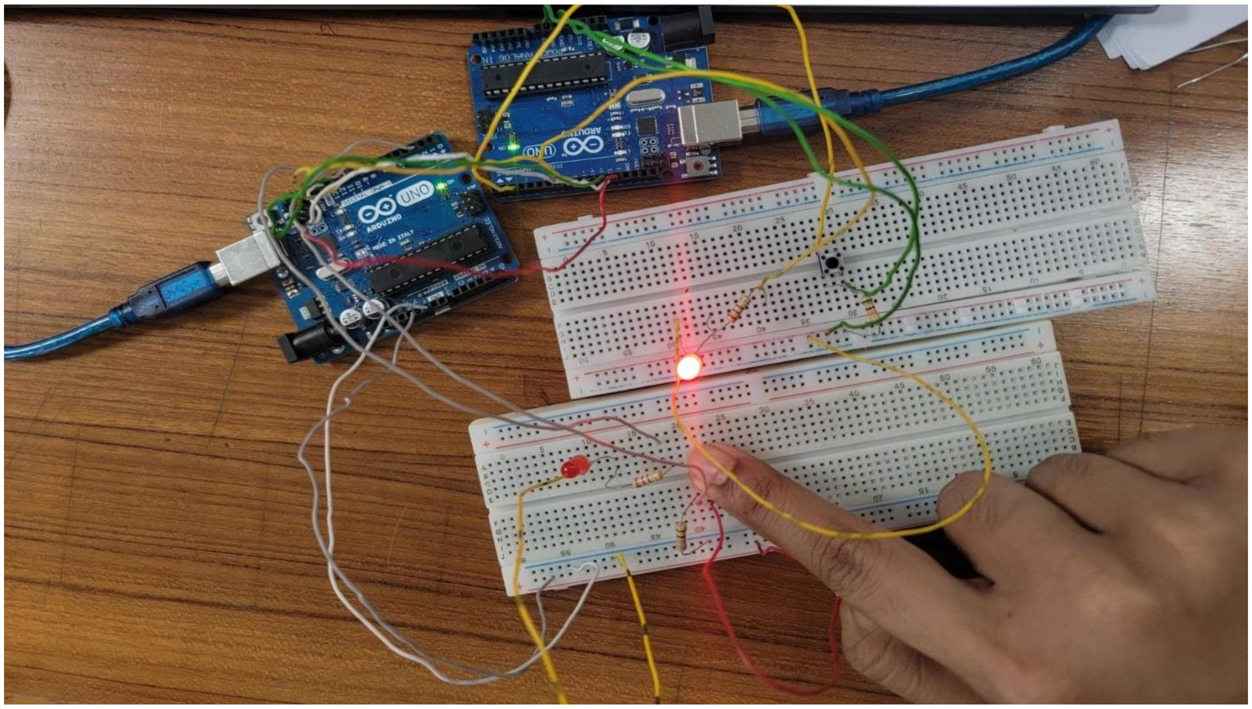
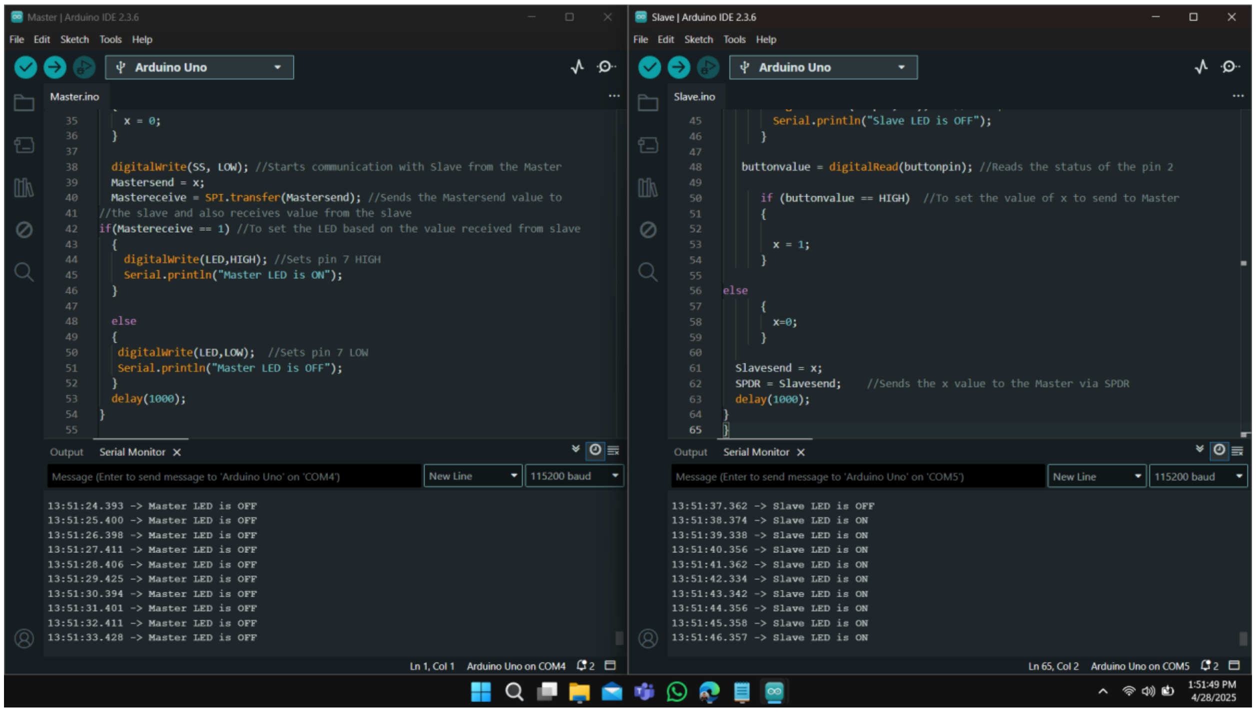


Figure-04: Slave Arduino LED turns ON when Master Arduino switch is pressed



```
    x = 0;
   }

  digitalWrite(SS, LOW); //Starts communication with Slave from the Master
  Mastersend = x;
  Mastereceive = SPI.transfer(Mastersend); //Sends the Mastersend value to
//the slave and also receives value from the slave
if(Mastereceive == 1) //To set the LED based on the value received from slave
  {
    digitalWrite(LED,HIGH); //Sets pin 7 HIGH
    Serial.println("Master LED is ON");
  }

  else
  {
   digitalWrite(LED,LOW);  //Sets pin 7 LOW
   Serial.println("Master LED is OFF");
  }
  delay(1000);
}
```

```
13:51:24.393 -> Master LED is OFF
13:51:25.400 -> Master LED is OFF
13:51:26.398 -> Master LED is OFF
13:51:27.411 -> Master LED is OFF
13:51:28.406 -> Master LED is OFF
13:51:29.425 -> Master LED is OFF
13:51:30.394 -> Master LED is OFF
13:51:31.401 -> Master LED is OFF
13:51:32.411 -> Master LED is OFF
13:51:33.428 -> Master LED is OFF
```

Ln 1, Col 1   Arduino Uno on COM4
Slave | Arduino IDE 2.3.6
File Edit Sketch Tools Help
Arduino Uno
Slave.ino

```
      Serial.println("Slave LED is OFF");
    }

   buttonvalue = digitalRead(buttonpin); //Reads the status of the pin 2

    if (buttonvalue == HIGH)  //To set the value of x to send to Master
    {

      x = 1;
    }

else
    {
      x=0;
    }

  Slavesend = x;
  SPDR = Slavesend;    //Sends the x value to the Master via SPDR
  delay(1000);
}
}
```

```
13:51:37.362 -> Slave LED is OFF
13:51:38.374 -> Slave LED is ON
13:51:39.338 -> Slave LED is ON
13:51:40.356 -> Slave LED is ON
13:51:41.362 -> Slave LED is ON
13:51:42.334 -> Slave LED is ON
13:51:43.342 -> Slave LED is ON
13:51:44.356 -> Slave LED is ON
13:51:45.358 -> Slave LED is ON
13:51:46.357 -> Slave LED is ON
```

Figure-05: Slave Arduino LED turns ON when Master Arduino switch is pressed

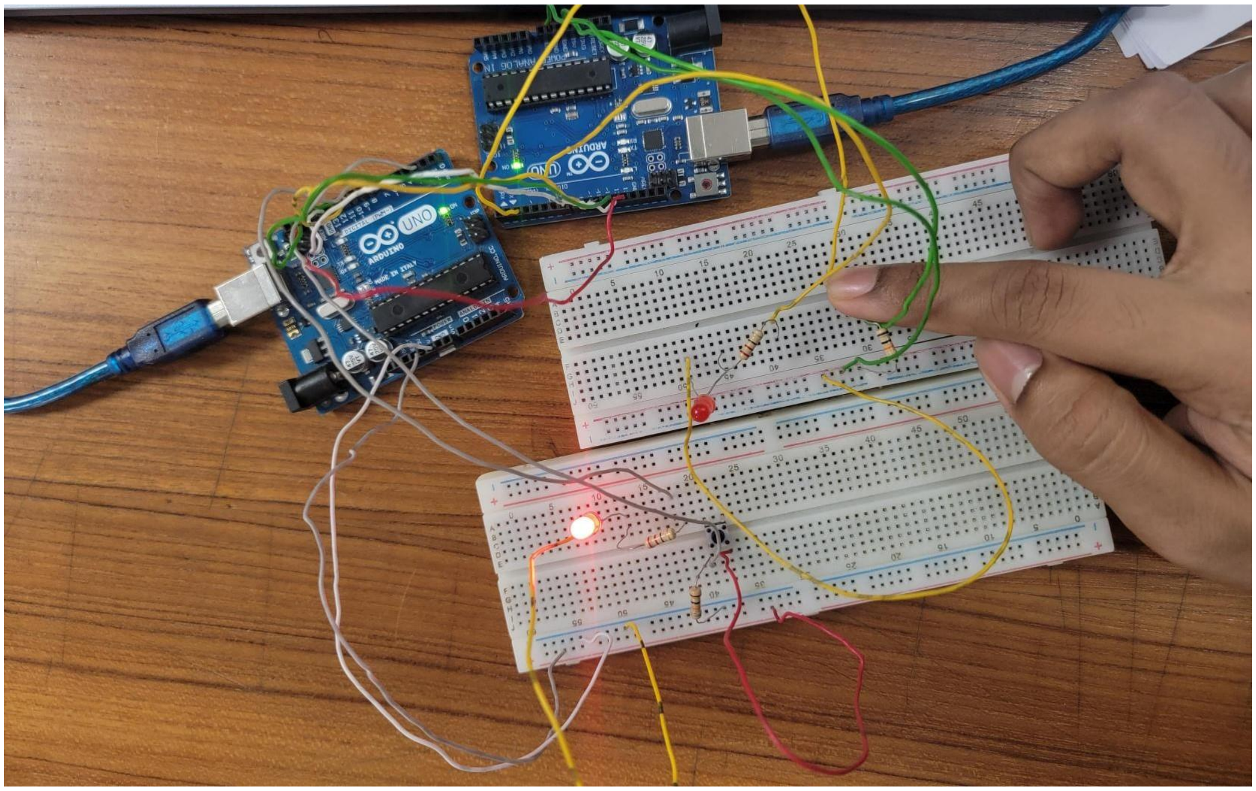
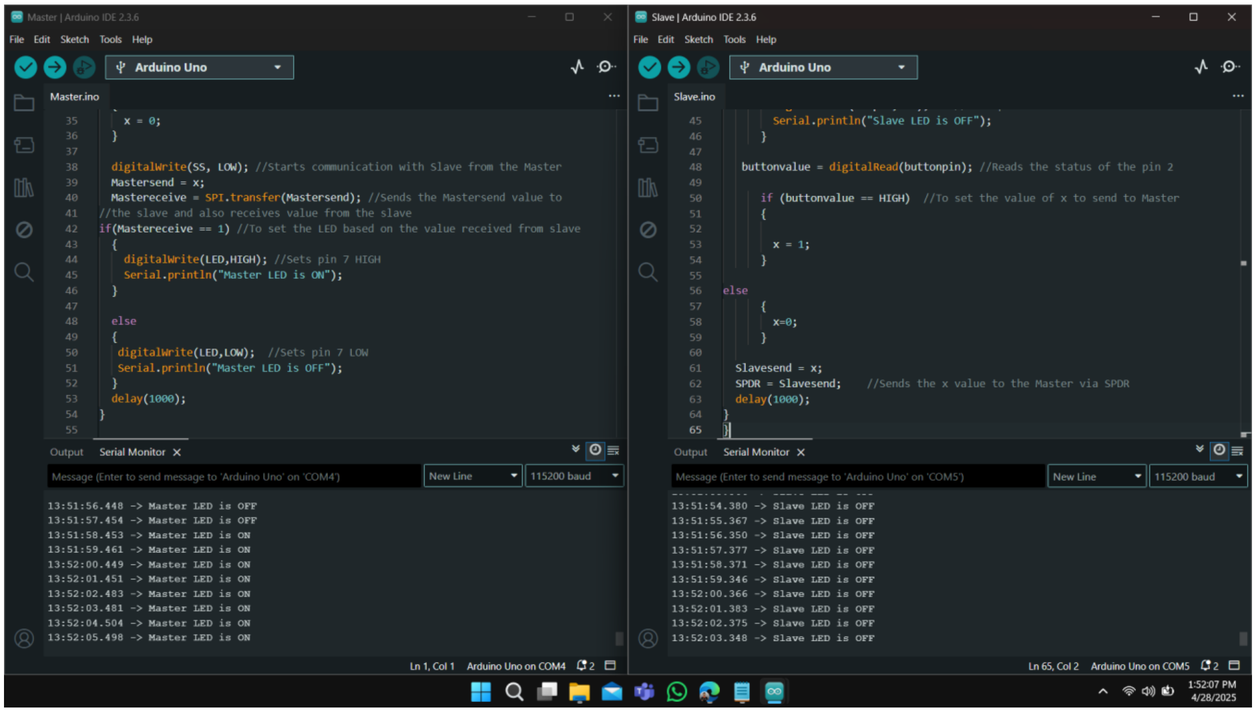


Figure-06: Master Arduino LED turns ON when Slave Arduino switch is pressed


Master | Arduino IDE 2.3.6
File Edit Sketch Tools Help
Arduino Uno
Master.ino
Output Serial Monitor
Message (Enter to send message to 'Arduino Uno' on 'COM4')
New Line
115200 baud
Ln 1, Col 1 Arduino Uno on COM4 2
Slave | Arduino IDE 2.3.6
File Edit Sketch Tools Help
Arduino Uno
Slave.ino
Output Serial Monitor
Message (Enter to send message to 'Arduino Uno' on 'COM5')
New Line
115200 baud
Ln 65, Col 2 Arduino Uno on COM5 2
1:52:07 PM
4/28/2025

```
    x = 0;
  }

  digitalWrite(SS, LOW); //Starts communication with Slave from the Master
  Mastersend = x;
  Mastereceive = SPI.transfer(Mastersend); //Sends the Mastersend value to
//the slave and also receives value from the slave
if(Mastereceive == 1) //To set the LED based on the value received from slave
  {
    digitalWrite(LED,HIGH); //Sets pin 7 HIGH
    Serial.println("Master LED is ON");
  }

  else
  {
   digitalWrite(LED,LOW);  //Sets pin 7 LOW
   Serial.println("Master LED is OFF");
  }
  delay(1000);
}
```

```
13:51:56.448 -> Master LED is OFF
13:51:57.454 -> Master LED is OFF
13:51:58.453 -> Master LED is ON
13:51:59.461 -> Master LED is ON
13:52:00.449 -> Master LED is ON
13:52:01.451 -> Master LED is ON
13:52:02.483 -> Master LED is ON
13:52:03.481 -> Master LED is ON
13:52:04.504 -> Master LED is ON
13:52:05.498 -> Master LED is ON
```

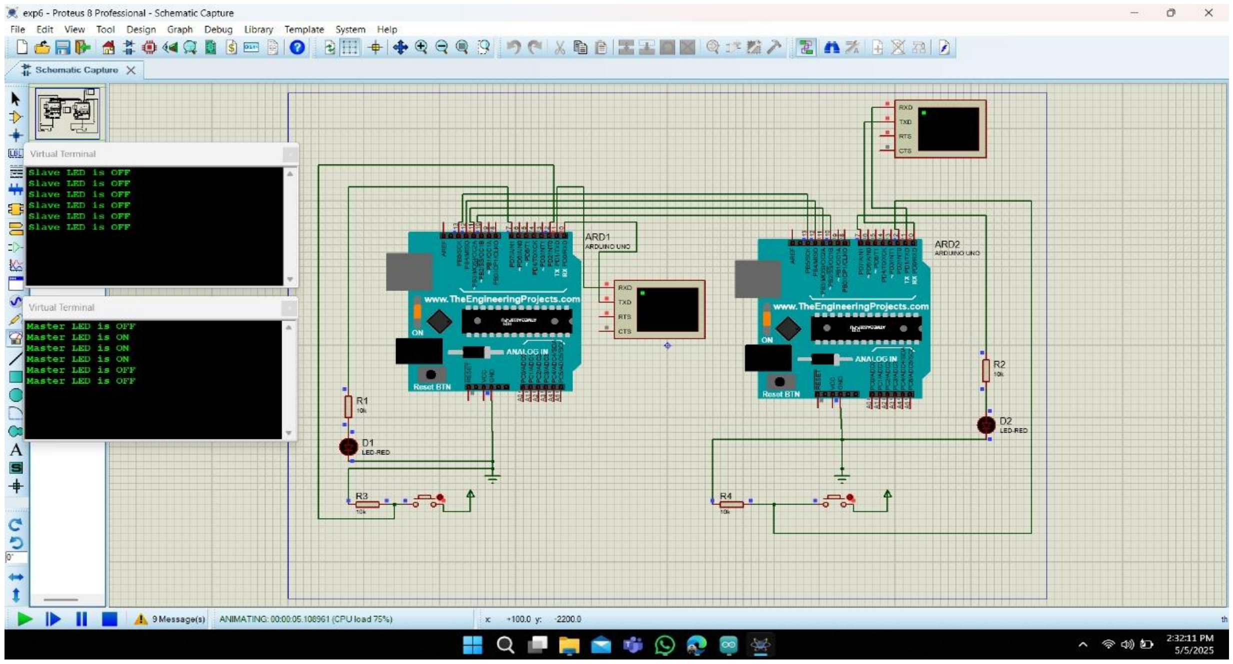
```
      Serial.println("Slave LED is OFF");
    }

  buttonvalue = digitalRead(buttonpin); //Reads the status of the pin 2

    if (buttonvalue == HIGH)  //To set the value of x to send to Master
    {

      x = 1;
    }

else
    {
      x=0;
    }

  Slavesend = x;
  SPDR = Slavesend;    //Sends the x value to the Master via SPDR
  delay(1000);
}
}
```

```
13:51:54.380 -> Slave LED is OFF
13:51:55.367 -> Slave LED is OFF
13:51:56.350 -> Slave LED is OFF
13:51:57.377 -> Slave LED is OFF
13:51:58.371 -> Slave LED is OFF
13:51:59.346 -> Slave LED is OFF
13:52:00.366 -> Slave LED is OFF
13:52:01.383 -> Slave LED is OFF
13:52:02.375 -> Slave LED is OFF
13:52:03.348 -> Slave LED is OFF
```

Figure-07: Master Arduino LED turns ON when Slave Arduino switch is pressed

Explanation: This demonstrates SPI (Serial Peripheral Interface) communication between two Arduino Uno boards—one acting as the Master, and the other as the Slave. The hardware setup includes two push buttons (one for each Arduino) and two LEDs connected to digital pin 7 of each board to visually represent the output based on the received SPI signals. Each button is connected to digital pin 2 and uses a pull-down resistor to ensure stable input. The SPI connections are made as follows:MOSI (Master Out Slave In) – Master pin 11 → Slave pin 11,MISO (Master In Slave Out) – Master pin 12 → Slave pin 12,SCK (Clock) – Master pin 13 → Slave pin 13,SS (Slave Select) – Master pin 10 → Slave pin 10.
The screenshot images displays the Arduino IDE's Serial Monitors for both the Master (left) and the Slave (right) devices. Each Serial Monitor shows the real-time feedback of SPI communication:On the 1st Screenshot Master side, messages such as "Master LED is OFF" are printed, which indicates that the Master is not receiving a value of 1 from the Slave, so it keeps the LED turned off.On the Slave side, similar "Slave LED is OFF" messages are seen, indicating that the Slave is also receiving 0 from the Master, hence its LED remains off as well. On the 2nd Screenshot Master side, messages such as "Master LED is OFF" are printed, which indicates that the Master is not receiving a value of 1 from the Slave, so it keeps the LED turned off.On the Slave side "Slave LED is ON" messages are seen, indicating that the Slave is receiving 1 from the Master. On the 3rd Screenshot Master side, messages such as "Master LED is ON" are printed, which indicates that the Master is receiving a value of 1 from the Slave, so it keeps the LED turned on. On the Slave side "Slave LED is OFF" messages are seen, indicating that the Slave is receiving 0 from the Master. These results confirm that SPI communication is functioning correctly in both directions. The Master can send data to the Slave and receive data from it, and vice versa.

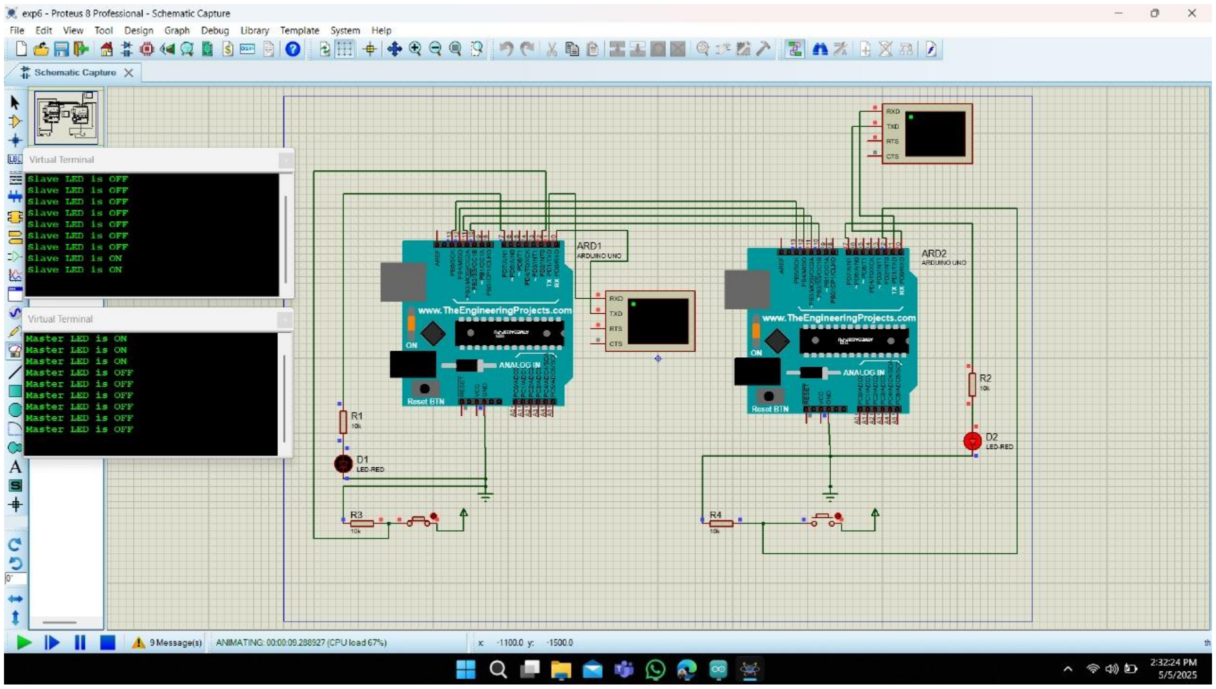
Simulation Output Results:


exp6 - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Virtual Terminal
Virtual Terminal
ARD1
ARDUINO UNO
ARD2
ARDUINO UNO
R1
R2
R3
R4
D1
LED-RED
D2
LED-RED
RXD
TXD
RTS
CTS
9 Message(s)
ANIMATING: 00:00:05.108961 (CPU load 75%)

```
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
```

```
Master LED is OFF
Master LED is ON
Master LED is ON
Master LED is ON
Master LED is OFF
Master LED is OFF
```

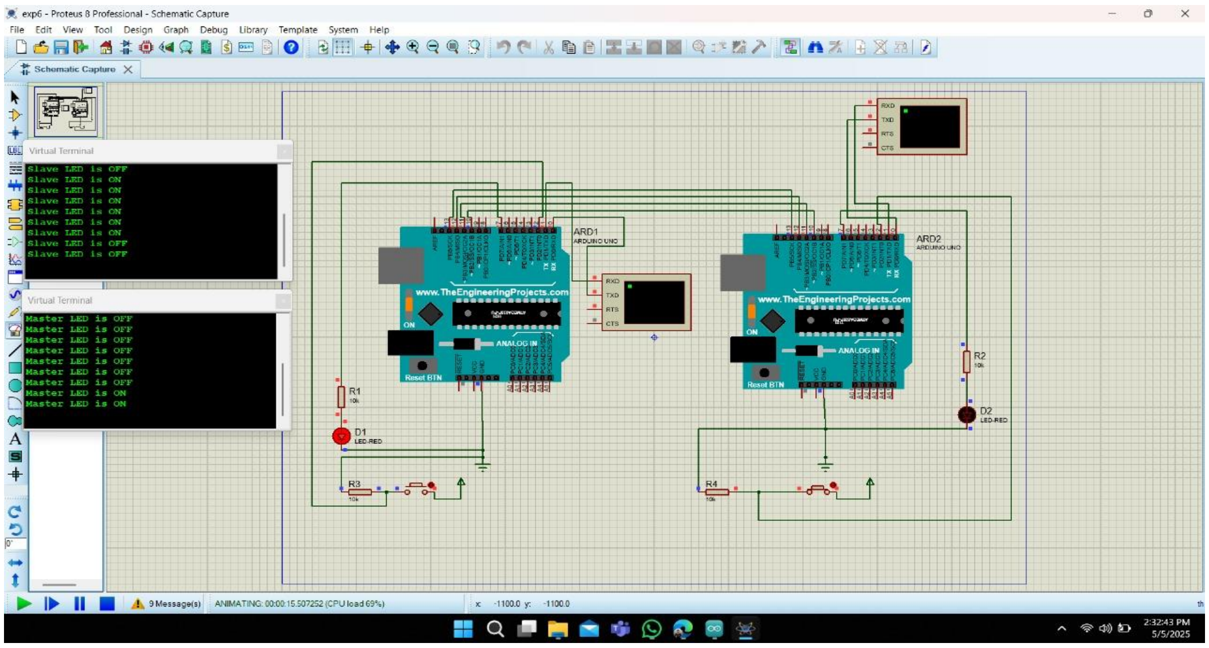
Figure-08: Both Master and Slave LEDs remain OFF when switches are not pressed(Simulation)



```
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is ON
Slave LED is ON
```

```
Master LED is ON
Master LED is ON
Master LED is ON
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
```

Figure-09: Slave Arduino LED turns ON when Master Arduino switch is pressed(Simulation)



```
Slave LED is OFF
Slave LED is ON
Slave LED is ON
Slave LED is ON
Slave LED is ON
Slave LED is ON
Slave LED is ON
Slave LED is OFF
Slave LED is OFF
```

```
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is ON
Master LED is ON
```

Figure-10: Master Arduino LED turns ON when Slave Arduino switch is pressed(Simulation)

Explanation: In this simulation, the Arduino Uno board, 220Ω and 10kΩ resistors, and LEDs and switches were set up according to the hardware configuration used in the lab. The program was written and verified in the Arduino IDE 2.3.5, which generated a HEX file. This file was then loaded into the Proteus simulation to simulate the behavior of the circuit. After running the simulation, the results were observed and compared with the actual hardware results to check for consistency and verify the performance of the system.

Answer to Question:

3. Configure the port numbers for outputs and inputs according to your ID. Consider the last two digits from your ID (if your ID is XY-PQABC-Z then consider input port as B and output port as C of your ID). Include all the programs and results within your lab report.

Ans: **ID:23-51242-1,**so the port B is 4 and port C is 2.

Master code:

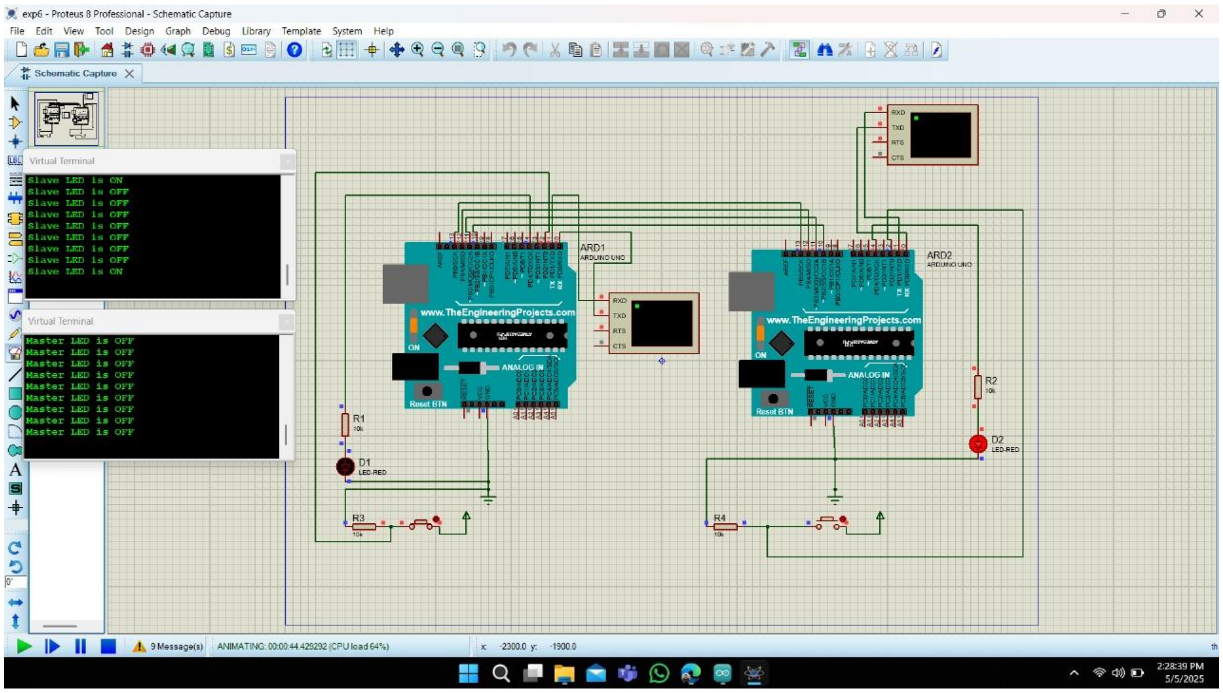
```
//SPI MASTER (ARDUINO)
//SPI COMMUNICATION BETWEEN TWO ARDUINO CIRCUIT DIGEST

#include<SPI.h>          //Library for SPI
 #define LED 4
#define ipbutton 2
 int buttonvalue;
 int x;

void setup (void){
  Serial.begin(115200); //Starts Serial Communication at Baud Rate 115200
  pinMode(ipbutton,INPUT);       //Sets pin 2 as input
  pinMode(LED,OUTPUT);          //Sets pin 4 as Output
  SPI.begin();                //Begins the SPI communication
  SPI.setClockDivider(SPI_CLOCK_DIV8); //Sets clock for SPI communication at  8 (16/8 = 2 MHz)
  digitalWrite(SS,HIGH); //Setting SS to HIGH do disconnect master from slave
}

void loop(void){
  byte Mastersend, Mastereceive;
  buttonvalue = digitalRead(ipbutton);   //Reads the status of the pin 2

  if(buttonvalue == HIGH) //Setting x for the slave based on input at pin 2
  {
    x = 1;
  }
  else
  {
    x = 0;
  }

digitalWrite(SS, LOW); //Starts communication with Slave from the Master
  Mastersend = x;
  Mastereceive = SPI.transfer(Mastersend); //Sends the Mastersend value to
//the slave and also receives value from the slave
if(Mastereceive == 1) //To set the LED based on the value received from slave
  {
    digitalWrite(LED,HIGH); //Sets pin 4 HIGH
    Serial.println("Master LED is ON");
  }
else
  {
   digitalWrite(LED,LOW);  //Sets pin 4 LOW
   Serial.println("Master LED is OFF");
  }
  delay(1000);
}
```

Slave code:

```
//SPI SLAVE (ARDUINO)
//SPI COMMUNICATION BETWEEN TWO ARDUINO

#include<SPI.h>
 #define LEDpin 4
#define buttonpin 2

volatile boolean received;
volatile byte Slavereceived, Slavesend;
int buttonvalue;
int x;

void setup(){
  Serial.begin(115200);
  pinMode(buttonpin,INPUT);    // Setting pin 2 as INPUT
  pinMode(LEDpin,OUTPUT);      // Setting pin 4 as OUTPUT
  pinMode(MISO,OUTPUT);     //Sets MISO as OUTPUT to send data to

Master In
  SPCR |= _BV(SPE);           //Turn on SPI in Slave Mode
  received = false;
  SPI.attachInterrupt();   //Interrupt ON is set for SPI communication
}
 ISR(SPI_STC_vect)          //Interrupt routine function
{
  Slavereceived = SPDR; // Value received from Master stored in
Slavereceived
  received = true;      //Sets received as True
}
void loop() {
if(received) //To set LED ON/OFF based on the value received from Master
   {
  if (Slavereceived == 1)
      {
        digitalWrite(LEDpin, HIGH);   //Sets pin 4 as HIGH to turn on LED
       Serial.println("Slave LED is ON");
       }
else
      {
        digitalWrite(LEDpin,LOW);    //Sets pin 4 as LOW to turn off
LED
        Serial.println("Slave LED is OFF");
      }

   buttonvalue = digitalRead(buttonpin); //Reads the status of the pin 2
   if (buttonvalue == HIGH)  //To set the value of x to send to Master
      {
        x = 1;
      }
 else
      {
        x=0;
      }
Slavesend = x;
  SPDR = Slavesend;    //Sends the x value to the Master via SPDR
  delay(1000);
    }
}
```

Explanation: This project demonstrates two-way SPI communication between two Arduino boards, where one functions as the Master and the other as the Slave. Each Arduino is connected to a button and an LED. The objective is to control the LED on each board based on the button status of the opposite board using SPI (Serial Peripheral Interface) communication. In the Master code, the button is connected to pin 2 and the LED to pin 4. SPI communication is initialized using the SPI.begin() function, and the clock speed is set to 2 MHz. Within the main loop, the Master continuously reads the button status. If the button is pressed, it sends a value of 1 to the Slave; otherwise, it sends 0. During the transfer, it also receives a value from the Slave. Based on the received value, the Master controls its LED—turning it ON if the received value is 1, or OFF if it is 0.On the Slave side, SPI is initialized with interrupts enabled, allowing it to react immediately when data is received from the Master. The Slave reads the value sent by the Master using the SPI interrupt service routine (ISR), and stores it in a variable. It then uses this value to control its own LED. Simultaneously, the Slave reads its own button status on pin 2 and prepares a response value to be sent back to the Master during the next SPI transaction. This enables both Arduinos to update their LED status based on the remote board's button input.
Overall, this code showcases the basic implementation of SPI for real-time communication between two Arduinos. It allows each board to both send and receive data, making it a useful approach for projects involving synchronized control or sensor data sharing between multiple microcontrollers.

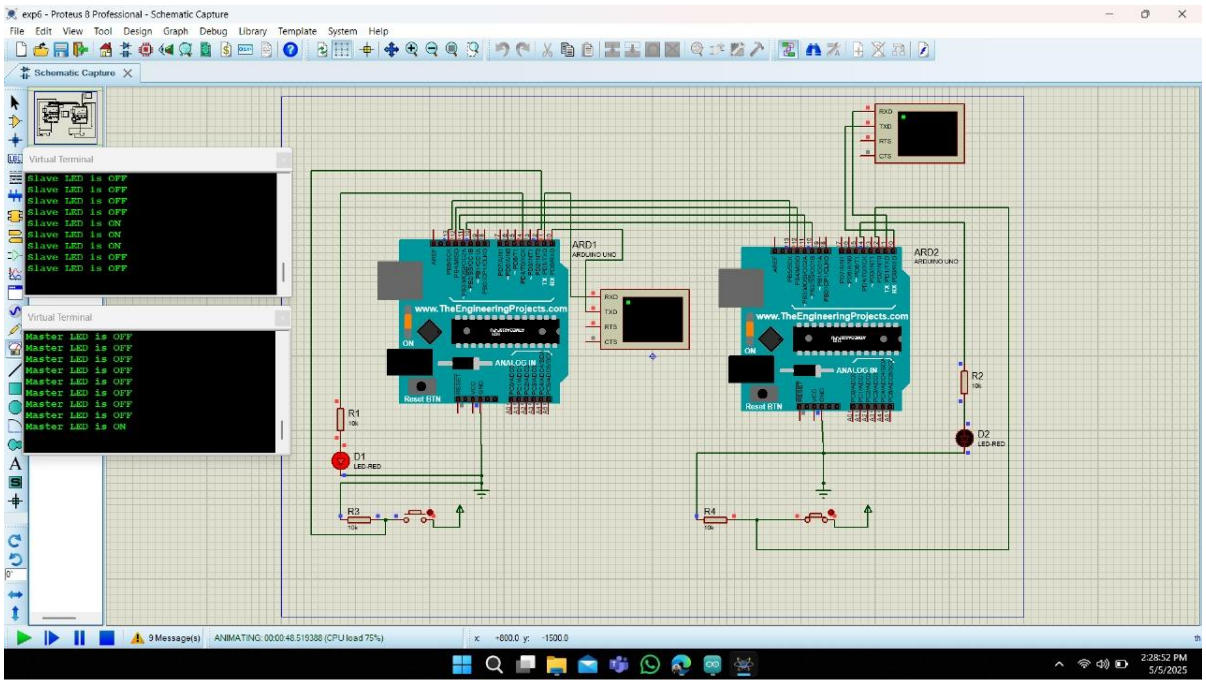
Simulation:


exp6 - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Virtual Terminal
ARD1
ARDUINO UNO
ARD2
ARDUINO UNO
R1
R2
R3
R4
D1
LED-RED
D2
LED-RED
RXD
TXD
RTS
CTS
9 Message(s)
ANIMATING: 00:00:44.429292 (CPU load 64%)
2:28:39 PM
5/5/2025

```
Slave LED is ON
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is ON
```

```
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
```

Figure-11: Slave Arduino LED turns ON when Master Arduino switch is pressed(Simulation)


exp6 - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Virtual Terminal
ARD1
ARDUINO UNO
ARD2
ARDUINO UNO
R1
R2
R3
R4
D1
LED-RED
D2
LED-RED
RXD
TXD
RTS
CTS
9 Message(s)
ANIMATING: 00:00:48.519388 (CPU load 75%)
2:28:52 PM
5/5/2025

```
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is OFF
Slave LED is ON
Slave LED is ON
Slave LED is ON
Slave LED is OFF
Slave LED is OFF
```

```
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is OFF
Master LED is ON
```

Figure-12: Master Arduino LED turns ON when Slave Arduino switch is pressed(Simulation)

Explanation: In this simulation, the Arduino Uno board, 220Ω and 10kΩ resistors, and LEDs and switches were set up according to the hardware configuration used in the lab. The program was written and verified in the Arduino IDE 2.3.5, which generated a HEX file. This file was then loaded into the Proteus simulation to simulate the behavior of the circuit.

Discussion:

This experiment focused on establishing and understanding SPI (Serial Peripheral Interface) communication between two Arduino boards, one configured as a Master and the other as a Slave. The objective was to gain practical experience in setting up SPI communication and to explore how digital data can be exchanged between two microcontrollers in real time. The task involved reading button inputs on both Arduinos and updating their respective LEDs based on the remote Arduino's button state. This setup highlights the advantages of using SPI for fast, full-duplex communication, which is particularly useful for applications requiring synchronized data exchange.

In this setup, the Master Arduino initiates the communication by sending a data byte (based on its button state) to the Slave Arduino using the SPI.transfer() function. Simultaneously, it receives a response from the Slave, which also reflects the Slave's button state. This bidirectional communication happens through a shared clock signal and four wires: MISO, MOSI, SCK, and SS. The Slave Arduino, configured to use SPI interrupt (SPI.attachInterrupt()), instantly responds to the incoming data by processing the Master's signal, updating its own LED status accordingly, and preparing a response to be sent back to the Master in the same clock cycle.

This implementation provided practical exposure to key SPI components, including data registers (SPDR), interrupts, and clock dividers. Using interrupts on the Slave side allowed immediate handling of data without polling, making the communication more efficient and responsive. Additionally, we learned how SPI communication avoids the blocking behavior seen with functions like delay() or Serial.read(), which is a critical feature for real-time systems where responsiveness and parallel processing are essential.

Here are some real-life scenarios and applications of SPI-based Arduino communication:

1. **Industrial Automation**: In factories, SPI enables Arduinos to act as controllers for various sensors and actuators, ensuring synchronized operation with minimal delay. For example, multiple temperature and pressure sensors can report to a central controller via SPI for real-time monitoring
2. **Data Logging Systems**: SPI is widely used to connect microcontrollers to SD cards or EEPROMs for fast data storage. For example, a Master Arduino could log temperature data received from several sensor Slaves into a memory device.
3. **Multi-Sensor Control in Robotics**: Robots often have many sensors and modules. Using SPI, a central controller can communicate with sensor modules like gyroscopes, accelerometers, and magnetometers for coordinated navigation and balance.
4. **Wearable and Health Devices**: Smart health devices use SPI to communicate between the main microcontroller and biometric sensors or display modules, ensuring real-time user feedback.
5. **Home Automation Systems**: SPI communication helps coordinate actions between different microcontroller units responsible for lighting, temperature control, and security, all in a fast and synchronized manner.
6. **Gaming Controllers and Displays**: Fast response and data sharing between the main board and control/display units in game consoles often rely on SPI communication for real-time input handling.
7. **Drone Flight Controllers**: Arduinos or similar boards in drones use SPI to interact with gyroscopes, barometers, and wireless communication modules, where timing and reliability are crucial.

Conclusion:

Through this experiment, we learned how to establish SPI communication between two Arduino boards by setting one as the Master and the other as the Slave. We observed how data can be exchanged in both directions in real time using the SPI.transfer() function. By connecting push buttons and LEDs to both Arduinos, we were able to send and receive digital signals based on button presses, and control LEDs accordingly. We also understood the importance of using interrupts on the Slave side to handle incoming data more efficiently. This practical task helped us better understand how SPI works and why it is useful in systems where fast and synchronized communication between devices is needed. Overall, this experiment improved our knowledge of Arduino programming, SPI protocol, and real-time data handling in embedded systems.

References:

[1] **https://www.arduino.cc/.**
[2] **https://www.educba.com**
[3] **https://www.geeksforgeeks.org//**