Objective:

Serial Peripheral Interface (SPI) is a widely used synchronous serial communication protocol that allows for high-speed data transfer between devices. The protocol is commonly used in embedded systems such as microcontrollers, to communicate with other devices, such as sensors, memory modules and displays. The SPI protocol uses four lines for communications.

In this experiment, we will demonstrate how to use the SPI protocol for communication between two Arduinos. One arduino will act as a master and the other as a slave. We will connect two LEDs and push buttons to each Arduino and use SPI communication to control the LEDs from the opposite push button. We will discuss the working methodology of SPI and how it can be used to communicate between multiple devices. By the end of the experiment, we will have gained a better understanding of SPI communication and its applications in embedded systems.

The apparatus and software name:

1| Arduino UNO (2);
2| LED (2);
3| Push Buttons (2);
4| Resistors 10K, 2.2K (2+2);
5| Breadboard;
6| Connecting ~~wri~~ wires.

Theory:

A microcontroller typically communicates with various sensors and modules using different communication protocols, such as serial communication. Serial communication is the process of transmitting data one bit at a time over a communication channel or bus. There are various types of serial communication, including UART, CAN, USB, I2C and SPI communication.

SPI is a synchronous serial communication protocol developed by Motorola in 1970. It has become a widely used protocol for interconnecting microcontrollers, sensors, and other peripherals.

SPI uses four lines for communication, including:

1| MOSI (Master output, slave Input):
The master sends data to the slave via this line.

2| MISO (Master Input, slave output):
The slave sends data to the master via this line.

3| SCK (serial clock): This line provides the clock signal for synchronization.

4| SS (slave select): This line selects the slave with wich the master wants to communicate.

In SPI communication, one device acts as the master, while the other device acts as the slave. The master initiates the communication and controls the clock signal. The slave responds to the master's requests and sends data back. SPI can have only one master, but it can have multiple slaves.

A brief Procedure:

1| Connect the LED and the Push buttons to the master board. The LED should be connected to pin 7 and the Push button should be connected to pin 2.

2| Then we included the SPI library in the code.

3| Then we define the Pin ~~unit~~ numbers of the LED and Push buttons.

4| We initialize the SPI communication using "SPI begin()".

5| Then we set the clock divider for SPI Communication.

6| Now set the slave select pin(SS) to HIGH to disconnect the master from the slave.

7| In the loop function, read the input from the Push buttons. If the Push button is Pressed, set the value of 'x' to 1 otherwise 0.

8| Send the value of 'x' to the slave and also Receive the value from the slave.

9| Set the SS pin to LOW to begin the communication.

10| If the value received from the slave is 1, turn on LED by setting pin 7 to HIGH otherwise turn it off by setting pin 7 to LOW.

Source Code:

Master/Controller Arduino Code:

sketch_mar24a | Arduino IDE 2.0.4

File Edit Sketch Tools Help

Select Board

sketch_mar24a.ino

```
//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
```

Output

sketch_mar24a | Arduino IDE 2.0.4

File Edit Sketch Tools Help

Select Board

sketch_mar24a.ino

```
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 value received from slaveu
{
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);
}
```

Output

Figure: Master/Controller Arduino Code

Slave/Peripheral Arduino Code:

sketch_mar24a | Arduino IDE 2.0.4

File Edit Sketch Tools Help

Select Board

sketch_mar24a.ino

```
//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)
{
```

Output

sketch_mar24a | Arduino IDE 2.0.4

File Edit Sketch Tools Help

Select Board

sketch_mar24a.ino

```
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);
}
}
```

Output

Figure: Slave/Peripheral Arduino Code:

Simulation:

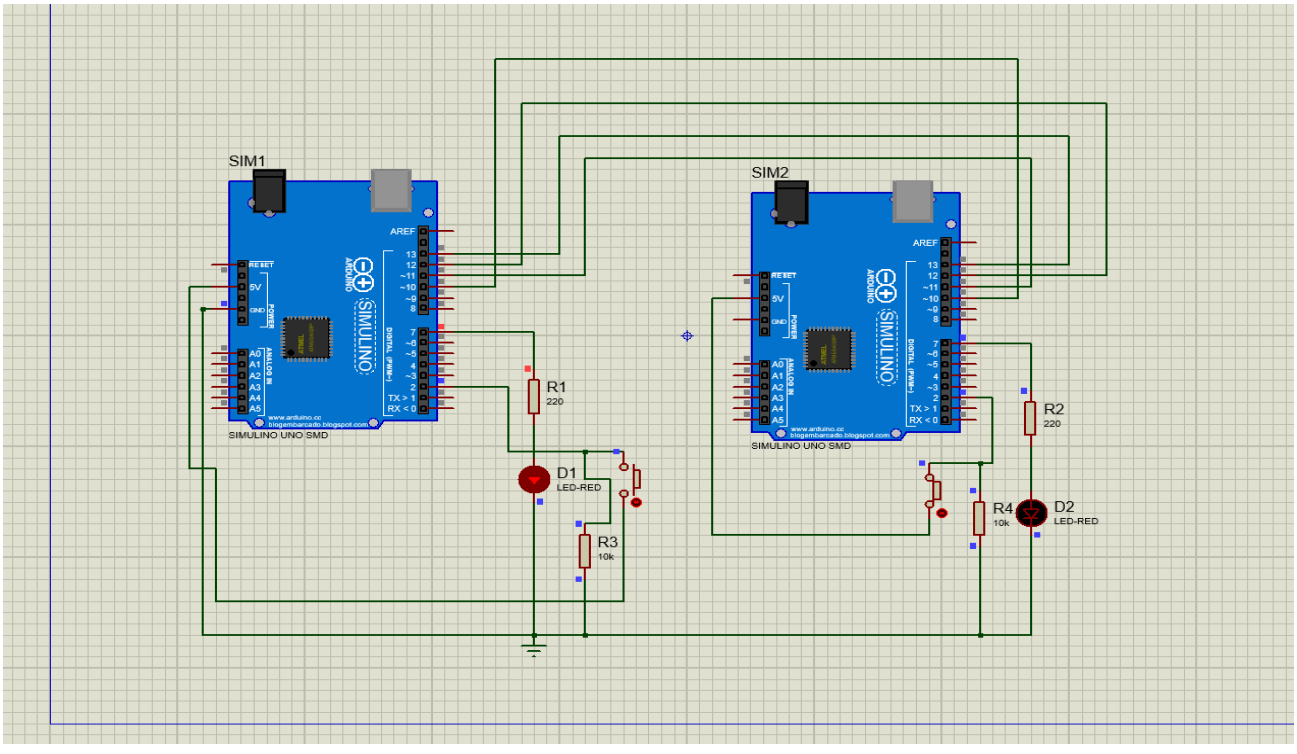

Figure: Two Arduino board's pin connections for SPI communications

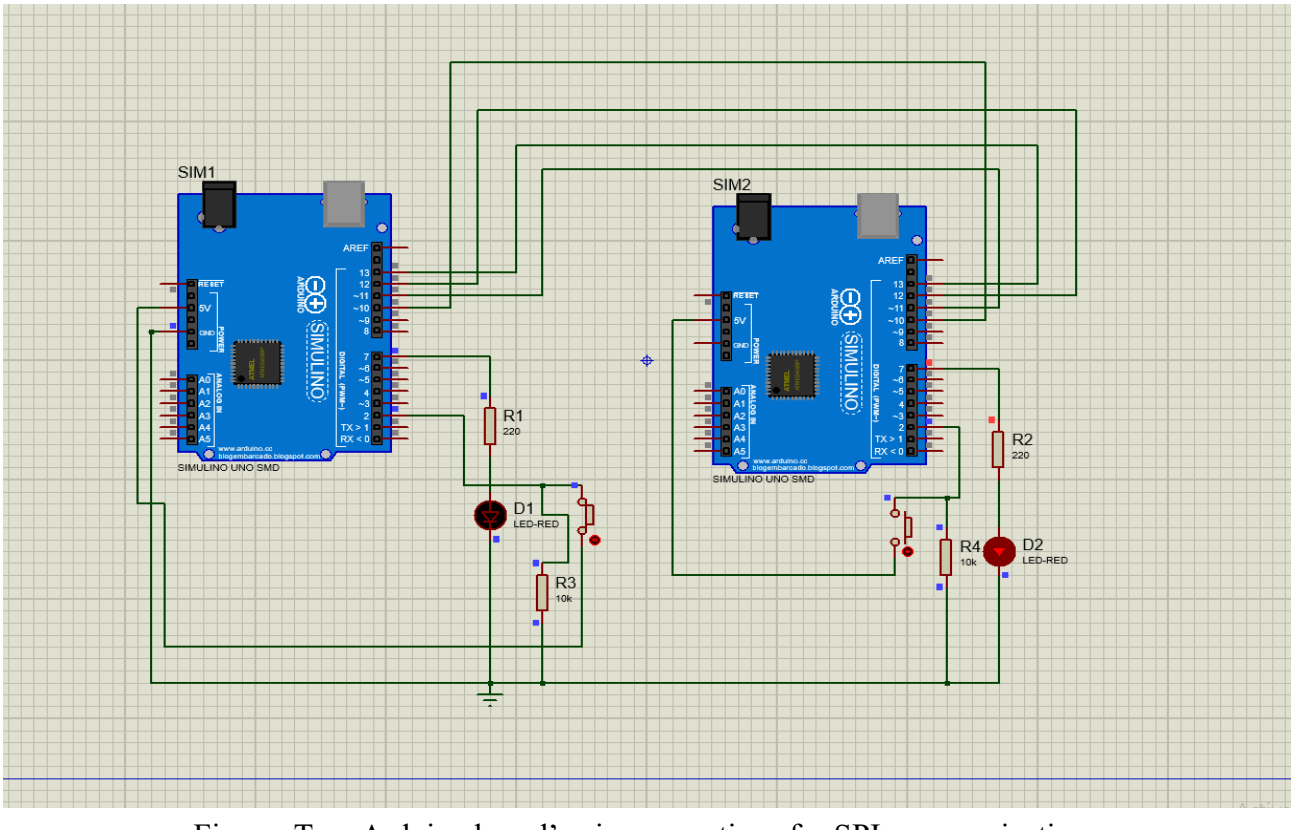

Figure: Two Arduino board's pin connections for SPI communications

Lab Simulation:

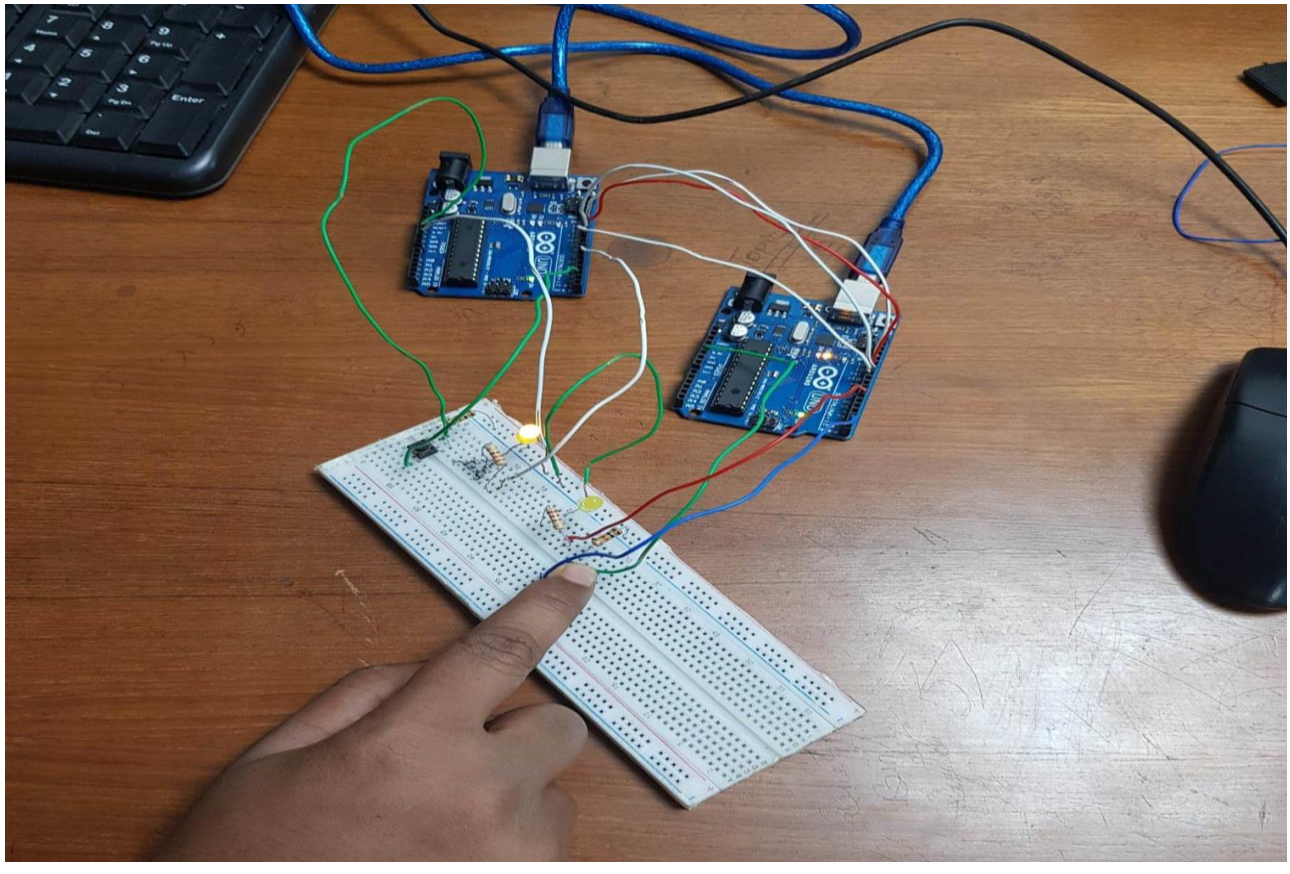

Figure: Two Arduino board's pin connections for SPI communications

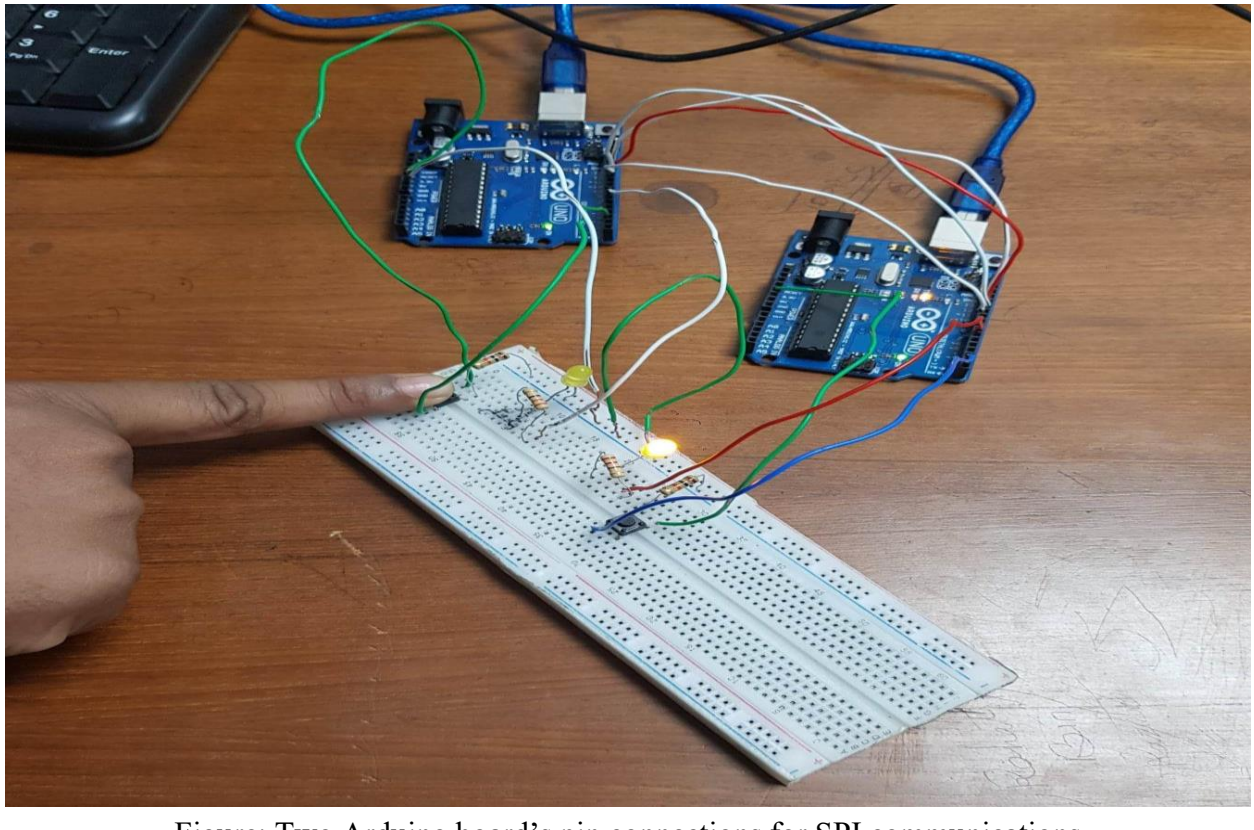

Figure: Two Arduino board's pin connections for SPI communications

Report:

All codes, scripts and proteus simulation of the blink program and traffic light system is attached above.

Discussions:

The master/controller code and slave/Peripheral code both are uses the SPI library to establish communication between two Arduinos. They sets up the serial communication at a baud rate of 115200, configures the pins as input and output, and sets the clock for SPI communication at 2MHz. The master code then continuously Reads the status of a button connected to pin 2 and sends the value to the slave via SPI. The slave code ~~ass~~ also ~~Receives data~~ sets the SPI pins as input and output and turns on SPI in slave mode. The master code also receives data from the slave via SPI and sets the LED pin to high or low depending on the Received value and the slave code sends the value to the master via SPI. The code also includes a delay of 1 second between each communication.

Overall, the both master and slave code is simple and easy to understand. It effectively uses the SPI library and sets up communication between the two Arduinos. It also demonstrates how data can be transferred between two Arduinos using SPI.

Conclusions:

The SPI communication protocol is an efficient way of transferring data between two Arduinos. The master and slave code presented here effectively demonstrate how data can be transferred using SPI. The code is simple and easy to understand and can be used as a starting point for more complex projects. Overall, the code effectively shows the capabilities of the SPI communication protocol and provides a solid foundation for more advanced projects.