INTERNET OF THINGS

ASSIGNMENT - 2

PREPARED BY

Garv Baheti

(2020BTechCSE031)

SUBMITTED TO

Mr. Divanshu Jain

Department of Computer Science Engineering
Institute of Engineering and Technology
JK Lakshmipat University Jaipur

October 03, 2022

OBJECTIVE

Explain the SPI protocol in details with suitable sketches. Also connect two Arduino boards using SPI protocol. Control the Led connected to Arduino 2 using Switch 1 connected to Arduino 1 and vice-versa.

SPI – SERIAL PERIPHERAL INTERFACE

SPI is one of the most popular serial communication protocols. SPI – Serial Peripheral Interface is an interface bus commonly used to transmit and receive the data between microcontrollers and small peripherals such as shift registers, sensors, displays, flash memory, etc. Since it is synchronous communication, it uses a separate clock line and data lines. The interface was developed by Motorola in the mid-1980s.

SPI Specification

An SPI – Serial Peripheral Interface protocol system consists of one master device and one or more slave devices. But in I2C, we can have one or more master devices and one or more slave devices. The master is providing the SPI clock and the slave is receiving the SPI clock from the master.

SPI needs 4 wires at least. But some of the recent day devices support 3 pin mode and 4pin modes also.

1. **MOSI** – Master Out Slave In
2. **MISO** – Master In Slave Out
3. **SCLK** – Serial Clock
4. **SS/CS/CE** – Salve Select/Chip Select/Chip Enable

- **MOSI** – Master Out Slave In is used to send the data from the master to the slave device.
- **MISO** – Master In Slave Out is used to receive the data from the slave to the master device.
- **SCLK** – Serial Clock is the clock that is generated by the master device.
- **SS/CS/CE** – Salve Select/Chip Select/Chip Enable line is used to select the slave device in the SPI bus interface.

SPI protocol is a Full-duplex as this protocol has separate pins for incoming data and outgoing data. So, we can transmit and receive the data at the same time.

Note: *on a slave-only device, MOSI may be labeled as* ***SDI*** *(Serial Data In) and MISO may be labeled as* ***SDO*** *(Serial Data Out).*

Data Transmission

To begin communication, the bus master configures the clock, using a frequency supported by the slave device, typically up to a few MHz. Once the clock has been configured, then the master makes the Slave select (SS/CS) pin LOW that initiates the transfer. When multiple slaves are used, the master has to make the respective slave's Slave select (SS/CS) pin LOW.

During each SPI clock cycle, full-duplex data transmission occurs. The master sends a bit on the **MOSI** pin and the slave reads it, while the slave sends a bit on the **MISO** pin and the master reads it. This sequence is maintained even when only one-directional data transfer is intended.

Transmissions normally involve two shift registers of eight bits (mostly but size varies based on the slaves), one in the master and one in the slave. Data is usually shifted out with the most significant bit (MSB) first. On the clock edge, both master and slave shift out a bit and output it on the transmission line to the counterpart. On the next clock edge, at each receiver, the bit is sampled from the transmission line and set as a new least-significant bit of the shift register. After the register bits have been shifted out and in, the master and slave have exchanged register values. If more data needs to be exchanged, the shift registers are reloaded and the process repeats. Transmission may continue for any number of clock cycles.

SPI Modes

The SPI interface defines no protocol for data exchange, limiting overhead and allowing for high-speed data streaming. We saw how the data is being transferred to the slave from the master. As we have an understanding of data communication, let's go a little bit deeper. In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. Clock polarity (**CPOL**) and clock phase (**CPHA**) are the two parameters that are used to form four unique modes to provide flexibility in communication between master and slave.

CPOL	CPHA	MODE
0	0	MODE 0
0	1	MODE 1
1	0	MODE 2
1	1	MODE 3

Advantages of SPI

- It’s way faster than asynchronous serial communication and I2C (almost twice as fast).
- No start and stop bits, so the data can be streamed continuously without interruption.
- No slave addressing mechanism like I2C.
- It has separate MISO and MOSI lines, so data can be sent and received at the same time (Full duplex).
- Not Limited to 8-bit data.
- The received hardware (slave) can be a simple shift register.
- It supports multiple slave devices.

Disadvantages of SPI

- It requires more lines (wires) than other communication methods. (UART and I2C only require 2 lines).
- There is no acknowledgment to inform that data has been successfully transferred like I2C and No error detection protocol is defined.
- The communications must be well-defined in advance (you can’t send random amounts of data whenever you want).

- The master must control all communications (slaves can’t talk directly to each other).
- It usually requires separate CS lines to each peripheral, which can be problematic if numerous slaves are needed.
- SPI only supports a single master.
- Can be used only from short distances.

Applications

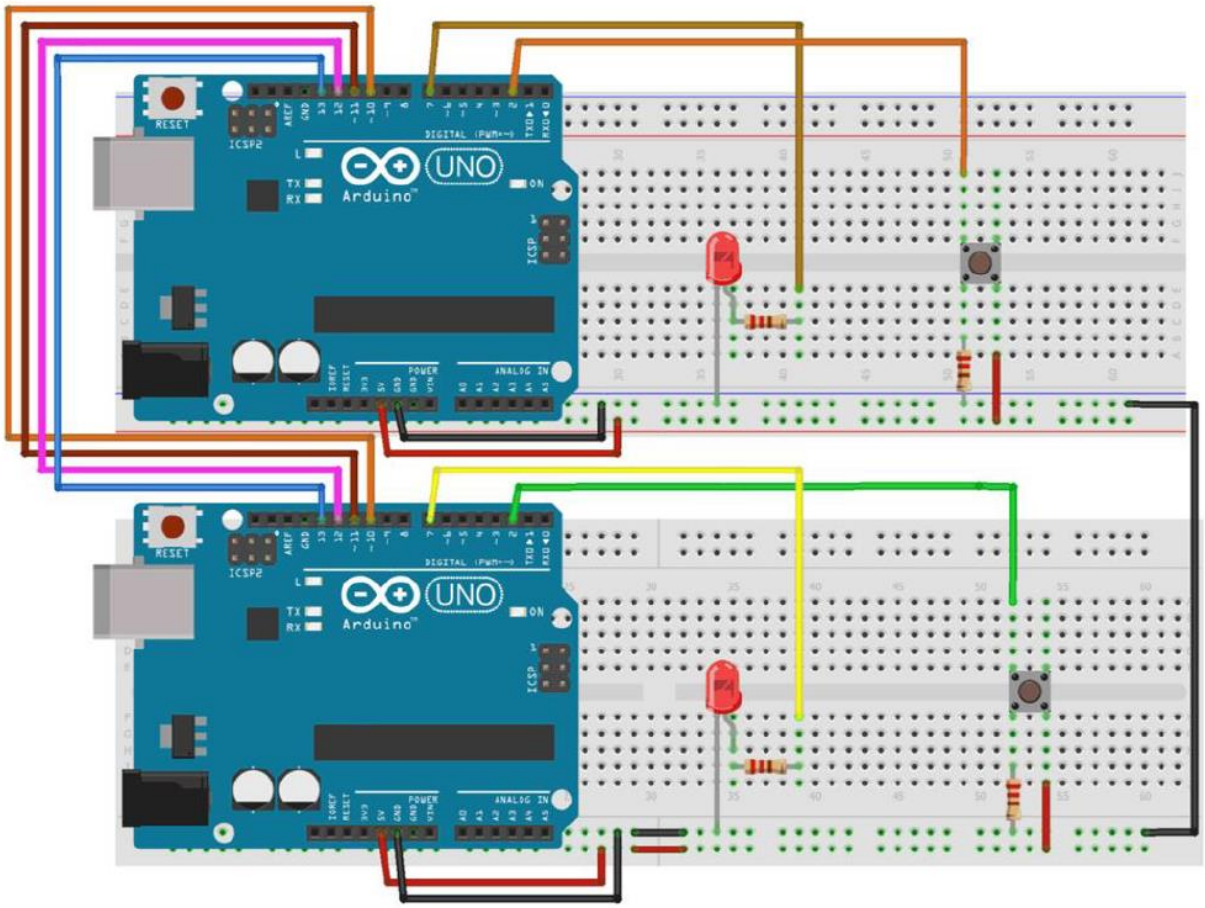
- SPI is used to talk to a variety of peripherals, such as,
    - **Sensors**: temperature, pressure, ADC, touchscreens, video game controllers
    - **Control devices**: audio codecs, digital potentiometers, DAC
    - **Camera lenses**: Canon EF lens mount
    - **Communications**: Ethernet, USB, USART, CAN, IEEE 802.15.4, IEEE 802.11, handheld video games
    - **Memory**: flash and EEPROM
    - **Real-time clocks**
    - **LCD**
    - Any **MMC** or **SD card**

I2C vs SPI

It is worth comparing the two synchronous serial communications I2C and SPI. We have posted a separate post for I2C. You can read the I2C Basics and advanced concepts.

I2C	SPI
I2C has been developed in 1982 by Philips Semiconductors (NXP).	SPI is developed by Motorola in the mid-1980s.
It supports multi masters.	It supports only one master.
It needs a maximum of 2 wires.	It needs a minimum of 4 wires and the maximum is based on the number of slaves.
It is half-duplex communication as it has only one wire for Tx and Rx.	It is full-duplex communication as it has separate wires for Rx and Tx.
It is slower than the SPI.	It is faster than the I2C.
I2C has an acknowledgment mechanism.	This doesn’t have an acknowledgment mechanism.
In this protocol, the slave has to have a unique address.	In this protocol, the slave doesn’t need an address. Master will use the SS/CS wire to select the slaves.
Due to Start, Stop, Address bits, it has overhead.	But it will directly start the data transfer. Start and Stop are not required. So, no overhead here.
I2C is better for long-distance.	SPI is better for a short distance.

CIRCUIT



SLAVE CODE

```
#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 (Have to Send data to Master IN
SPCR |= _BV(SPE); //Turn on SPI in Slave Mode
received = false;
SPI.attachInterrupt(); //Interuupt ON is set for SPI commnucation
}
ISR (SPI_STC_vect) //Inerrrput routine function
{
Slavereceived = SPDR; // Value received from master if store in variable slavereceived
received = true; //Sets received as True
}
void loop()
{ if(received) //Logic to SET LED ON OR OFF depending upon the value recerived from master
{
if (Slavereceived==1)
{
digitalWrite(LEDpin,HIGH); //Sets pin 7 as HIGH LED ON
Serial.println("Slave LED ON");

}else
{
digitalWrite(LEDpin,LOW); //Sets pin 7 as LOW LED OFF
Serial.println("Slave LED OFF");
}
buttonvalue = digitalRead(buttonpin); // Reads the status of the pin 2
if (buttonvalue == HIGH) //Logic to set the value of x to send to master
{
x=1;
}else
{
x=0;
}
Slavesend=x;
SPDR = Slavesend; //Sends the x value to master via SPDR
delay(1000);
}
}
```

Master CODE

```
#include<SPI.h> //Library for SPI

#define LED 7

#define ipbutton 2

int buttonvalue;

int x;

void setup (void)

{

void loop(void)

{

byte Mastersend,Mastereceive;

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

if(buttonvalue == HIGH) //Logic for Setting x value (To be sent to slave) depending upon input from pin 2

{

x = 1;

}

else

{

x = 0;

}

digitalWrite(SS, LOW); //Starts communication with Slave connected to master

Mastersend = x;

Mastereceive=SPI.transfer(Mastersend); //Send the mastersend value to slave also receives value from slave
if(Mastereceive == 1) //Logic for setting the LED output depending upon value received from slave
{
digitalWrite(LED,HIGH); //Sets pin 7 HIGH
Serial.println("Master LED ON");
} else
{
digitalWrite(LED,LOW); //Sets pin 7 LOW
Serial.println("Master LED OFF");
}
delay(1000);
}
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 commnuication
SPI.setClockDivider(SPI_CLOCK_DIV8); //Sets clock for SPI communication at 8 (16/8=2Mhz)
digitalWrite(SS,HIGH); // Setting SlaveSelect as HIGH (So master doesnt connnect with slave)
}
```

OUTPUT