Lab Manual for Embedded System Design

Lab No. 9

Serial Peripheral Interfacing

Objectives

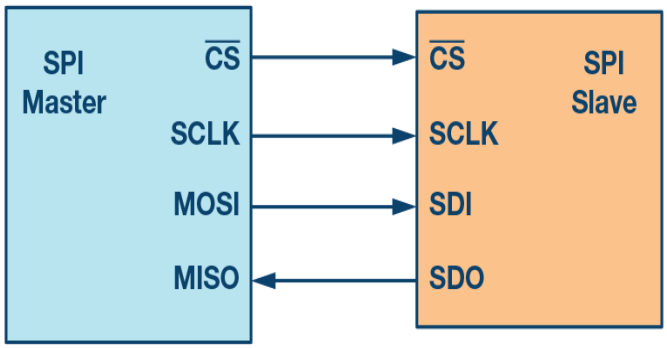
Understanding the basic concept of Serial Peripheral Interfacing and implementation of concepts on Arduino UNO and IDE.

LAB # 9
Serial Peripheral Interfacing

Introduction

Serial peripheral interface (SPI) is one of the most widely used interfaces between microcontroller and peripheral ICs such as sensors, ADCs, DACs, shift registers, SRAM, and others. This article provides a brief description of the SPI interface followed by an introduction to Analog Devices' SPI enabled switches and muxes, and how they help reduce the number of digital GPIOs in system board design.

SPI is a synchronous, full duplex master-slave-based interface. The data from the master or the slave is synchronized on the rising or falling clock edge. Both master and slave can transmit data at the same time. The SPI interface can be either 3-wire or 4-wire. This article focuses on the popular 4-wire SPI interface



SPI configuration with master and a slave.

4-wire SPI devices have four signals:

- Clock (SPI CLK, SCLK)
- Chip select (CS)
- Master out, slave in (MOSI)
- Master in, slave out (MISO)

The device that generates the clock signal is called the master. Data transmitted between the master and the slave is synchronized to the clock generated by the master. SPI devices support much higher clock frequencies compared to I^2C interfaces. Users should consult the product data sheet for the clock frequency specification of the SPI interface.

SPI interfaces can have only one master and can have one or multiple slaves. Figure 1 shows the SPI connection between the master and the slave.

The chip select signal from the master is used to select the slave. This is normally an active low signal and is pulled high to disconnect the slave from the SPI bus. When multiple slaves are used, an individual chip select signal for each slave is required from the master. In this article, the chip select signal is always an active low signal.

MOSI and MISO are the data lines. MOSI transmits data from the master to the slave and MISO transmits data from the slave to the master.

Data Transmission

To begin SPI communication, the master must send the clock signal and select the slave by enabling the CS signal. Usually chip select is an active low signal; hence, the master must send a logic 0 on this signal to select the slave. SPI is a full-duplex interface; both master and slave can send data at the same time via the MOSI and MISO lines respectively. During SPI communication, the data is simultaneously transmitted (shifted out serially onto the MOSI/SDO bus) and received (the data on the bus (MISO/SDI) is sampled or read in). The serial clock edge synchronizes the shifting and sampling of the data. The SPI interface provides the user with flexibility to select the rising or falling edge of the clock to sample and/or shift the data. Please refer to the device data sheet to determine the number of data bits transmitted using the SPI interface.

Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Arduino & Proteus	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

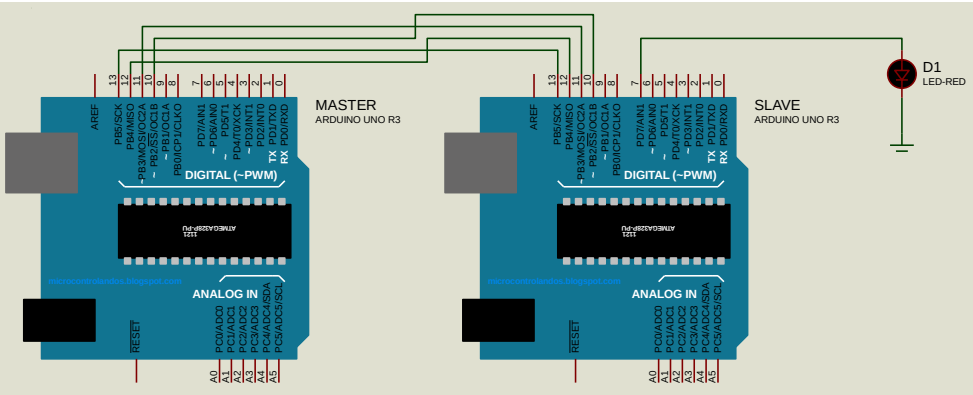
Objectives

This Lab exercise delivers the idea/concept of:

- Understand the purpose/ advantage of using Arduino UNO.
- Understand use of Serial Peripheral Interfacing.

Lab Tasks/Practical Work

1. In this task we will write a code for SPI communication between two Arduino as master and slaves. LED will blink on slave Arduino uno through Master Arduino using SPI communication.



Code (Master Arduino):

```
#include <SPI.h>

void setup (void) {
```

```
  digitalWrite(SS, HIGH); // disable Slave Select
  SPI.begin ();
  SPI.setClockDivider(SPI_CLOCK_DIV2);//divide the clock by 8
 }
 int state;
int oldstate=0;
void loop (void) {
  int c  ;
state = !oldstate;
  digitalWrite(SS, LOW); // enable Slave Select
  // send test string
  c = state;
   SPI.transfer(c);
 digitalWrite(SS, HIGH);
 oldstate = state;// disable Slave Select
 delay(500);
}
```

Code (Slave Arduino):

```
#include <SPI.h>
int buff;
volatile boolean process;

void setup (void) {
   pinMode(MISO, OUTPUT); // have to send on master in so it set as output
   SPCR |= _BV(SPE); // turn on SPI in slave mode
   process = false;
   SPI.attachInterrupt(); // turn on interrupt
   pinMode(7,OUTPUT);
}
ISR (SPI_STC_vect) // SPI interrupt routine {
{   int c = SPDR;
     buff = c;
```

```
    process = true;// read byte from SPI Data Register
}




void loop (void) {
  if (process) {
    process = false; //reset the process
   digitalWrite(7,buff);
   }
}
```

1. Using the Concept of SPI communication, write a program to control a LED on slave arduino by a SPDT switch on master arduino. Attach proteus simulation results and arduino code.