tronixstuff

tronixstuff fun and learning with electronics

- Home
- Tronixlabs
- Kit Reviews
- Reviews
- About
- Contact Us

***Categorized* | AD5204, arduino, COM-09767, education, learning electronics, lesson, MCP4162, microcontrollers, SPI, tutorial**

Tutorial: Arduino and the SPI bus

Posted on 13 May 2011. Tags: 10k, 1280, 2560, 5204, ad5204, analog, arduino, bus, clock, COM-09767, com-11440, CS, data, devices, digital, DIY, duemilanove, fade, freetronics, guide, guides, how, interface, kitten, learn, lesson, lessons, master, mega, MISO, MOSI, ohm, pinout, pot, potentiometer, PWM, SCK, sheet, slave, SPI, SS, synchronous, to, tronixstuff, tutorial, tutorials, understand, uno

Learn how to use the SPI data bus with Arduino in chapter thirty-four of a series originally titled "Getting Started/Moving Forward with Arduino!" by John Boxall – A seemingly endless tutorial on the Arduino universe. The first chapter is here, the complete series is detailed here.

[Updated 10/01/2013]

This is the first of two chapters in which we are going to start investigating the SPI data bus, and how we can control devices using it with our Arduino systems. The SPI bus may seem to be a complex interface to master, however with some brief study of this explanation and practical examples you will soon become a bus master! To do this we will learn the necessary theory, and then apply it by controlling a variety of devices. In this tutorial things will be kept as simple as possible.

But first of all, what is it? And some theory…

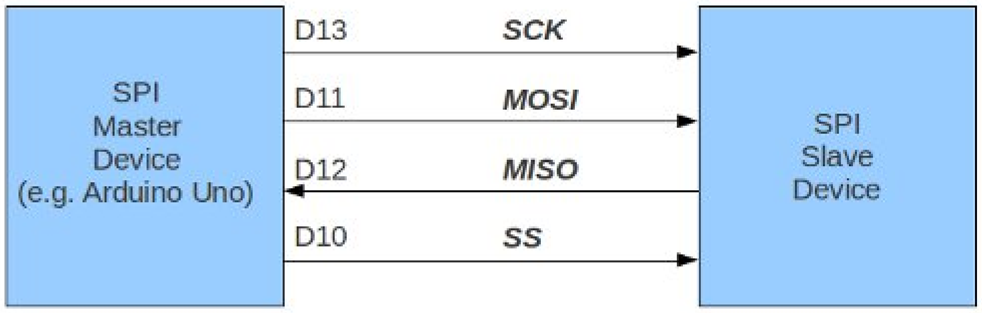
SPI is an acronym for "Serial Peripheral Interface". It is a *synchronous* serial data bus – data can travel in both directions at the same time, as opposed to (for example) the I2C bus that cannot do so. To allow synchronous data transmission, the SPI bus uses four wires. They are called:

- MOSI – *Master-out, Slave-in.* This line carries data from our Arduino to the SPI-controlled device(s);
- MISO – *Master-in, Slave out.* This line carries data from the SPI-controlled device(s) back to the Arduino;
- SS – *Slave-select.* This line tells the device on the bus we wish to communicate with it. Each SPI device needs a unique SS line back to the Arduino;
- SCK – *Serial clock.*

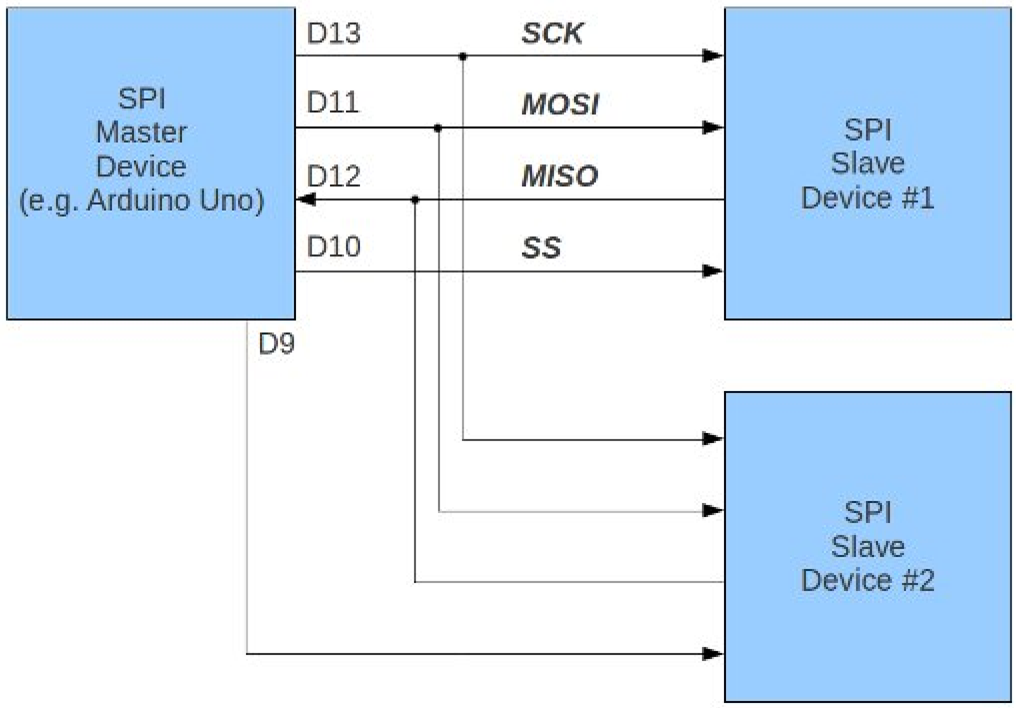
Within these tutorials we consider the Arduino board to be the master and the SPI devices to be slaves. On our Arduino Duemilanove/Uno and compatible boards the pins used are:

- SS – digital 10. You can use other digital pins, but 10 is generally the default as it is next to the other SPI pins;
- MOSI – digital 11;
- MISO – digital 12;
- SCK – digital 13;

Arduino Mega users – MISO is 50, MOSI is 51, SCK is 52 and SS is usually 53. If you are using an Arduino Leonardo, the SPI pins are on the ICSP header pins. See here for more information. You can control one or more devices with the SPI bus. For example, for one device the wiring would be:

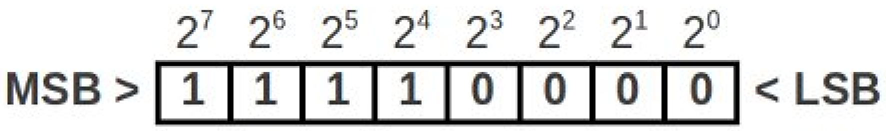


Data travels back and forth along the MOSI and MISO lines between our Arduino and the SPI device. This can only happen when the SS line is set to LOW. In other words, to communicate with a particular SPI device on the bus, we set the SS line to that device to LOW, then communicate with it, then set the line back to HIGH. If we have two or more SPI devices on the bus, the wiring would resemble the following:

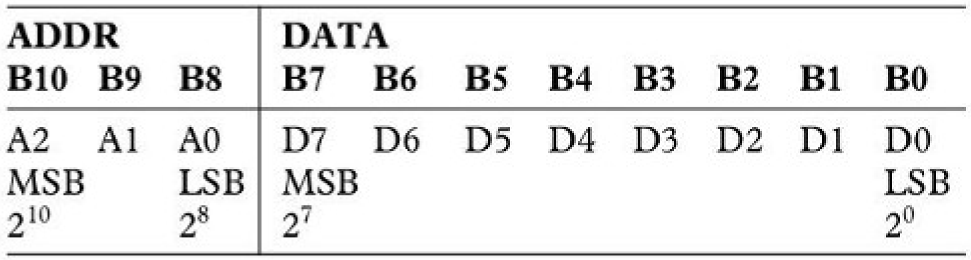


Notice how there are two SS lines – we need one for each SPI device on the bus. You can use any free digital output pin on your Arduino as an SS line. Just remember to have all SS lines high *except* for the line connected to the SPI device you wish to use at the time.

Data is sent to the SPI device in *byte* form. You should know by now that eight bits make one byte, therefore representing a binary number with a value of between zero and 255. When communicating with our SPI devices, we need to know which way the device deals with the data – MSB or LSB first. MSB (*most significant bit*) is the left-hand side of the binary number, and LSB (*least significant bit*) is the right-hand side of the number. That is:



Apart from sending numerical values along the SPI bus, binary numbers can also represent commands. You can represent eight on/off settings using one byte of data, so a device's parameters can be set by sending a byte of data. These parameters will vary with each device and should be illustrated in the particular device's data sheet. For example, a digital potentiometer IC with six pots:



ADDR			**DATA**							
B10	**B9**	**B8**	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**
A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
MSB		LSB	MSB							LSB
2^{10}		2^8	2^7							2^0

This device requires two bytes of data. The ADDR byte tells the device which of six potentiometers to control (numbered 0 to 5), and the DATA byte is the value for the potentiometer (0~255). We can use integers to represent these two values. For example, to set potentiometer number two to 125, we would send 2 then 125 to the device.

How do we send data to SPI devices in our sketches?

First of all, we need to use the SPI library. It is included with the default Arduino IDE installation, so put the following at the start of your sketch:

```
#include "SPI.h"
```

Next, in *void.setup()* declare which pin(s) will be used for SS and set them as OUTPUT. For example,

```
pinMode(ss, OUTPUT);
```

where *ss* has previously been declared as an integer of value ten. Now, to activate the SPI bus:

```
SPI.begin();
```

and finally we need to tell the sketch which way to send data, MSB or LSB first by using

```
SPI.setBitOrder(MSBFIRST);
```

or

```
SPI.setBitOrder(LSBFIRST);
```

When it is time to send data down the SPI bus to our device, three things need to happen. First, set the digital pin with SS to low:

```
digitalWrite(SS, LOW);
```

Then send the data in bytes, one byte at a time using:

```
SPI.transfer(value);
```

Value can be an integer/byte between zero and 255. Finally, when finished sending data to your device, end the transmission by setting SS high:

```
digitalWrite(ss, HIGH);
```

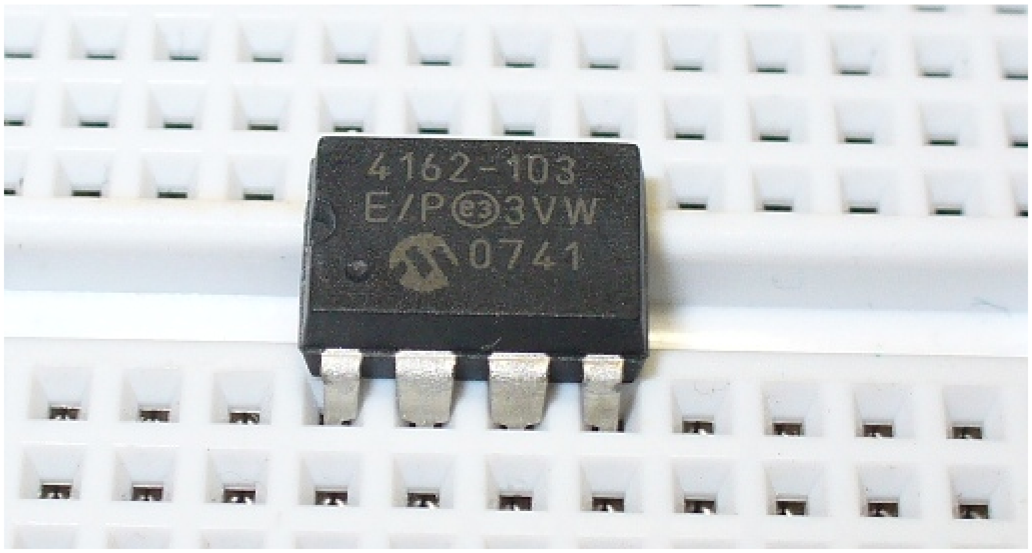
Sending data is quite simple. Generally the most difficult part for people is interpreting the device data sheet to understand how commands and data need to be structured for transmission. But with some practice, these small hurdles can be overcome.

Now for some practical examples!

Time to get on the SPI bus and control some devices. By following the examples below, you should gain a practical understanding of how the SPI bus and devices can be used with our Arduino boards.

Example 34.1

Our first example will use a simple yet interesting part – a digital potentiometer (we also used one in the I2C tutorial). This time we have a Microchip MCP4162-series 10k rheostat:



Here is the data sheet.pdf for your perusal. To control it we need to send two bytes of data – the first byte is the *control byte*, and thankfully for this example it is always zero (as the address for the wiper value is 00h [see table 4-1 of the data sheet]). The second byte is the the value to set the wiper, which controls the resistance. So to set the wiper we need to do three things in our sketch…

First, set the SS (slave select) line to low:

```
digitalWrite(10, LOW);
```

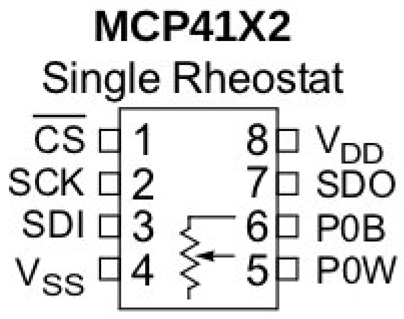
Then send the two byes of data:

```
SPI.transfer(0); // command byte
SPI.transfer(value); // wiper value
```

Finally set the SS line back to high:

```
digitalWrite(10, HIGH);
```

Easily done. Connection to our Arduino board is very simple – consider the MCP4162 pinout:



Vdd connects to 5V, Vss to GND, ~~CS~~ to digital 10, SCK to digital 13, SDI to digital 11 and SDO to digital 12. Now let's run through the available values of the MCP4162 in the following sketch:

```
/*
 Example 34.1 - SPI bus demo using a Microchip MCP4162 digital potentiometer [http://bi
 http://tronixstuff.com/tutorials > chapter 34 | CC by-sa-nc | John Boxall
*/

#include "SPI.h" // necessary library
int ss=10; // using digital pin 10 for SPI slave select
int del=200; // used for various delays

void setup()
{
  pinMode(ss, OUTPUT); // we use this for SS pin
  SPI.begin(); // wake up the SPI bus.
  SPI.setBitOrder(MSBFIRST);
  // our MCP4162 requires data to be sent MSB (most significant byte) first
}

void setValue(int value)
{
  digitalWrite(ss, LOW);
  SPI.transfer(0); // send command byte
  SPI.transfer(value); // send value (0~255)
  digitalWrite(ss, HIGH);
}

void loop()
{
  for (int a=0; a<256; a++)
  {
    setValue(a);
    delay(del);
  }
  for (int a=255; a>=0; --a)
  {
    setValue(a);
    delay(del);
  }
}
```

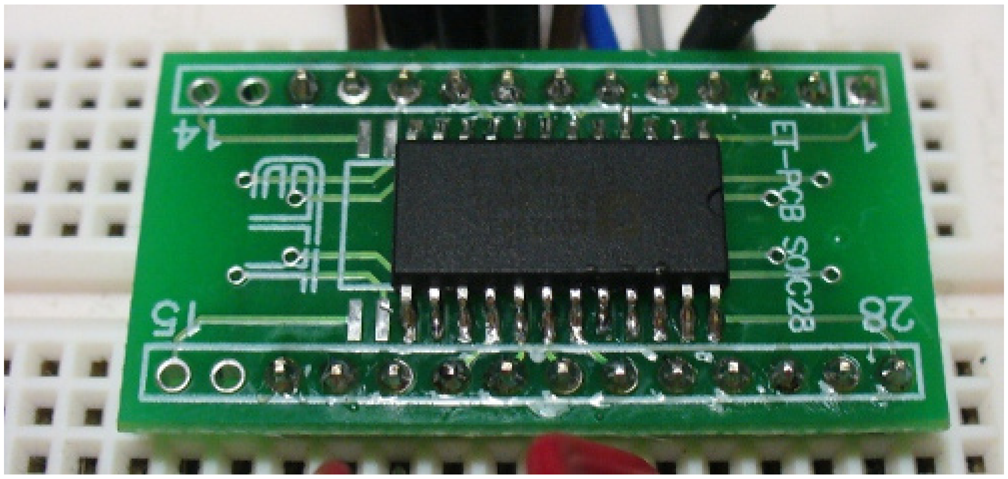
Now to see the results of the sketch. In the following video clip, a we run up through the resistance range and measure the rheostat value with a multimeter:

Before moving forward, if digital potentiometers are new for you, consider reading this short guide written by Microchip about the differences between mechanical and digital potentiometers.

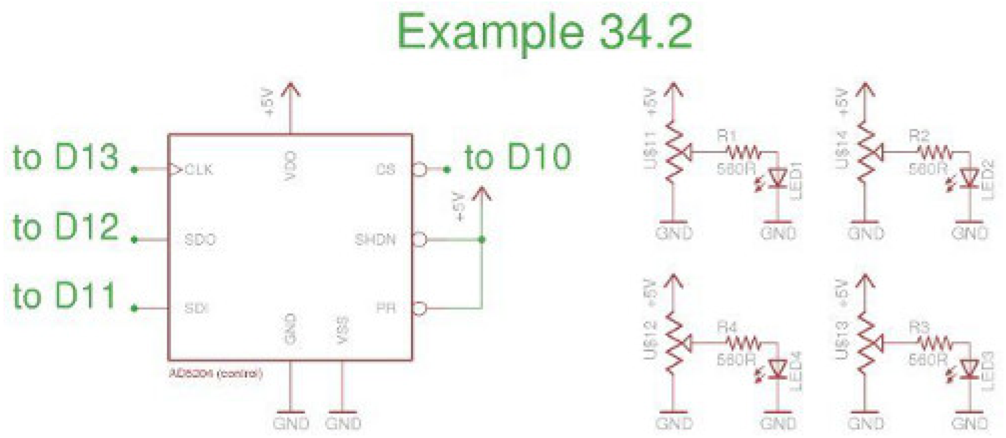
Example 34.2

In this example, we will use the Analog Devices AD5204 four-channel digital potentiometer (data sheet.pdf). It contains four 10k ohm linear potentiometers, and each potentiometer is adjustable to one of 256 positions. The settings are *volatile*, which means they are not remembered when the power is turned off. Therefore when power is applied the potentiometers are all pre set to the middle of the scale. Our example is the SOIC-24 surface mount example, however it is also manufactured in DIP format as well.

To make life easier it can be soldered onto a SOIC breakout board which converts it to a through-hole package:



In this example, we will control the brightness of four LEDs. Wiring is very simple. Pinouts are in the data sheet.pdf.



And the sketch:

```
// Example 34.2
#include <SPI.h> // necessary library
int ss=10; // using digital pin 10 for SPI slave select
int del=5; // used for fading delay
void setup()
{
 pinMode(ss, OUTPUT); // we use this for SS pin
 SPI.begin(); // wake up the SPI bus.
 SPI.setBitOrder(MSBFIRST);
 // our AD5204 requires data to be sent MSB (most significant byte) first. See data she
 allOff(); // we do this as pot memories are volatile
}
void allOff()
// sets all potentiometers to minimum value
{
 for (int z=0; z<4; z++)
 {
 setPot(z,0);
 }
}
void allOn()
// sets all potentiometers to maximum value
```

```
{
 for (int z=0; z<4; z++)
 {
 setPot(z,255);
 }
}
void setPot(int pot, int level)
// sets potentiometer 'pot' to level 'level'
{
 digitalWrite(ss, LOW);
 SPI.transfer(pot);
 SPI.transfer(level);
 digitalWrite(ss, HIGH);
}
void blinkAll(int count)
{
 for (int z=0; z
void indFade()
{
 for (int a=0; a<4; a++)
 {
 for (int l=0; l<255; l++)
 {
 setPot(a,l);
 delay(del);
 }
 for (int l=255; l>=0; --l)
 {
 setPot(a,l);
 delay(del);
 }
 }
}
void allFade(int count)
{
 for (int a=0; a<count; a++)="" {="" for="" (int="" l="0;" l<255;="" l++)="" setpot(0,l
 {
 setPot(0,l);
 setPot(1,l);
 setPot(2,l);
 setPot(3,l);
 delay(del);
 }
 }
}
void loop()
{
 blinkAll(3);
 delay(1000);
 indFade();
 allFade(3);
}
```

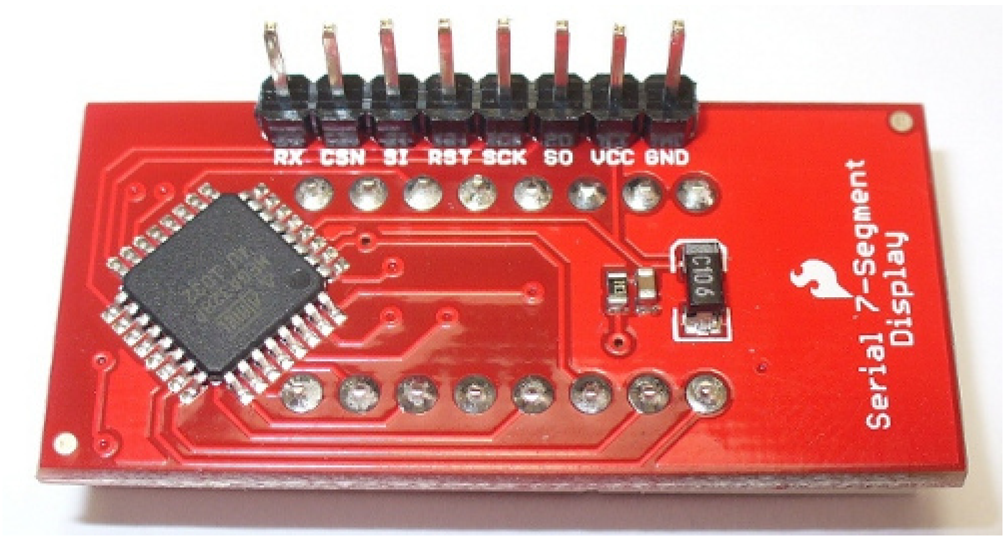
The function *allOff()* and *allOn()* are used to set the potentiometers to minimum and maximum respectively. We use *allOff()* at the start of the sketch to turn the LEDs off. This is necessary as on power-up the wipers are generally set half-way. Furthermore we use them in the *blinkAll()* function to … blink the LEDs. The function *setPot()* accepts a wiper number (0~3) and value to set that wiper (0~255). Finally the function *indFade()* does a nice job of fading each LED on and off in order – causing an effect very similar to pulse-width modulation.

Finally, here it is in action:

Example 34.3

In this example, we will use use a four-digit, seven-segment LED display that has an SPI interface. Using such a display considerably reduces the amount of pins required on the micro controller and also negates the use of shift register ICs which helps reduce power consumption and component count. The front of our example:

and the rear:



Thankfully the pins are labelled quite clearly. Please note that the board *does not* include header pins – they were soldered in after receiving the board. Although this board is documented by Sparkfun there seems to be issues in the operation, so instead we will use a sketch designed by members of the Arduino forum. Not wanting to ignore this nice piece of hardware we will see how it works and use it with the new sketch from the forum.

Again, wiring is quite simple:

- Board GND to Arduino GND
- Board VCC to Arduino 5V
- Board SCK to Arduino D12
- Board SI to Arduino D11
- Board CSN to Arduino D10

The sketch is easy to use, you need to replicate all the functions as well as the library calls and variable definitions. To display numbers (or the letters A~F) on the display, call the function

```
write_led(a,b,c);
```

where a is the number to display, b is the base system used (2 for binary, 8 for octal, 10 for usual, and 16 for hexadecimal), and c is for padded zeros (0 =off, 1=on). If you look at the *void loop()* part of the example sketch, we use all four number systems in the demonstration. If your number is too large for the display, it will show **OF** for overflow. To control the decimal points, colon and the LED at the top-right the third digit, we can use the following:

```
write_led_decimals(1); // left-most decimal point
write_led_decimals(2);
write_led_decimals(4);
write_led_decimals(8); // right-most decimal point
write_led_decimals(16); // colon LEDs
write_led_decimals(32); // apostrophe LED
write_led_decimals(0); // off
```

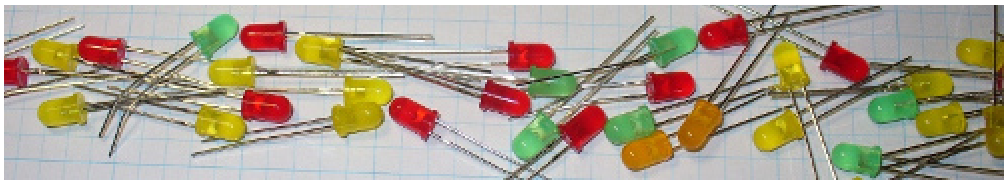
After all that, here is the demonstration sketch for your perusal:

```
/*
```

```
 Example 34.3 - SPI bus demo using SFE 4-digit LED display [http://bit.ly/ixQdbT]
 http://tronixstuff.com/tutorials > chapter 34
 Based on code by Quazar & Busaboi on Arduio forum - http://bit.ly/iecYBQ
*/
#define DATAOUT 11 //MOSI
#define DATAIN 12 //MISO - not used, but part of builtin SPI
#define SPICLOCK 13 //sck
#define SLAVESELECT 10 //ss
char spi_transfer(volatile char data)
{
 SPDR = data; // Start the transmission
 while (!(SPSR & (1<
 {
 };
 return SPDR; // return the received byte
}
void setup()
{
 byte clr;
 pinMode(DATAOUT, OUTPUT);
 pinMode(DATAIN, INPUT);
 pinMode(SPICLOCK, OUTPUT);
 pinMode(SLAVESELECT, OUTPUT);
 digitalWrite(SLAVESELECT, HIGH); //disable device
 SPCR = (1<
 clr=SPSR;
 clr=SPDR;
 delay(10);
 write_led_numbers(0x78,0x78,0x78,0x78); //Blank display
 write_led_decimals(0x00); // All decimal points off
}
void write_led_decimals(int value)
{
 digitalWrite(SLAVESELECT, LOW);
 delay(10);
 spi_transfer(0x77); // Decimal Point OpCode
 spi_transfer(value); // Decimal Point Values
 digitalWrite(SLAVESELECT, HIGH); //release chip, signal end transfer
}
void write_led_numbers(int digit1, int digit2, int digit3, int digit4)
{
 digitalWrite(SLAVESELECT, LOW);
 delay(10);
 spi_transfer(digit1); // Thousands Digit
 spi_transfer(digit2); // Hundreds Digit
 spi_transfer(digit3); // Tens Digit
 spi_transfer(digit4); // Ones Digit
 digitalWrite(SLAVESELECT, HIGH); //release chip, signal end transfer
}
void write_led(unsigned short num, unsigned short base, unsigned short pad)
{
 unsigned short digit[4] = {
 0, ' ', ' ', ' ' };
 unsigned short place = 0;
if ( (base<2) || (base>16) || (num>(base*base*base*base-1)) ) {
 write_led_numbers(' ', 0x00, 0x0f, ' '); // indicate overflow
 }
 else {
 while ( (num || pad) && (place<4) ) {
 if ( (num>0) || pad )
 digit[place++] = num % base;
 num /= base;
 }
 write_led_numbers(digit[3], digit[2], digit[1], digit[0]);
 }
}
void pointDemo()
{
 write_led_decimals(1);
 delay(1000);
 write_led_decimals(2);
 delay(1000);
 write_led_decimals(4);
 delay(1000);
 write_led_decimals(8);
 delay(1000);
 write_led_decimals(16);
 delay(1000);
 write_led_decimals(32);
 delay(1000);
 write_led_decimals(0); // non-digits all off
}
void loop()
{
 pointDemo();
 delay(500);
 for (int i = 0; i < 100; i++) {
 write_led (i,10,1);
 delay(25);
 }
 delay(500);
 for (int i = 100; i >=0; --i) {
 write_led (i,10,0);
 delay(25);
 }
 delay(500); // now binary
 for (int i = 0; i < 16; i++) {
 write_led (i,2,0);
 delay(100);
 }
 delay(500);
 for (int i = 15; i >=0; --i) {
 write_led (i,2,0);
 delay(100);
 }
 delay(500); // now octal
 for (int i = 0; i < 500; i++) {
 write_led (i,8,0);
 delay(50);
 }
 delay(500);
 // now hexadecimal
 for (int i = 20000; i < 22000; i++) {
 write_led (i,16,0);
 delay(50);
 }
 delay(500);
}
```

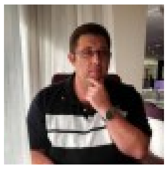
And a short video of the demonstration:

So there you have it – hopefully an easy to understand introduction to the world of the SPI bus and how to control the devices within. As always, now it is up to you and your imagination to find something to control or get up to other shenanigans. In the next SPI article we will look at reading and writing data via the SPI bus.



In the meanwhile have fun and keep checking into tronixstuff.com. Why not follow things on twitter, Google+, subscribe for email updates or RSS usng the links on the right-hand column? And join our friendly Google Group – dedicated to the projects and related items on this website. Sign up – it's free, helpful to each other – and we can all learn something.

Bio Latest Posts



John Boxall

Person. Author of http://arduinoworkshop.com Director of http://tronixlabs.com.au Rare updater of http://tronixstuff.com VK3FJBX

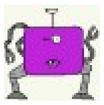
Like this:

Like

2 bloggers like this.

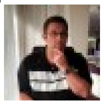
← Discovering Arduino's internal EEPROM lifespan
Kit review – nootropic design Hackvision →

32 Responses to "Tutorial: Arduino and the SPI bus"

1.  *Sergegsx* says:
   May 26, 2011 at 8:22 pm

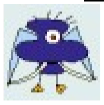
   Hi, first of all congrats on a series of tutorials which are high-quality material.
   If you have the time and the interest, it would be very useful to see the code on how to use 2 SPI devices on the same script by changing to low the CS pin. Ive tried to do this with the ethernet shield official of arduino and was not successful so it would be nice to see an example.
   It will also come to hand in my current project.
   thanks and once again, congrats.

   Reply

   -  *John Boxall* says:
     May 27, 2011 at 3:40 pm

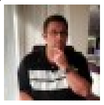
     Hello and thanks for your feedback.
     There is a known issue with the genuine Arduino Ethernet shield and SPI due to the Wiznet chip not relinquishing the SPI bus properly.
     Easiest way to fix the problem is to get another shield like http://www.freetronics.com/products/ethernet-shield-with-poe which is designed to solve the problem.
     In the next couple of weeks I will be writing more about SPI bus.
     cheers
     john

     Reply

2.  *PRABHAKARAN RAJA* says:
   January 13, 2012 at 9:13 pm

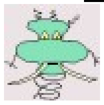
   hello sri ,, how can i overlap on led dot matrix board using HT1632c …
   already i know how it's work and all details ,,, i need concept of overlapping on HT 1632c ,,,,,

   Reply

   -  *John Boxall* says:
     January 14, 2012 at 9:10 am

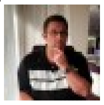
I haven't used that part so cannot help you.
John

Reply

3.  *Nick* says:
May 4, 2012 at 2:19 am

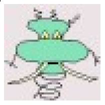
   John,
   Great tut! I am having an issue talking to my digital pot. It is in the same family as the one you used, I have a MCP4161. Can't seem to get it to vary resistances for the life of me. Any suggestions?

   Reply

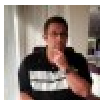
   -  *John Boxall* says:
     May 4, 2012 at 10:26 am

     Thanks, glad you like it. Troubleshooting is hard without being there so to speak, however the usual things come to mind. Check your wiring, especially if using a solderless breadboard. Also note the pot pins are different on an MCP4161 – you use pins 5~7. See page 1 of the data sheet.
     cheers
     John

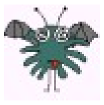
     Reply

     -  *Nick* says:
       May 5, 2012 at 6:36 am

       John,
       Thanks for the help! Got it working fine. Next, more odd question. Is there anyway to change which pin the MOSI uses. I'm altering some timers as well for my project and have pin 11 tied up there. Is it at all possible to change the SPI bus pins?

     -  *John Boxall* says:
       May 8, 2012 at 8:54 am

       Easiest way would be to modify your project to free up D11 for SPI use. Otherwise you could change to an Arduino Mega/etc. as SPI is on pins 50~53.

4.  *Jose* says:
May 22, 2012 at 9:22 pm

Hi John:
can you make a tutorial on SPI with accelerometers?

Reply

- *John Boxall* says:
  May 23, 2012 at 8:45 am

  What is the model number of the accelerometer?

  Reply

5. *Clark Darin Gozon* says:
June 20, 2012 at 3:21 pm

Hi John,

Im new to SPI also with arduino programing… In your tutorial you have discuss about sending data from arduino to SPI devices… my case is recieving data from SPI device… I am planing to use CS5463 Single Phase, Bi-directional Power/Energy IC for my project. The chip was designed to accurately measure instantaneous current and voltage, and calculate VRMS, IRMS, instantaneous power, apparent power, active power, and reactive power for single-phase, 2- or 3-wire power metering applications.

CS5463 also uses PSI bus and i had hard times understanding the data sheet. Could you help me with this… thanx

Regard,
Clark

Reply

- *John Boxall* says:
  June 20, 2012 at 10:25 pm

  SPI tutorial part 2 deals with receiving data from SPI devices –
  http://tronixstuff.com/2011/06/15/tutorial-arduino-and-the-spi-bus-part-ii/

  Reply

6. *Jordan Whitleyq* says:
August 6, 2012 at 8:12 am

John,
I am really enjoying your tutorials so far, they are incredibly helpful and awesome! I just uploaded the code above for the single rheostat, and I am having a few problems. First off, where are you taking your measurements for your resistance? I have just run two fly wires off

of pins 5 and 6 on the IC, and clipped my multimeter leads to both of the wires. Whenever there is no power, I am getting a resistance of somewhere around 8Mohm, now whenever I turn the power on, I am getting a constant 5.07Kohm resistance, it is not fluctuating whatsoever. I am using the Arduino Mega 2560, do you think this could be the problem somehow? If you could give me your thoughts on this I would greatly appreciate it, and also where you are taking your readings from. Thank you very much! Keep up all of the great work!

Reply

- *Jordan Whitley* says:
  August 6, 2012 at 8:17 am

  Actually now that I look at it, the MEGA pins for SPI are located in a different area. The pins are 50(MISO), 51(MOSI), 52(SCK), and 53(SS). Could you please just tell me which pins need to be connected to what pins on the IC? Thank you very much.

  Reply

  - *John Boxall* says:
    August 6, 2012 at 8:26 am

    Vdd connects to 5V, Vss to GND, CS to digital 53, SCK to digital 52, SDI to digital 51 and SDO to digital 50.

- *John Boxall* says:
  August 6, 2012 at 8:21 am

  Yes, measure resistance over 5 and 6.

  Reply

7. *Jordan Whitley* says:
   August 7, 2012 at 9:42 am

   Thanks a lot John! These tutorials are extremely helpful! Keep up the great work!

   Reply

8. *John Boxall* says:
   September 19, 2012 at 11:03 am

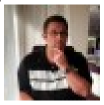
   Interesting idea, I'll put it on the list.

   Reply

9. *Mike* says:
   February 12, 2013 at 12:31 pm

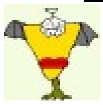
   John, thank you very much for your tutorials, I posted a link on Google+ Arduino forum for others to find. I am looking to port some Adafruit SPI display code to the Due and their implementation of SPI is very '328 specific, like your Sparkfun display code. ..
   The Due doesn't like register manipulations like the 328 so I'm looking to use Arduino SPI library calls. If you have some insight to what's being done here I'd appreciate it. Thanks, Mike

   Reply

   -  *John Boxall* says:
     February 12, 2013 at 3:58 pm

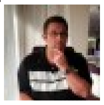
     You'd be better off posting all that in the Adafruit support forums.

     Reply

10.  *Ashfaque* says:
    April 27, 2013 at 5:06 am

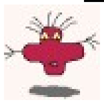
    I want to use a Microchip 25LC1024 SPI EEPROM with my arduino DUE. Can you provide me instructions or a tutorial. I have found some forums on the web discussing this, but I can't find anything specific to the DUE.

    Reply

    -  *John Boxall* says:
      April 27, 2013 at 10:50 am

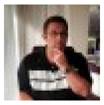
      Try http://arduino.cc/forum/index.php/topic,63655.0.html. If you actually want me to spend the time to write a custom tutorial, please send a money order for $264 to John Boxall, PO Box 5435 Clayton 3168. I'll write it up and send you a tax invoice.

      Reply

11.  *Mihailo* says:
    May 27, 2013 at 10:22 pm

    In the void setPot() function, can you explain a little more about SPI.transfer(pot), specifically the parameter. Where on the data sheet do we know that we can go from 0 to 4? To my understanding, you first "transfer" the address, and then the value. So how are 0,1,2,3,4 referencing the addresses that belong to the four digital potentiometers?

    Reply

-  John Boxall says:
  May 28, 2013 at 6:37 pm

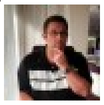
  There are only four pots, numbered (and referenced as) 0 to 3. Data addressing is explained in table one, page eight of the data sheet. We send two bytes to the AD5204, however it only reads the least three bits of the first byte (which is the pot number to set), then the second byte is the pot value.

  Reply

12. *sabjorn* says:
May 30, 2013 at 5:50 am

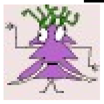
Can you do a tutorial for the MCP 41X1?
There are a few reasons why:
1 – The SDI and SDO are on one pin. Getting that to work with an arduino has been a pain for me.
2 – This chip has the ability to disconnect any of it's terminals (TCON). This I am also having trouble with. It may be related to the first point.

Reply

-  John Boxall says:
  May 30, 2013 at 11:10 am

  I don't have any in stock, so not at the moment. This may be of interest – https://github.com/declanshanaghy/ArduinoLibraries/tree/master/MCP4161

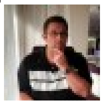
  Reply

13.  *Pete G* says:
August 6, 2013 at 11:31 pm

Hi John

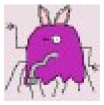
What voltage levels are the SPI Atmega 2560 outputs.

Reply

-  John Boxall says:
  August 7, 2013 at 9:30 am

  Well it's just high and low, so depends on the supply voltage.

  Reply

14.  *francesco* says:
    November 30, 2014 at 6:35 pm

    Good morning,

    my name is Francesco.

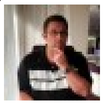
    A year ago i bought a led panel "sure electronics P4 32X8 3208 red led
    dot matrix unit board spi like" and the "Arduino UNO" but i
    can't
    understand how to use the panel program.

    I would like to add some buttons and potentiometers to the system.
    For example: pressing the button "1" display lights up with the
    saved
    name wich runs for 30 seconds at least. Pressing another button display
    lights up with another saved name wich also runs for 30 seconds at least
    and so on for the other buttons wich have to recall the saved words.
    I also would like to know how to add two potentiometers: on for slide
    speed control and one just for time control.

    I'm waiting for your answer and i thank you in advance.

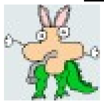
    Cordially,
    Francesco

    Reply

    -  *John Boxall* says:
      December 1, 2014 at 11:49 am

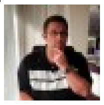
      We can't spend time to debug your individual projects. Either ask the supplier for
      support, or talk to this person who seems to use them a lot:
      http://timewitharduino.blogspot.com

      Reply

15.  *kevenlaker* says:
    December 9, 2014 at 8:00 pm

    You may need [crappy shield from ElecFreaks]: [URL deleted]

    Reply

    -  *John Boxall* says:
      December 9, 2014 at 9:15 pm

Stop spamming my website.

Reply

Trackbacks/Pingbacks

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit Comment

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.

Visit tronixlabs.com

Helping you make it with Australia's best value for supported hobbyist electronics from adafruit, Altronics, DFRobot, Freetronics, Pololu and more!

Subscribe via email

Receive notifications of new posts by email.

Email Address

Subscribe

Search the site

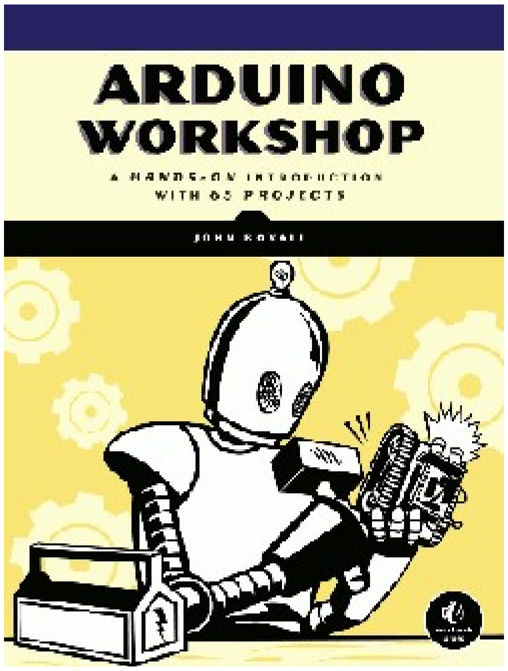
Search

tronixstuff forum

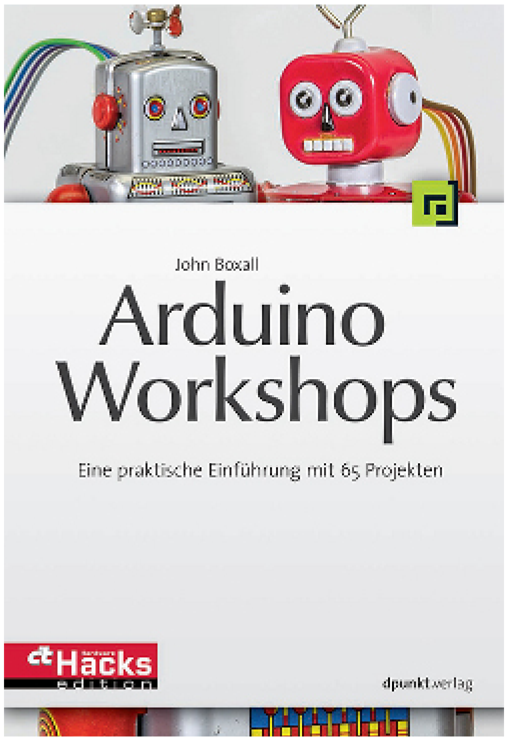
Why not join our moderated discussion forum?

Arduino Tutorials

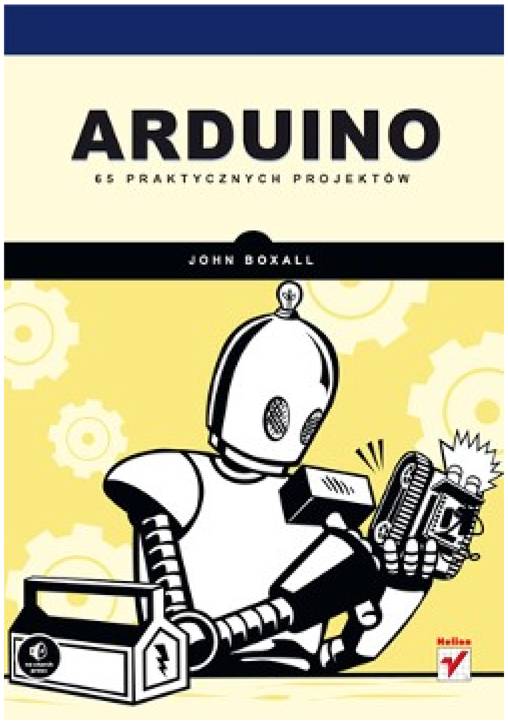
The Arduino Book



Für unsere deutschen Freunde



Dla naszych polskich przyjaciół ...



Australian Electronics!

Buy and support Silicon Chip - Australia's only Electronics Magazine.

Interesting Sites

Amazing Arduino Shield Directory
David L. Jones' eev blog
Silicon Chip magazine Always a great read!
Talking Electronics
Dangerous Prototypes
The Amp Hour podcast
Superhouse.tv High-tech home renovation

Use of our content…