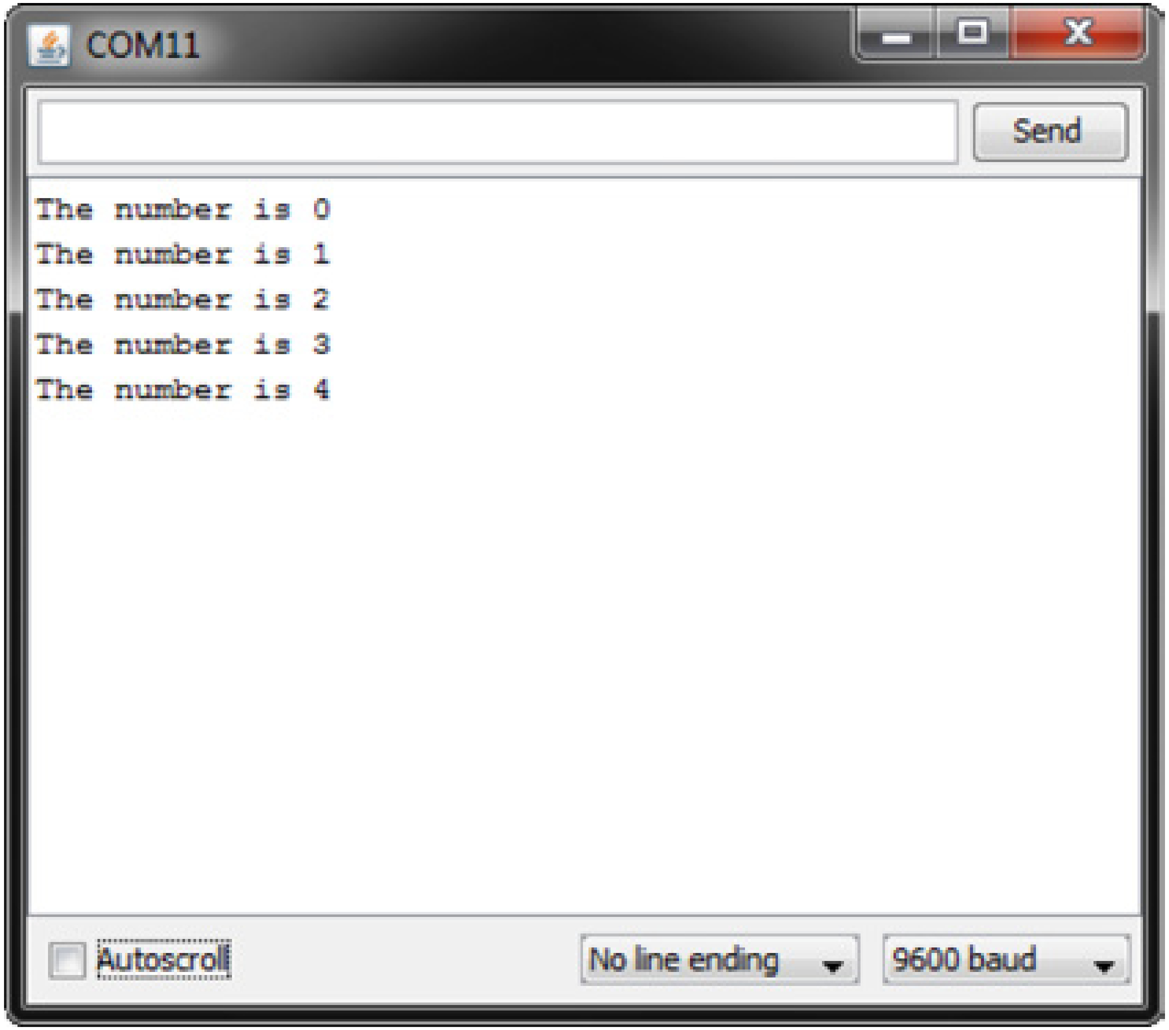
Embedded Systems

Serial Communications

Introduction:

- Serial communications provide an easy and flexible way for your Arduino board to interact with your computer and other devices.
- This chapter explains how to send and receive information using this capability.
- Serial communications are also a handy tool for debugging. You can send debug messages from Arduino to the computer and display them on your computer screen or an external LCD display.

Introduction:



Serial Hardware:

- Serial hardware sends and receives data as electrical pulses that represent sequential bits.
- The zeros and ones carrying the information, which makes up a byte, can be represented in various ways.
- The scheme used by Arduino is 0 volts to represent a bit value of 0, and 5 volts (or 3.3 volts) to represent a bit value of 1.

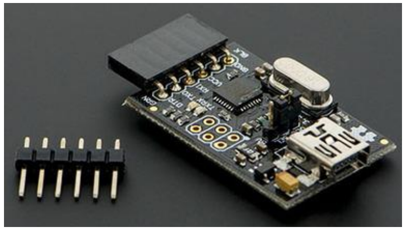
> Using 0 volts (for 0) and 5 volts (for 1) is very common. This is referred to as the *TTL level* because that was how signals were represented in one of the first implementations of digital logic, called Transistor-Transistor Logic (TTL).

Serial Hardware:

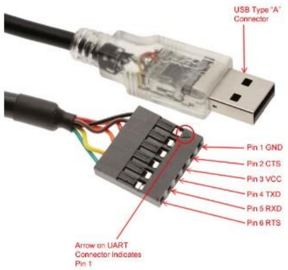
- **Some popular USB adapters include:**

✓ Mini USB Adapter

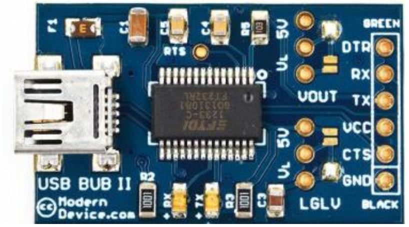
USB Serial Light Adapter



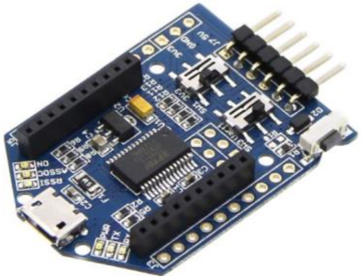
✓ FTDI USB TTL Adapter



Modern Device U

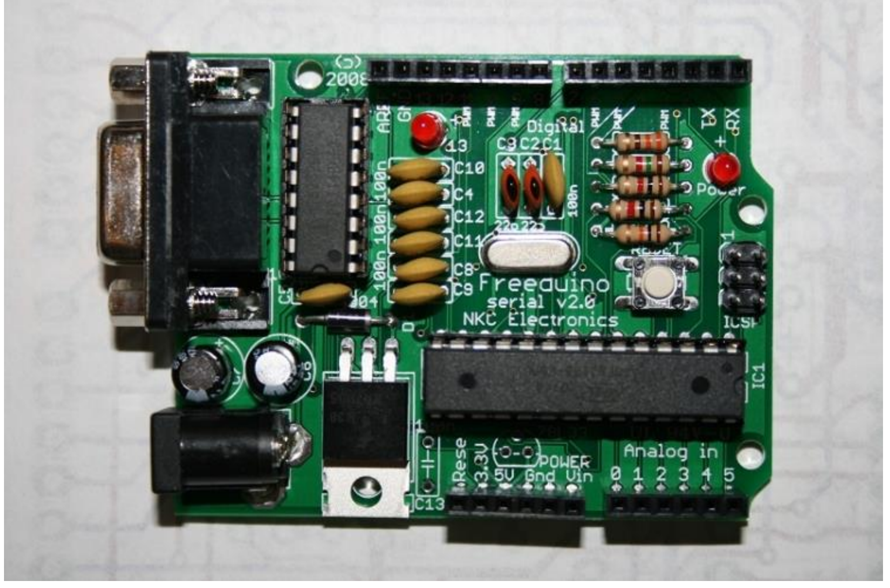


✓ Seeed studio UartSBee



Serial Hardware:

- Some serial devices use the RS-232 standard for serial connection. These usually have a nine-pin connector, and an adapter is required to use them with the Arduino. RS-232 is an old and venerated communications protocol that uses voltage levels not compatible with Arduino digital pins.



Serial Hardware:

- The Arduino Mega has four hardware serial ports that can communicate with up to four different serial devices. Only one of these has a USB adapter built in (you could wire a USB-TTL adapter to any of the other serial ports). Table 4-1 shows the port names and pins used for all Mega serial ports.

Table 4-1. Arduino Mega serial ports

Port name	Transmit pin	Receive pin
Serial	1 (also USB)	0 (also USB)
Serial1	18	19
Serial2	16	17
Serial3	14	15

Software Serial:

- You will usually use the built-in Arduino Serial library to communicate with the hardware serial ports. Serial libraries simplify the use of the serial ports without having to worry about hardware complexities.

- Sometimes you need more serial ports than those available. If this is the case, you can use an additional library that uses software to emulate serial hardware. Recipes 4.13 and 4.14 show how to use a software serial library to communicate with multiple devices.

Serial Message Protocol:

- Meaningful serial communication, or any kind of machine-to-machine communication, can only be achieved if the sending and receiving sides fully agree how information is organized in the message.

- The formal organization of information in a message and the range of appropriate responses to requests is called a *communications protocol*.

- Messages can contain one or more special characters that identify the start of the message (called the *header)*.

Serial Message Protocol:

- One or more characters can also be used to identify the end of a message (called the *footer*).

- Sending and receiving messages in text format involves sending commands and numeric values as human-readable letters and words.

- Numbers are sent as the string of digits that represent the value. For example, if the value is 1234, the characters 1, 2, 3, and 4 are sent as individual characters.

New in Arduino 1.0:

- Serial.flush now waits for all outgoing data to be sent rather than discarding received data. You can use the following statement to discard all data in the receive buffer:
  while(Serial.read() >= 0) ; // flush the receive buffer

- Serial.write and Serial.print do not block. Earlier code would wait until all characters were sent before returning. From 1.0, characters sent using Serial.write are transmitted in the background (from an interrupt handler) allowing your sketch code to immediately resume processing. This is usually a good thing (it can make the sketch more responsive) but sometimes you want to wait until all characters are sent. You can achieve this by calling Serial.flush() immediately following Serial.write().

New in Arduino 1.0:

- Serial print functions return the number of characters printed. This is useful when text output needs to be aligned or for applications that send data which includes the total number of characters sent.

- There is a built-in parsing capability for streams such as Serial to easily extract numbers and find text.

- A Serial.peek function has been added to let you ‘peek’ at the next character in the receive buffer. Unlike Serial.read, the character is not removed from the buffer with Serial.peek.

Sending Debug Information from Arduino to Your Computer:

- **<u>Problem:</u>**
  - You want to send text and data to be displayed on your PC or Mac using the Arduino IDE or the serial terminal program of your choice.

- **<u>Solution:</u>**
  - This sketch prints sequential numbers on the Serial Monitor:

Sending Debug Information from Arduino to Your Computer:

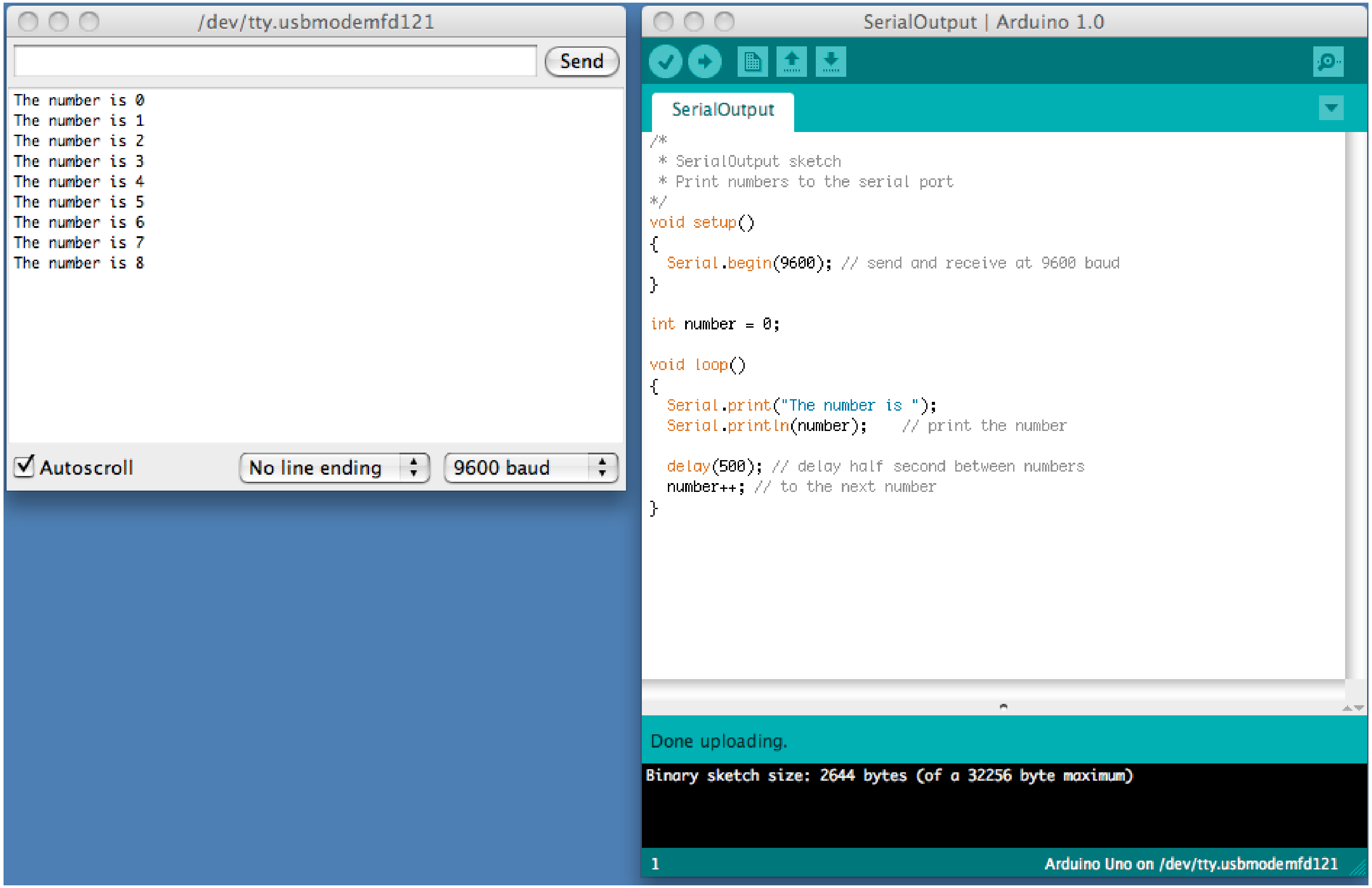
```
/*
 * SerialOutput sketch
 * Print numbers to the serial port
*/
void setup()
{
  Serial.begin(9600); // send and receive at 9600 baud
}

int number = 0;

void loop()
{
  Serial.print("The number is ");
   Serial.println(number);    // print the number

   delay(500); // delay half second between numbers
   number++; // to the next number
 }
```

Sending Debug Information from Arduino to Your Computer:



Sending Debug Information from Arduino to Your Computer:

- This example and most of the others in this book use a speed of 9,600 baud (*baud* is a measure of the number of bits transmitted per second).
- The 9,600 baud rate is approximately 1,000 characters per second.
- You can send at lower or higher rates (the range is 300 to 115,200), but make sure both sides use the same speed.
- The Serial Monitor sets the speed using the baud rate drop down (at the bottom right of the Serial Monitor window in Figure 4-2). If your output looks something like this:

you should check that the selected baud rate on the serial monitor on your computer matches the rate set by Serial.begin() in your sketch.

```
`3??f<ÌxÌ▯▯▯ü`³??f<
```

Sending Debug Information from Arduino to Your Computer:

> If your send and receive serial speeds are set correctly but you are still getting unreadable text, check that you have the correct board selected in the IDE Tools→Board menu. There are chip speed variants of some boards, if you have selected the wrong one, change it to the correct one and upload to the board again.

Sending Formatted Text and Numeric Data from Arduino:

- **<u>Problem:</u>**
  - You want to send serial data from Arduino displayed as text, decimal values, hexadecimal, or binary.

- **<u>Solution:</u>**
  - You can print data to the serial port in many different formats; here is a sketch that demonstrates all the format options:

Sending Formatted Text and Numeric Data from Arduino:

```
/*
 * SerialFormatting
 * Print values in various formats to the serial port
 */
char chrValue = 65;  // these are the starting values to print
byte byteValue = 65;
int intValue  = 65;
float floatValue = 65.0;

void setup()
{
  Serial.begin(9600);
}
```

Sending Formatted Text and Numeric Data from Arduino:

```
void loop()
{
  Serial.println("chrValue: ");
  Serial.println(chrValue);
  Serial.write(chrValue);
  Serial.println();
  Serial.println(chrValue,DEC);

  Serial.println("byteValue: ");
  Serial.println(byteValue);
  Serial.write(byteValue);
  Serial.println();
  Serial.println(byteValue,DEC);
```

```
  Serial.println("intValue: ");
  Serial.println(intValue);
  Serial.println(intValue,DEC);
  Serial.println(intValue,HEX);
  Serial.println(intValue,OCT);
  Serial.println(intValue,BIN);

  Serial.println("floatValue: ");
  Serial.println(floatValue);

  delay(1000); // delay a second between numbers
  chrValue++;  // to the next value
  byteValue++;
  intValue++;
  floatValue +=1;
}
```

Sending Formatted Text and Numeric Data from Arduino:

- The output (condensed here onto a few lines) is as follows:

Serial Monitor

```
chrValue:
A
A
65
byteValue:
65
A
65
intValue:
65
65
41
101
1000001
floatValue:
65.00
```

Here are some specific examples; all of them create variables that have similar values:

```
char asciiValue  = 'A';   // ASCII A has a value of 65
char chrValue    = 65;    // an 8 bit signed character, this also is ASCII 'A'
byte byteValue   = 65;    // an 8 bit unsigned character, this also is ASCII 'A'
int intValue     = 65;    // a 16 bit signed integer set to a value of 65
float floatValue = 65.0;  // float with a value of 65
```

Table 4-2. Output formats using Serial.print

Data type	print (val)	print (val,DEC)	write (val)	print (val,HEX)	print (val,OCT)	print (val,BIN)
char	A	65	A	41	101	1000001
byte	65	65	A	41	101	1000001
int	65	65	A	41	101	1000001
long	Format of long is the same as int					
float	65.00	Formatting not supported for floating-point values				
double	65.00	double is the same as float				

Sending Formatted Text and Numeric Data from Arduino:

The expression `Serial.print(val,BYTE);` is no longer supported in Arduino 1.0.

If your code expects byte variables to behave the same as char variables (that is, for them to print as ASCII), you will need to change this to `Serial.write(val);`.

Receiving Serial Data in Arduino:

- **<u>Problem:</u>**
  - You want to receive data on Arduino from a computer or another serial device; for example, to have Arduino react to commands or data sent from your computer.

- **<u>Solution:</u>**
  - It’s easy to receive 8-bit values (chars and bytes), because the Serial functions use 8-bit values. This sketch receives a digit (single characters 0 through 9) and blinks the LED on pin 13 at a rate proportional to the received digit value:

Receiving Serial Data in Arduino:

```
/*
 * SerialReceive sketch
 * Blink the LED at a rate proportional to the received digit value
*/
const int ledPin = 13; // pin the LED is connected to
int   blinkRate=0;     // blink rate stored in this variable

void setup()
{
  Serial.begin(9600); // Initialize serial port to send and receive at 9600 baud
  pinMode(ledPin, OUTPUT); // set this pin as output
}

void loop()
{
  if ( Serial.available()) // Check to see if at least one character is available
  {
    char ch = Serial.read();
    if( isDigit(ch) ) // is this an ascii digit between 0 and 9?
    {
      blinkRate = (ch - '0');      // ASCII value converted to numeric value
      blinkRate = blinkRate * 100; // actual rate is 100ms times received digit
    }
  }
  blink();
}
```

Receiving Serial Data in Arduino:

```
// blink the LED with the on and off times determined by blinkRate
void blink()
{
  digitalWrite(ledPin,HIGH);
  delay(blinkRate); // delay depends on blinkrate value
  digitalWrite(ledPin,LOW);
  delay(blinkRate);
}
```

Upload the sketch and send messages using the Serial Monitor. Open the Serial Monitor by clicking the Monitor icon (see Recipe 4.1) and type a digit in the text box at the top of the Serial Monitor window. Clicking the Send button will send the character typed into the text box; if you type a digit, you should see the blink rate change.

Receiving Serial Data in Arduino:

Converting the received ASCII characters to numeric values may not be obvious if you are not familiar with the way ASCII represents characters. The following converts the character `ch` to its numeric value:

```
blinkRate = (ch - '0');   // ASCII value converted to numeric value
```

The ASCII characters '0' through '9' have a value of 48 through 57 (see Appendix G). Converting '1' to the numeric value one is done by subtracting '0' because '1' has an ASCII value of 49, so 48 (ASCII '0') must be subtracted to convert this to the number one. For example, if `ch` is representing the character 1, its ASCII value is 49. The

expression `49- '0'` is the same as `49-48`. This equals 1, which is the numeric value of the character 1.

In other words, the expression `(ch - '0')` is the same as `(ch - 48)`; this converts the ASCII value of the variable `ch` to a numeric value.

Receiving numbers with more than one digit involves accumulating characters until a character that is not a valid digit is detected. The following code uses the same `setup()` and `blink()` functions as those shown earlier, but it gets digits until the newline character is received. It uses the accumulated value to set the blink rate.

Sending Multiple Text Fields from Arduino in a Single Message:

- **Problem:**
  - You want to send a message that contains more than one piece of information (field). For example, your message may contain values from two or more sensors. You want to use these values in a program such as Processing, running on your PC or Mac.

- **Solution:**
  - The easiest way to do this is to send a text string with all the fields separated by a delimiting (separating) character, such as a comma:

Sending Multiple Text Fields from Arduino in a Single Message:

```
// CommaDelimitedOutput sketch

void setup()
{
  Serial.begin(9600);
}

void loop()

{
  int value1 = 10;    // some hardcoded values to send
  int value2 = 100;
  int value3 = 1000;

  Serial.print('H'); // unique header to identify start of message
  Serial.print(",");
  Serial.print(value1,DEC);
  Serial.print(",");
  Serial.print(value2,DEC);
  Serial.print(",");
  Serial.print(value3,DEC);
  Serial.print(",");  // note that a comma is sent after the last field
  Serial.println();  // send a cr/lf
  delay(100);
}
```

Receiving Multiple Text Fields from Arduino in a Single Message:

- **Problem:**
  - You want to receive a message that contains more than one field. For example, your message may contain an identifier to indicate a particular device (such as a motor or other actuator) and what value (such as speed) to set it to.

- **Solution:**
  - Arduino does not have the split() function used in the Processing code in Recipe 4.4, but similar functionality can be implemented as shown in this recipe. The following code receives a message with three numeric fields separated by commas. It uses the technique described in Recipe 4.4 for receiving digits, and it adds code to identify comma-separated fields and store the values into an array:

Receiving Multiple Text Fields from Arduino in a Single Message:

```
/*
 * SerialReceiveMultipleFields sketch
 * This code expects a message in the format: 12,345,678
 * This code requires a newline character to indicate the end of the data
 * Set the serial monitor to send newline characters
 */

const int NUMBER_OF_FIELDS = 3; // how many comma separated fields we expect
int fieldIndex = 0;            // the current field being received
int values[NUMBER_OF_FIELDS];   // array holding values for all the fields
```

Receiving Multiple Text Fields from Arduino in a Single Message:

```
void setup()
{
  Serial.begin(9600); // Initialize serial port to send and receive at 9600 baud
}

void loop()
{
  if( Serial.available())
  {
    char ch = Serial.read();
    if(ch >= '0' && ch <= '9') // is this an ascii digit between 0 and 9?
    {
      // yes, accumulate the value if the fieldIndex is within range
      // additional fields are not stored
      if(fieldIndex < NUMBER_OF_FIELDS) {
```

Receiving Multiple Text Fields from Arduino in a Single Message:

```
        values[fieldIndex] = (values[fieldIndex] * 10) + (ch - '0');
      }
    }
    else if (ch == ',')  // comma is our separator, so move on to the next field
    {
        fieldIndex++;   // increment field index
    }
    else
    {
      // any character not a digit or comma ends the acquisition of fields
      // in this example it's the newline character sent by the Serial Monitor

      // print each of the stored fields
      for(int i=0; i < min(NUMBER_OF_FIELDS, fieldIndex+1); i++)
      {
        Serial.println(values[i]);
        values[i] = 0; // set the values to zero, ready for the next message
      }
      fieldIndex = 0;  // ready to start over
    }
  }
}
```

Sending Binary Data from Arduino:

- **Problem:**
  - You need to send data in binary format, because you want to pass information with the fewest number of bytes or because the application you are connecting to only handles binary data.
- **Solution:**
  - This sketch sends a header followed by two integer (16-bit) values as binary data. The values are generated using the Arduino random function:

Sending Binary Data from Arduino:

```
/*
 * SendBinary sketch
 * Sends a header followed by two random integer values as binary data.
*/

int intValue;    // an integer value (16 bits)

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.print('H'); // send a header character

  // send a random integer
```

Sending Binary Data from Arduino:

```
  intValue = random(599); // generate a random number between 0 and 599
  // send the two bytes that comprise an integer
  Serial.write(lowByte(intValue));  // send the low byte
  Serial.write(highByte(intValue)); // send the high byte

  // send another random integer
  intValue = random(599); // generate a random number between 0 and 599
  // send the two bytes that comprise an integer
  Serial.write(lowByte(intValue));  // send the low byte
  Serial.write(highByte(intValue)); // send the high byte

  delay(1000);
}
```

Sending the Value of Multiple Arduino Pins:

- **Problem:**
  - You want to send groups of binary bytes, integers, or long values from Arduino. For example, you may want to send the values of the digital and analog pins to Processing.
- **Solution:**
  - This recipe sends a header followed by an integer containing the bit values of digital pins 2 to 13. This is followed by six integers containing the values of analog pins 0 through 5. Chapter 5 has many recipes that set values on the analog and digital pins that you can use to test this sketch:

Sending the Value of Multiple Arduino Pins:

```
/*
 * SendBinaryFields
 * Sends digital and analog pin values as binary data
 */

const char HEADER = 'H';  // a single character header to indicate
                          // the start of a message

void setup()
{
  Serial.begin(9600);
  for(int i=2; i <= 13; i++)
  {
```

Sending the Value of Multiple Arduino Pins:

```
    pinMode(i, INPUT);        // set pins 2 through 13 to inputs
    digitalWrite(i, HIGH);    // turn on pull-ups
  }
}

void loop()
{
  Serial.write(HEADER); // send the header
  // put the bit values of the pins into an integer
  int values = 0;
  int bit = 0;

  for(int i=2; i <= 13; i++)
  {
    bitWrite(values, bit, digitalRead(i));  // set the bit to 0 or 1 depending
                                            // on value of the given pin
    bit = bit + 1;                          // increment to the next bit
  }
  sendBinary(values); // send the integer
```

Sending the Value of Multiple Arduino Pins:

```
  for(int i=0; i < 6; i++)
  {
    values = analogRead(i);
    sendBinary(values); // send the integer
  }
  delay(1000); //send every second
}

// function to send the given integer value to the serial port
void sendBinary( int value)
{
  // send the two bytes that comprise an integer
  Serial.write(lowByte(value));  // send the low byte
  Serial.write(highByte(value)); // send the high byte
}
```

Sending the Value of Multiple Arduino Pins:

Discussion

The code sends a header (the character H), followed by an integer holding the digital pin values using the bitRead function to set a single bit in the integer to correspond to the value of the pin (see Chapter 3). It then sends six integers containing the values read from the six analog ports (see Chapter 5 for more information). All the integer values are sent using sendBinary, introduced in Recipe 4.6. The message is 15 bytes long—1 byte for the header, 2 bytes for the digital pin values, and 12 bytes for the six analog integers. The code for the digital and analog inputs is explained in Chapter 5.

Assuming analog pins have values of 0 on pin 0, 100 on pin 1, and 200 on pin 2 through 500 on pin 5, and digital pins 2 through 7 are high and 8 through 13 are low, this is the decimal value of each byte that gets sent:

```
72   // the character 'H' - this is the header
     // two bytes in low high order containing bits representing pins 2-13
63   // binary 00111111 :  this indicates that pins 2-7 are high
```

Sending the Value of Multiple Arduino Pins:

```
0    // this indicates that  8-13 are low

     // two bytes for each pin representing the analog value
0    // pin 0 has an integer value of 0 so this is sent as two bytes
0

100  // pin 1 has a value of 100, sent as a byte of 100 and a byte of 0
0
...
     // pin 5 has a value of 500
244  // the remainder when dividing 500 by 256
1    //  the number of times 500 can be divided by 256
```

How to Move the Mouse Cursor on a PC or Mac:

- **Problem:**
  - You want Arduino to interact with an application on your computer by moving the mouse cursor. Perhaps you want to move the mouse position in response to Arduino information. For example, suppose you have connected a Wii nunchuck to your Arduino and you want your hand movements to control the position of the mouse cursor in a program running on a PC.

- **ution:**
  - You can send serial commands that specify the mouse cursor position to a program running on the target computer. Here is a sketch that moves the mouse cursor based on the position of two potentiometers:

How to Move the Mouse Cursor on a PC or Mac:

```
// SerialMouse sketch
const int buttonPin = 2;  //LOW on digital pin enables mouse

const int potXPin = 4;    // analog pins for pots
const int potYPin = 5;

void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
  digitalWrite(buttonPin, HIGH); // turn on pull-ups
}
```

How to Move the Mouse Cursor on a PC or Mac:

```
void loop()
{
  int x = (512 - analogRead(potXPin)) / 4;  // range is -127 to +127
  int y = (512 - analogRead(potYPin)) / 4;
  Serial.print("Data,");
  Serial.print(x,DEC);
  Serial.print(",");
  Serial.print(y,DEC);
  Serial.print(",");
  if(digitalRead(buttonPin) == LOW)
    Serial.print(1);  // send 1 when button pressed
  else
    Serial.print(0);
  Serial.println(",");
  delay(50); // send position 20 times a second
}
```

How to Move the Mouse Cursor on a PC or Mac:

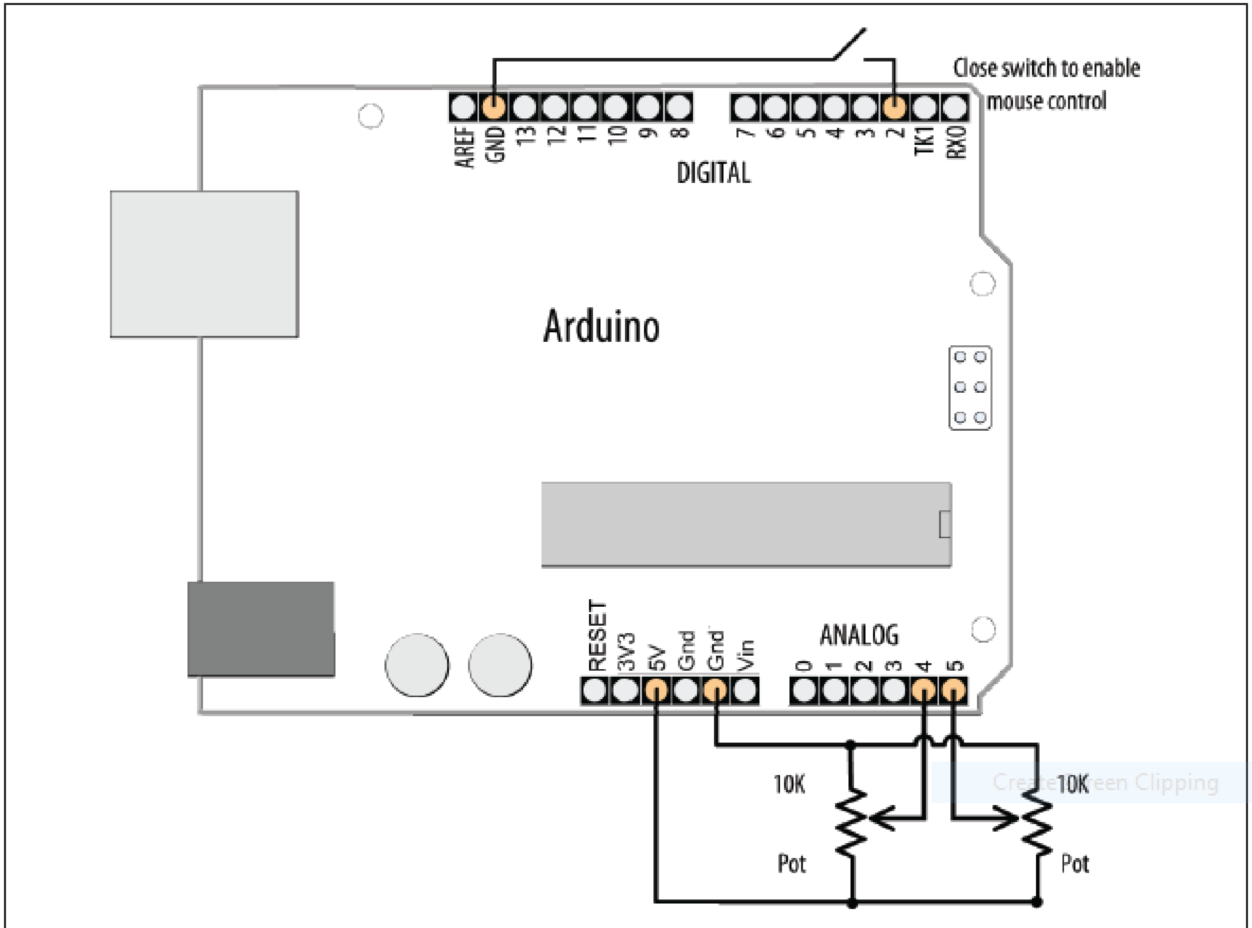


Figure 4-4. Wiring for mouse control using two potentiometers

Sending Data to Two Serial Devices at the Same Time:

- **Problem:**
  - You want to send data to a serial device such as a serial LCD, but you are already using the built-in serial port to communicate with your computer.

- **Solution:**
  - On a Mega this is not a problem, as it has four hardware serial ports; just create two serial objects and use one for the LCD and one for the computer:

Sending Data to Two Serial Devices at the Same Time:

```
void setup() {
  // initialize two serial ports on a Mega
  Serial.begin(9600);  // primary serial port
  Serial1.begin(9600); // Mega can also use Serial1 through Serial3
}
```

- On a standard Arduino board (such as the Uno or Duemilanove) that only has one hardware serial port, you will need to create an emulated or “soft” serial port.
- You can use the distributed SoftwareSerial library for sending data to multiple devices.
- you can download NewSoftSerial from http://arduiniana.org/libraries/newsoftserial

Sending Data to Two Serial Devices at the Same Time:

Select two available digital pins, one each for transmit and receive, and connect your serial device to them. It is convenient to use the hardware serial port for communication with the computer because this has a USB adapter on the board. Connect the device's transmit line to the receive pin and the receive line to the transmit pin. In Figure 4-6, we have selected pin 2 as the receive pin and pin 3 as the transmit pin.

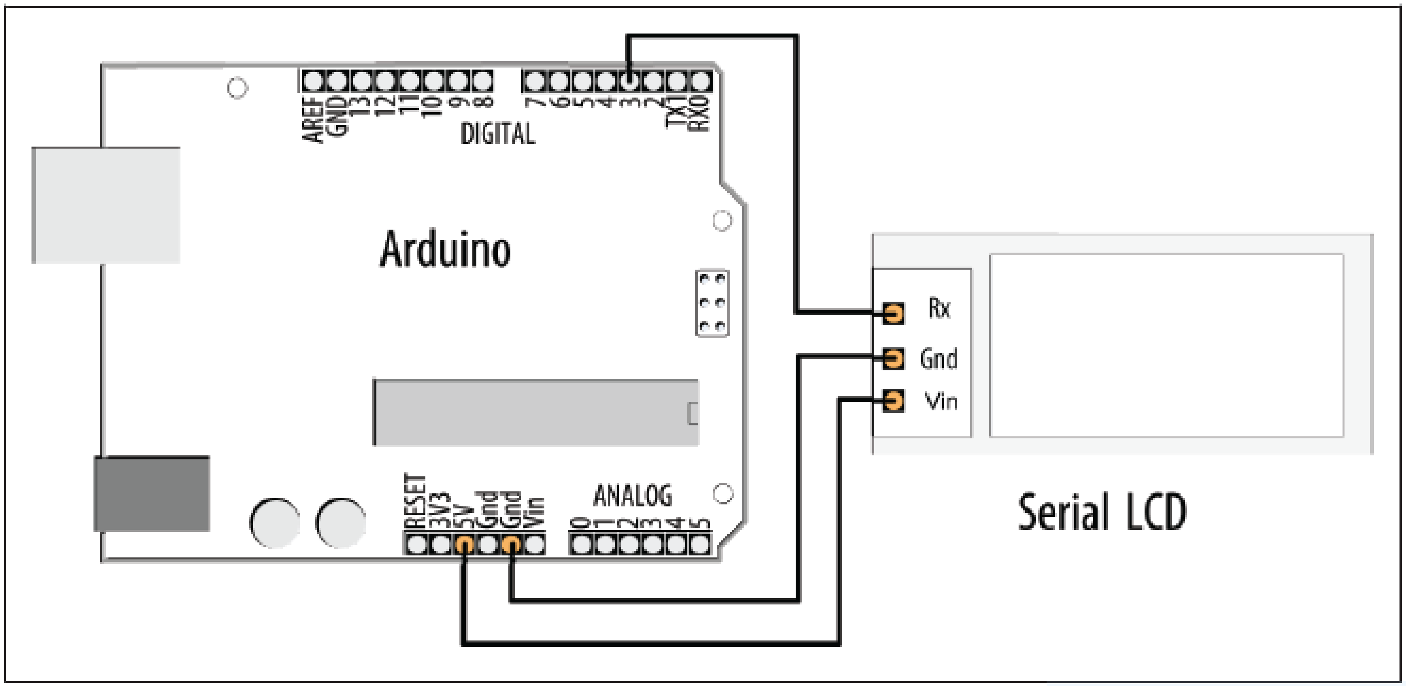


Figure 4-6. Connecting a serial device to a "soft" serial port

Sending Data to Two Serial Devices at the Same Time:

In your sketch, create a SoftwareSerial object and tell it which pins you chose as your emulated serial port. In this example, we're creating an object named serial_lcd, which we instruct to use pins 2 and 3:

```
/*
 * SoftwareSerialOutput sketch
 * Output data to a software serial port
 */

#include <SoftwareSerial.h>

const int rxpin = 2;           // pin used to receive (not used in this version)
const int txpin = 3;           // pin used to send to LCD
SoftwareSerial serial_lcd(rxpin, txpin); // new serial port on pins 2 and 3

void setup()
{
  Serial.begin(9600); // 9600 baud for the built-in serial port
  serial_lcd.begin(9600); //initialize the software serial port also for 9600
}
```

Sending Data to Two Serial Devices at the Same Time:

```
int number = 0;

void loop()
{
  serial_lcd.print("The number is ");  // send text to the LCD
  serial_lcd.println(number);    // print the number on the LCD
  Serial.print("The number is ");
  Serial.println(number);        // print the number on the PC console

  delay(500); // delay half second between numbers
  number++;   // to the next number
}
```

Sending Data to Two Serial Devices at the Same Time:

If you are using Arduino versions prior to 1.0, download the NewSoftSerial library and replace references to SoftwareSerial with NewSoftSerial:

```
// NewSoftSerial version

#include <NewSoftSerial.h>

const int rxpin = 2;           // pin used to receive from LCD
const int txpin = 3;           // pin used to send to LCD
NewSoftSerial serial_lcd(rxpin, txpin); // new serial port on pins 2 + 3
```

Sending Data to Two Serial Devices at the Same Time:

Discussion

Every Arduino microcontroller contains at least one built-in serial port. This special piece of hardware is responsible for generating the series of precisely timed pulses its partner device sees as data and for interpreting the similar stream that it receives in return. Although the Mega has four such ports, most Arduino flavors have only one. For projects that require connections to two or more serial devices, you'll need a software library that emulates the additional ports. A "software serial" library effectively turns an arbitrary pair of digital I/O pins into a new serial port.

To build your software serial port, you select a pair of pins that will act as the port's transmit and receive lines in much the same way that pins 1 and 0 are controlled by Arduino's built-in port. In Figure 4-6, pins 3 and 2 are shown, but any available digital pins can be used. It's wise to avoid using 0 and 1, because these are already being driven by the built-in port.

Sending Data to Two Serial Devices at the Same Time:

The syntax for writing to the soft port is identical to that for the hardware port. In the example sketch, data is sent to both the "real" and emulated ports using print() and println():

```
serial_lcd.print("The number is ");  // send text to the LCD
serial_lcd.println(number);          // send the number on the LCD

Serial.print("The number is ");      // send text to the hardware port
Serial.println(number);              // to output on Arduino Serial Monitor
```

Sending Data to Two Serial Devices at the Same Time:

If you are using a *unidirectional* serial device—that is, one that only sends or receives—you can conserve resources by specifying a nonexistent pin number in the SoftwareSerial constructor for the line you don't need. For example, a serial LCD is fundamentally an output-only device. If you don't expect (or want) to receive data from it, you can tell SoftwareSerial using this syntax:

```
#include <SoftwareSerial.h>
...
const int no_such_pin = 255;
const int txpin = 3;
SoftwareSerial serial_lcd(no_such_pin, txpin); // TX-only on pin 3
```

In this case, we would only physically connect a single pin (3) to the serial LCD's "input" or "RX" line.

Receiving Serial Data from Two Devices at the Same Time:

- **Problem**
  - ➢ You want to receive data from a serial device such as a serial GPS, but you are already using the built-in serial port to communicate with your computer.
- **Solution**
  - ➢ This problem is similar to the one in Recipe 4.13, and indeed the solution is much the same. If your Arduino’s serial port is connected to the console and you want to attach a second serial device, you must create an emulated port using a software serial library.In this case, we will be receiving data from the emulated port instead of writing to it, but the basic solution is very similar.

Receiving Serial Data from Two Devices at the Same Time:

Select two pins to use as your transmit and receive lines.

Connect your GPS as shown in Figure 4-7. Rx (receive) is not used in this example, so you can ignore the Rx connection to pin 3 if your GPS does not have a receive pin.

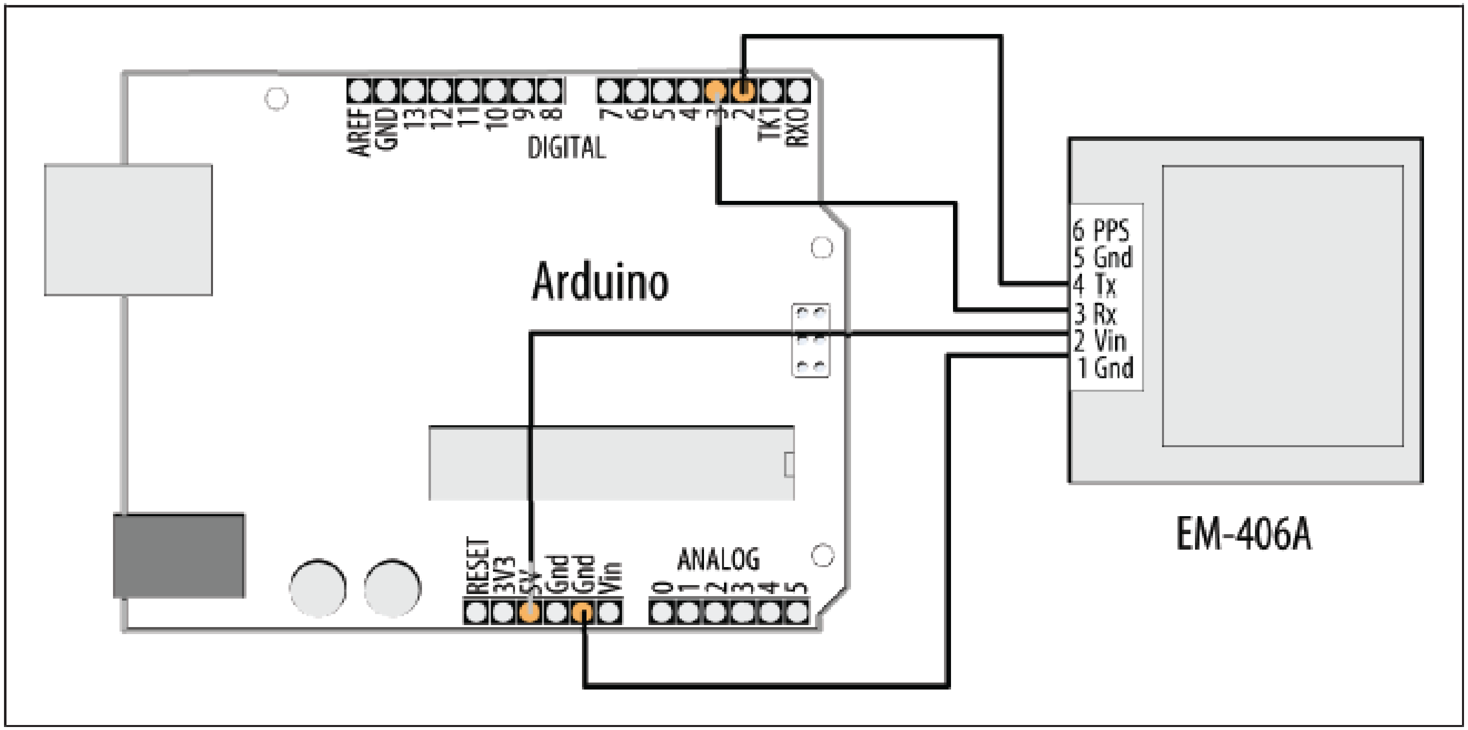


Figure 4-7. Connecting a serial GPS device to a "soft" serial port

Receiving Serial Data from Two Devices at the Same Time:

As you did in Recipe 4.13, create a SoftwareSerial object in your sketch and tell it which pins to control. In the following example, we define a soft serial port called serial_gps, using pins 2 and 3 for receive and transmit, respectively:

```
/*
 * SoftwareSerialInput sketch
 * Read data from a software serial port
 */

#include <SoftwareSerial.h>
const int rxpin = 2;                    // pin used to receive from GPS
const int txpin = 3;                    // pin used to send to GPS
SoftwareSerial serial_gps(rxpin, txpin); // new serial port on pins 2 and 3

void setup()
{
  Serial.begin(9600); // 9600 baud for the built-in serial port
  serial_gps.begin(4800); // initialize the port, most GPS devices
                          // use 4800 baud
}
```

Receiving Serial Data from Two Devices at the Same Time:

```
void loop()
{
  if (serial_gps.available() > 0) // any character arrived yet?
  {
    char c = serial_gps.read();   // if so, read it from the GPS
    Serial.write(c);              // and echo it to the serial console

  }
}
```

Receiving Serial Data from Two Devices at the Same Time:

Discussion

You initialize an emulated SoftwareSerial port by providing pin numbers for transmit and receive. The following code will set up the port to receive on pin 2 and send on pin 3:

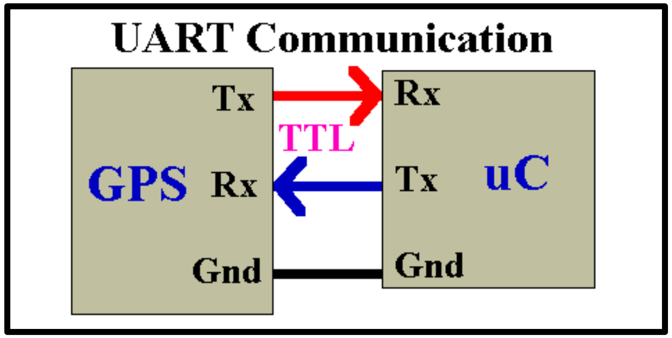
```
const int rxpin = 2;                     // pin used to receive from GPS
const int txpin = 3;                     // pin used to send to GPS
SoftwareSerial serial_gps(rxpin, txpin); // new serial port on pins 2 and 3
```

The `txpin` is not used in this example and can be set to 255 to free up pin 3, as explained in the previous recipe.

The syntax for reading an emulated port is very similar to that for reading from a built-in port. First check to make sure a character has arrived from the GPS with `available()`, and then read it with `read()`.

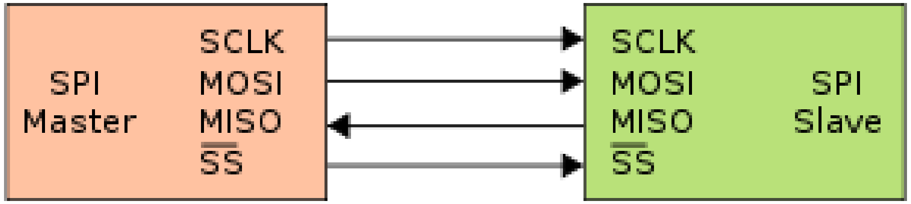
Universal Asynchronous Receiver/Transmitter (UART)

- A UART (Universal Asynchronous Receiver/Transmitter) is the microchip with programming that controls a computer's interface to its attached serial devices. Specifically, it provides the computer with the RS-232C Data Terminal Equipment (DTE) interface so that it can "talk" to and exchange data with modems and other serial devices.
- Serial communication on pins TX/RX uses TTL logic levels (5V or 3.3V depending on the board). Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board.

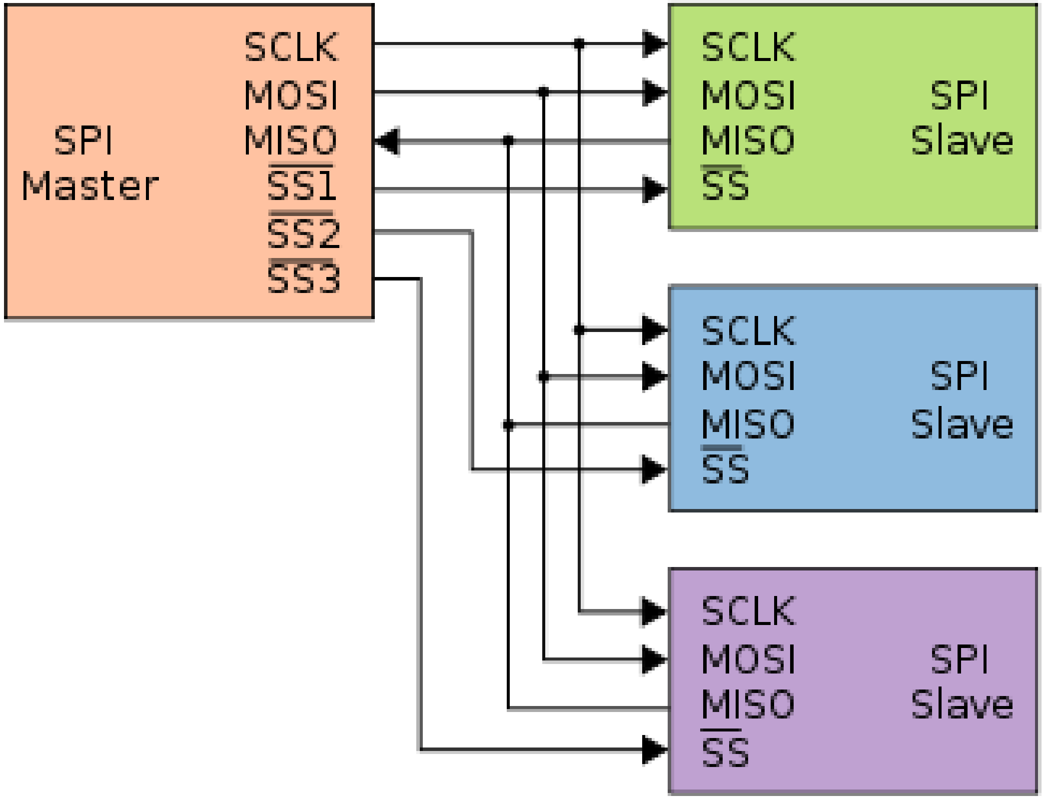


Serial peripheral interface (SPI)

- The Serial Peripheral Interface (SPI) bus is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems. The interface was developed by Motorola in the late eighties and has become a de facto standard. Typical applications include Secure Digital cards and liquid crystal displays.
- SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines.



Serial peripheral interface (SPI)



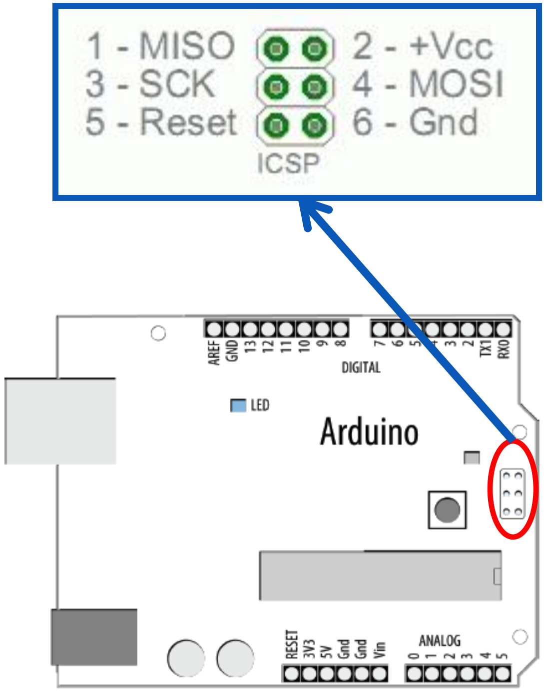
Serial peripheral interface (SPI)

Connections

The following table display on which pins the SPI lines are broken out on the different Arduino boards:

Arduino / Genuino Board	MOSI	MISO	SCK	SS (slave)	SS (master)	Level
Uno or Duemilanove	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10	-	5V
Mega1280 or Mega2560	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	53	-	5V
Leonardo	ICSP-4	ICSP-1	ICSP-3	-	-	5V
Due	ICSP-4	ICSP-1	ICSP-3	-	4, 10, 52	3,3V
Zero	ICSP-4	ICSP-1	ICSP-3	-	-	3,3V
101	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10	10	3,3V
MKR1000	8	10	9	-	-	3,3V

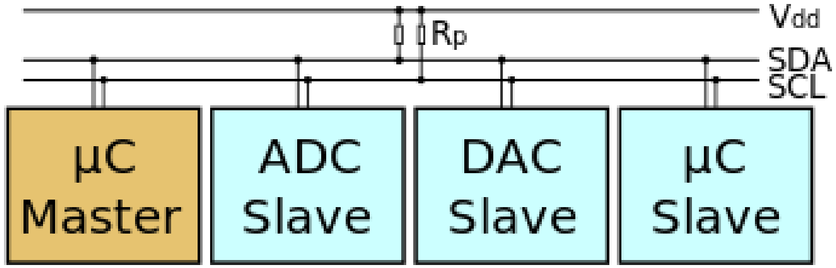
Note that MISO, MOSI, and SCK are available in a consistent physical location on the ICSP header; this is useful, for example, in designing a shield that works on every board.



I²C

- I²C (Inter-Integrated Circuit), pronounced I-squared-C, is a multi-master, multi-slave, single-ended, serial computer bus invented by Philips Semiconductor (now NXP Semiconductors). It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication. Alternatively I²C is spelled I2C (pronounced I-two-C) or IIC (pronounced I-I-C).

- I²C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock Line (SCL), pulled up with resistors. Typical voltages used are +5 V or +3.3 V although systems with other voltages are permitted.



I^2C Arduino

Board	I2C / TWI pins
Uno, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

Note

There are both 7- and 8-bit versions of I2C addresses. 7 bits identify the device, and the eighth bit determines if it's being written to or read from. The Wire library uses 7 bit addresses throughout. If you have a datasheet or sample code that uses 8 bit address, you'll want to drop the low bit (i.e. shift the value one bit to the right), yielding an address between 0 and 127. However the addresses from 0 to 7 are not used because are reserved so the first address that can be used is 8.