Practical methods for connecting physical objects.

3rd Edition

Making Things Talk

Learn by Discovery

USING SENSORS, NETWORKS, AND ARDUINO TO SEE, HEAR, AND FEEL YOUR WORLD

Tom Igoe

MAKER MEDIA

Make:
makezine.com

Project 4

Making Your Own Arduino-Compatible Board

There are times when you want to use more than one microcontroller in a project, or when you just need a controller with one or two I/O pins and don't want to spend the money on a full-featured microcontroller. There may also be times when it's simpler to have multiple processors each dedicated to one task. This is when Atmel's ATtiny microcontrollers come in handy. They're small, they cost only a few dollars, and you can program them using another Arduino-compatible microcontroller, using the SPI synchronous serial protocol.

MATERIALS

- **1 Arduino-compatible microcontroller**
  - **(Uno or MKR1000 shown)**
  - **Features used: UART, SPI**
- **1 ATtiny84 microcontroller**
- **1 solderless breadboard**
- **Jumper wires**
- **1 RGB LED, common cathode**
- **1 220-ohm resistor**

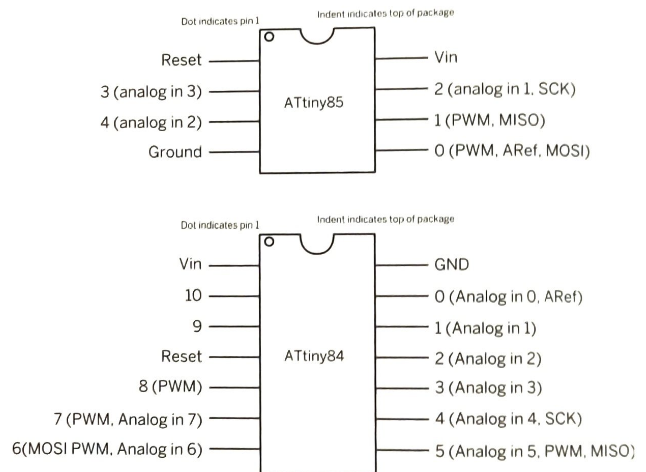


Figure 2-18
ATtiny84 and 85-pin diagrams.

Microcontroller boards like the MKR1000, the 101, and even the Uno have a lot more parts on them besides the microcontroller itself. Most have a reset button and a *clock crystal* or *resonator* to supply the processor with a precise timing pulse. Many have a *voltage regulator* that can take a variable input voltage and supply a constant output voltage for the controller. This allows you to plug in a variety of power sources to the DC power jack, for example. Some have a USB-to-serial adapter, as explained earlier. Others have additional sensors or radios on board, like the MKR1000 and the 101.

Without all these features, the microcontroller alone can be quite inexpensive. Unfortunately, most microcontrollers these days are too small to work with on a breadboard, but it is still possible to get some of Atmel's AVR 8-bit microcontollers in a *dual inline package (DIP)* that can fit into a breadboard. The ATmega328P, which is the controller at the heart of the Uno, is one of these. Two others, the ATtiny84 and ATtiny85, are perhaps the most useful for the size and cost. Their pin diagrams are shown in Figure 2-18. They each have a few GPIO pins that can be used as analog or digital input, and a few PWM pins as well. They can't do everything the others can, but they can do the basics of digital input and output, analog input, and pseudo-analog output using PWM. They don't have a UART on board, but they can manage asynchronous serial communication using the SoftwareSerial library. When combined with a more fully featured processor, they are quite useful. This project shows you how you can reproduce the LED lamp from the first project on a controller that costs less than four dollars.

Arduino as ISP

When you get an Uno or any Uno-compatible board, the microcontroller already has a program on it called a *bootloader*. The bootloader is a tiny program that stays on the

Introducing Serial Peripheral Interface (SPI)

In-crcuit serial programming uses a form of synchronous serial communication called Serial Peripheral Interface, or SPI. SPI, along with another synchronous serial protocol, Inter-Integrated Circuit or I2C (sometimes called Two-Wire Interface, or TWI), are two of the most common synchronous serial protocols you'll encounter. You'll see SPI used for other devices, like the WiFi radio on the MKR1000, or for communication with SD memory cards, as well as many sensors.

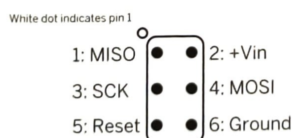
Synchronous serial protocols all feature a controlling device that generates a regular pulse, or clock signal, on one pin while exchanging data on every clock pulse. The advantage of a synchronous serial protocol is that it's a bus: you can have several devices sharing the same physical connections to one master controller.

SPI connections have three or four connections between the controlling device (or master device) and the peripheral device (or slave), as follows:

- **Clock (SCK)**: The pin that the master pulses regularly.
- **Master Out, Slave In (MOSI)**: The master device sends a bit of data to the slave on this line every clock pulse.
- **Master In, Slave Out (MISO)**: The slave device sends a bit of data to the master on this line every clock pulse.
- **Slave Select (SS)** or **Chip Select (CS)**: Because several slave devices can share the same bus, each has a unique connection to the master. The master sets this pin low to address this particular slave device. If the master's not talking to a given slave, it will set the slave's select pin high.

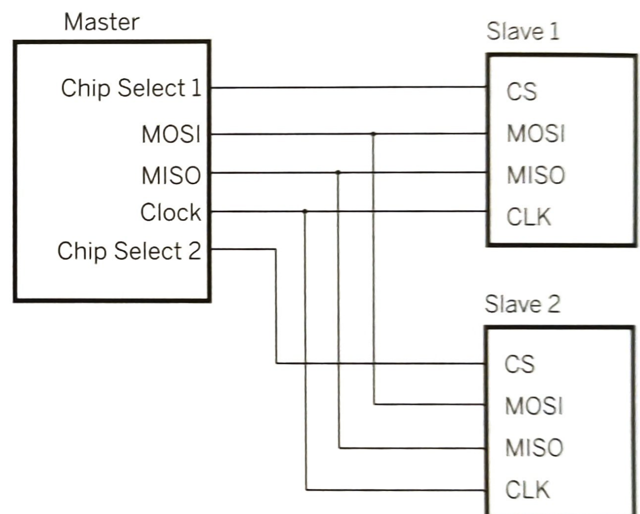
If the slave doesn't need to send any data to the master, there will be no MISO pin.

Since SPI is the standard method for programming AVR controllers, most boards using those controllers, including the Uno and earlier Arduino models, all have an ICSP header that breaks out the SPI pins. It looks like this:



If there's an ICSP header on your board, you can count on the pins having this arrangement. Different microcontrollers break the SPI functions to different pins, however. Here are the SPI pin configurations for the Uno, 101, and MKR1000:

Function	Uno	101	MKR1000
MOSI	11 or ICSP4	ICSP4	8
MISO	12 or ICSP1	ICSP1	10
Clock	13 or ICSP3	ICSP3	9
Chip Select	10	10	user choice

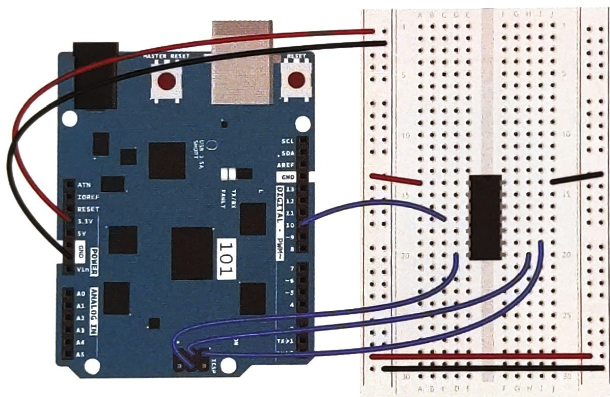


microcontroller at all times. Its only job is to read bytes in through the UART. If those bytes are formatted as a new program, the bootloader writes them to the rest of the controller's program memory. This is why you can upload a new program to your Uno via the USB serial port.

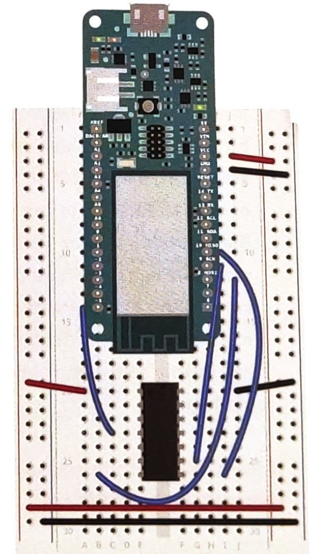
Microcontrollers don't normally come with a bootloader on board. Instead, they are programmed with a separate piece of hardware called an *in-circuit serial programmer (ICSP or ISP)*, using the SPI synchronous serial protocol. The programmer sends a clock signal to the microcontroller, and every time the clock pulses, the microcontroller reads another bit of data and writes it to program memory. When the whole program is written to memory and the controller is reset, the program runs. Programmers like Atmel's AVRISP mkII and Adafruit's USBTinyISP are popular AVR programmers. Different microcontroller architectures usually require different programmers.

There's a sketch included with the Arduino IDE examples that will turn most any board into an ICSP programmer. Click on File→Examples→ArduinoISP, and upload it to the board that you want to use as a programmer. Then connect an ATtiny84 to the board as shown in Figure 2-19.

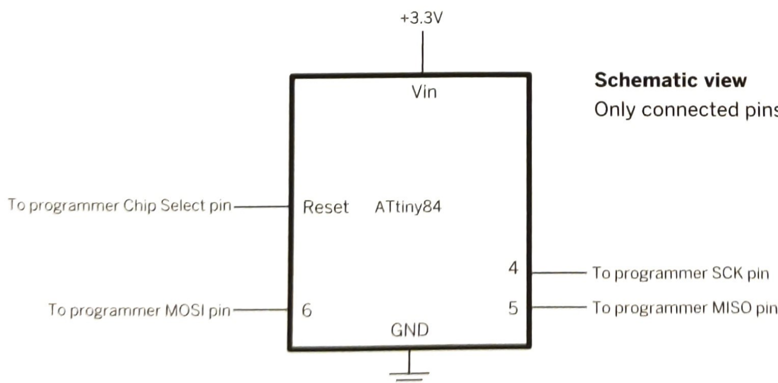
Before you can program the ATtiny84, you'll need to add its board definition to the IDE. As you did in Chapter 1 (see the "Boards Manager" sidebar), click on Preferences and look for the Additional Boards Manager URLs box. Enter the following URL in the box: http://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json. Then click OK and restart the IDE. When you restart, you'll find the ATtiny boards in the Tools Boards menu. For other processors in the AVR line, the process is the same. For example, the ATmega328P uses the Uno board definition. The ATmega2560 uses the Mega board definition.



101/Uno Breadboard view



MKR1000 Breadboard view



Schematic view
Only connected pins are shown.

Figure 2-19
Programming an ATtiny84 from an Arduino-compatible board. The half-circle at the top of the ATtiny indicates the top of the component, and the indent indicates physical pin number 1. This is standard on *dual-inline package (DIP)* components.

In order to program the ATtiny, you need to set the microcontroller's basic configurations. These are stored in 3 bytes of permanent memory and are called *fuses*. The fuses configure the controller's clock speed, running voltage, and so forth. In the Boards menu, set the ATtiny85 as follows:

- Board: ATtiny
- Processor: ATtiny84
- Clock: 8MHz (internal)
- Port: whatever port your programmer board is using

Click on the Tools Programmer menu and choose "Arduino as ISP." This tells the IDE to use your programmer board as the programmer for the ATtiny.

Next choose Burn Bootloader from the Boards menu. This will set the fuses. When it's done, you'll see the message "Done Burning Bootloader" in the console pane. Now you're ready to put a sketch on the ATtiny84.

You can see that the clock is set to 8MHz (internal). This means that the processor will use an internal circuit to keep its timing. With the ATtiny set like this, you need only a connection to voltage (between +3 and +5V) and ground and a pullup resistor to voltage on the reset pin to make it work. Try uploading the Blink sketch to it. First change the LED pin number from 13 to 6, then click Upload Using Programmer. The IDE will use your programmer board as the in-circuit serial programmer to program the ATtiny. Add an LED to digital I/O pin 6 and you should see the LED blinking when the ATtiny microcontroller is powered.

SoftwareSerial and ATtiny Type Brighter

The ATtiny has all the capabilities you need to run either of the projects in this chapter, so let's re-create the first, Type Brighter.

Connect the RGB LED to digital I/O pins 6 through 8 of the ATtiny84. These pins can be used for PWM, just like the PWM pins of the 101 and MKR1000 as you saw earlier. Load the Serial RGB LED controller sketch from the first project, and make the following changes:

Try It Add the following two lines to the beginning of the sketch. These lines include and make an instance of the SoftwareSerial library, which allows you to send and receive serial data on any two GPIO pins. You'll use pins 0 and 1.

Change the LED pin numbers as shown here as well.

```
#include <SoftwareSerial.h>  // include the SoftwareSerial library

SoftwareSerial swSerial(0, 1); // RX, TX

// constants to hold the output pin numbers:
const int redPin = 8;
const int greenPin = 7;
const int bluePin = 6;
```

Once you've added these lines, change every instance of `Serial` in your current Arduino sketch to `swSerial`. The SoftwareSerial library has the same `print()`, `println()`, `read()`, `write()`, and `available()` commands as the main Serial library.

Upload the changed sketch to your ATtiny84 using your programmer board just like you did with the Blink sketch. Then disconnect the ATtiny84 from the breadboard, and connect it to a USB-to-serial adapter as shown in Figure 2-20. The adapter's TX goes to pin 0 and RX goes to pin 1. In this case, you'll power the ATtiny84 from the adapter, so connect Vin and ground to the microcontroller's Vin and ground as well. Open CoolTerm or the Serial Monitor to your adapter and type commands just like you did in the first project, for example:

```
r5g3b7
```

The LED should change just like it did in the first project. It's the same project, but much simpler and less expensive!

X

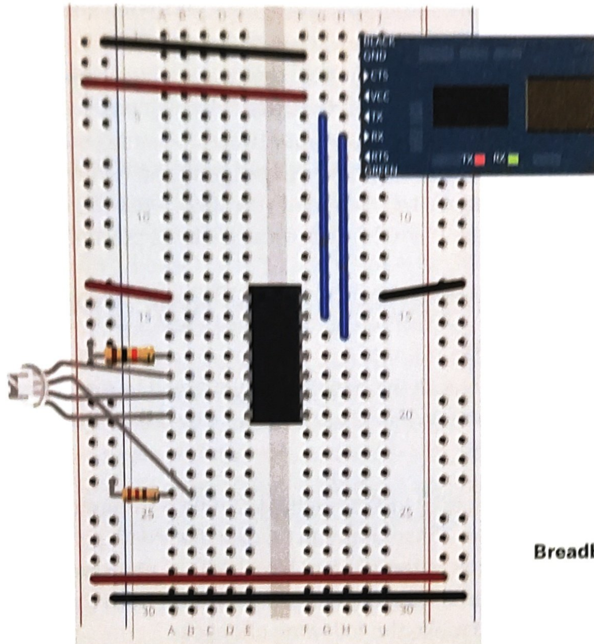
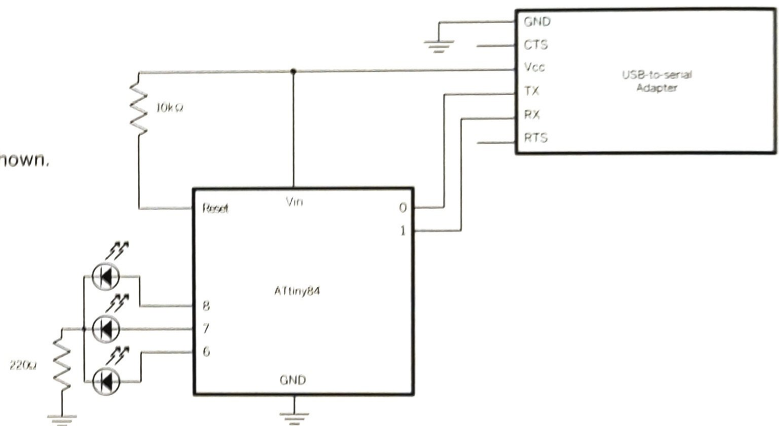


Figure 2-20
Type Brighter project on an ATtiny84. The USB-to-serial converter is supplying power for the circuit via USB from the personal computer. The ATtiny84 and the LED can operate at +3.3V or +5V, so it doesn't matter which the adapter is supplying. The 10-kilohm pullup resistor on the reset pin keeps the microcontroller from resetting spontaneously, by keeping the reset pin high. To reset the micrcontroller, connect the reset pin to ground.

Breadboard view

Schematic view
Only connected pins are shown.



More Serial Ports: Software Serial

A microcontroller's Universal Asynchronous Receiver-Transmitter (or UART, labeled RX and TX) allows it to send and receive serial data reliably, no matter what your code is doing, because the UART listens for serial all the time. What do you do when you need to attach more than one asynchronous serial device to your Arduino? Some microcontrollers have multiple UARTs. The Arduino Mega 2560 has four, for example. But you may not need all that a Mega has to offer just to get another serial port. SoftwareSerial allows you to use two digital pins as a "fake" UART. The library can listen for incoming serial on those pins, and transmit as well. Because it's not a dedicated hardware UART, SoftwareSerial isn't as reliable as hardware serial at very high or very low speeds, though it does well from 4800bps through 57.6kbps. When you're using limited controllers like the ATtiny processors, it's handy. It can also be useful on the Uno and other processors that have only one UART, when you want to connect to another asynchronous serial device and still want to use the main UART for communication with your personal computer.

X

> Although many projects work fine with just one microcontroller, there are cases where it's helpful to have two or more with different capabilities. For example, the ESP8266 controller, which you'll see in later chapters, is great at network communications via WiFi, but it's only got one analog input, and that input has only a 1-volt range. For a project that needs multiple analog inputs, like Monski Pong, the ESP8266 wouldn't be much good on its own. But it could be connected serially to an ATtiny84 or ATtiny85 to handle the analog input.

Being able to expand the number of inputs and outputs isn't the only reason you might want two controllers. You might have a project in which you need very precise timing of an output device at the same time as you need to make a complex network exchange. By separating the tasks to two processors, you can simplify the programming by splitting it into two programs.

The Arduino IDE can program a number of controllers in the AVR microcontroller family using in-circuit serial programming, simply by selecting the right board from the Boards menu and burning the bootloader. Here's what it can do with the built-in board settings:

- ATmega328P - bootload using Uno board setting
- ATMega168 - bootload using Diecimila or Duemilanove
- ATMega8 - bootload using NG or older
- ATMega2560 - bootload using Mega2560
- ATmega32U4 - bootload using Micro

Using the board definitions from this project, it can also program these:

- ATtiny84
- ATtiny44
- ATtiny 85
- ATtiny45

Of these, the ATtinys and the ATmega328P are probably the most useful. The former are small and very inexpensive, and the latter is the same processor used in the Uno, so it's compatible with many existing examples on the web.

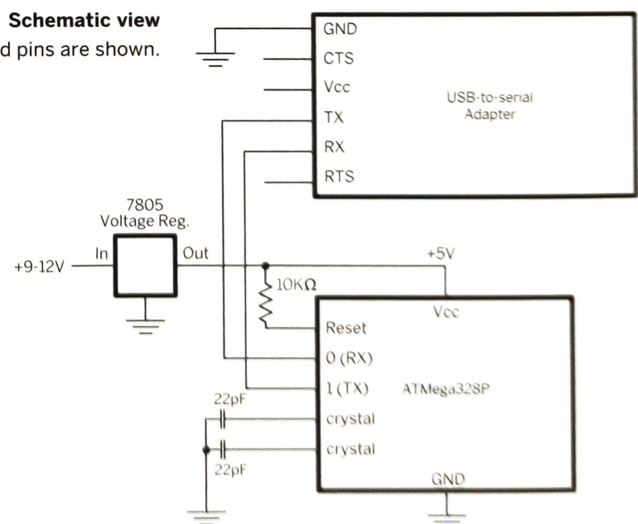
Figure 2-21 shows the circuit and components necessary to make your own Uno-compatble breadboard circuit. It's bootloaded just like the ATtiny controllers: connect the SPI pins to your programmer board, choose "Uno" as the board type, then burn the bootloader.

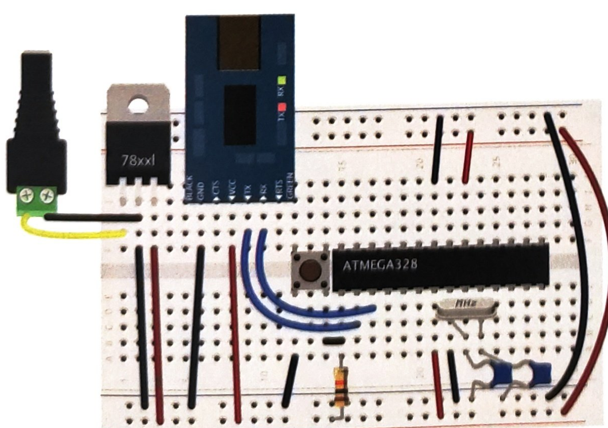
Once you've put the bootloader on the controller, you can program it over the USB-to-serial converter, but you'll need to hit the reset button right before upload each time.

It also includes a 5-volt voltage regulator and DC power jack so that you can power it from a 9–12V DC power source. The full pin diagram of the ATMega328P can be found at www.arduino.cc/en/Hacking/PinMapping168.

X

Figure 2-21
ATmega328 on a breadboard with a USB-to-serial adapter.





Breadboard view

Conclusion

The projects in this chapter have covered a number of ideas that are central to all networked data communication. First, remember that data communication is based on a layered series of agreements, starting with the physical layer; then the electrical, the logical, the data layers; and finally, the application layer. Keep these layers in mind as you design and troubleshoot your projects, and you'll find it's easier to isolate problems.

Second, remember that serial data can be sent either as ASCII-encoded or as raw binary values, and which you choose to use depends on both the capabilities and limitations of all the connected devices. It might not be wise to send raw binary data, for example, if the modems or the software environments you program in are optimized for ASCII data transfer.

Third, when you think about your project, think about the messages that need to be exchanged, and come up with a data protocol that adequately describes all the information you need to send. This is your data packet. You might want to add header bytes, separators, or tail bytes to make reading the sequence easier.

Fourth, consider the flow of data, and look for ways to ensure a smooth flow with as little overflowing of buffers or waiting for data as possible. A simple call-and-response approach can make data flow much smoother.

Fifth, get to know the communications devices that link the objects at the end of your connection, whether they're protocol adapters like the USB-to-serial adapter or radios like the Bluefruit. Understand their addressing schemes and data protocols so that you can factor their strengths and limitations into your planning, and eliminate those parts that make your life more difficult.

Finally, think about your projects in terms of distributed computing rather than a single computer. Now that you know how to program both high-end microcontroller modules and simple microcontrollers like the ATtiny84 or ATtiny85, and how to communicate with them using asynchronous serial communication, you can assign each task in your project to a separate controller if you want.

With a little planning, you can take the computing needs of your project, and distribute them across many different computers. This is the real power of microcontrollers and serial communication.

X

» The JitterBox by Gabriel Barcia-Colombo

The JitterBox is an interactive video Jukebox created from a vintage 1940s radio restored to working condition. It features a tiny video-projected dancer who shakes and shimmies to the music. The viewer can tune the radio and the dancer will move in time with the tunes. The JitterBox uses serial communication from an embedded potentiometer tuner—which is connected to an Arduino microcontroller—in order to select from a range of vintage 1940s songs. These songs are linked to video clips and played back out of a digital projector.
The dancer trapped in the JitterBox is Ryan Myers.