CHAPTER 2

Hardware for IOT Solutions

Most IOT solutions, whether they are built by hand or mass produced for commercial sale, are prototyped from basic designs using discrete components. Most hobbyists and enthusiasts base their solutions on components they can buy from retailers (or wholesalers if they produce many units for sale). Much of the commodity hardware therefore is readily available.

This chapter introduces several examples of the more popular commodity hardware you can use in building your IOT solutions. Since there are so many varieties of everything available, I will not attempt or claim this chapter is all-inclusive. For example, I briefly describe a few of the common low-cost computing boards available, but there are more being added every day. It would require a tome several times the size of this book to list all of them. However, I show examples of how to use some of the hardware mentioned in this chapter in Chapter 8.

Included in this chapter are a discussion of popular Arduino microcontroller boards, a brief tutorial on using the Arduino software, a discussion of popular low-cost computing boards, a survey of communication hardware, and even a short discussion on what sort of sensors are available.

While you do not need to become an expert on any of the hardware listed, reading through this chapter will get you the knowledge you need to select which hardware to buy and get started using the hardware. After all, this is a book on using MySQL for IOT solutions rather than an in-depth instruction on how to build IOT solutions—but I have some of this information in this chapter and more in Chapter 8.

Let's begin by examining one of the most common and perhaps the single most versatile hardware component available for IOT solutions—the microcontroller.

HOBBYIST OR ENTHUSIAST: WHAT'S THE DIFFERENCE?

Enthusiasts build things because they can and often because they get a lot of enjoyment out of the build, not necessarily because they need to build it. Enthusiasts also tend to indulge in stockpiling or collecting. A hobbyist builds things for a singular purpose and do not normally indulge as much as enthusiasts. Thus, an enthusiast is a hobbyist who has turned the corner on avocation versus obsession.

Microcontrollers

Microcontrollers[1] are small integrated circuit (IC) chips arranged on a small printed circuit board (PCB) with additional components to support features such as access to pins on the chip for connecting other components such as LEDs, servos, sensors, and more. Microcontrollers tend to be limited in the amount of

[1] https://en.wikipedia.org/wiki/Microcontroller

memory available, have limited command features (such as operations for interacting with the hardware), have few connections (typically a programming and power connector), and often use specialized programming languages.

There are many microcontrollers available ranging from a bare IC with the AVR firmware installed[2] to a microcontroller-based platform that supports loadable programs and a wide range of hardware expansion. Fortunately, many such platforms are available.

One of the most popular and widely available microcontroller platforms is the Arduino. The following sections present a wide array of information about the Arduino including examples of boards you can buy and even a tutorial on how to program the Arduino.

■ **Note** I often use the terms *microcontroller* or *microcontroller platform* to discuss a product category and *board* to refer to a specific version of the platform.

What Is an Arduino?

The Arduino is an open source hardware prototyping platform supported by an open source software environment. It was first introduced in 2005 and was designed with the goal of making the hardware and software easy to use and available to the widest audience possible. Thus, you don't have to be an electronics expert to use the Arduino.

The original target audience included artists and hobbyists who needed a microcontroller to make their designs and creations more interesting. However, given its ease of use and versatility, the Arduino has quickly become the choice for a wider audience and a wider variety of projects.

This means you can use the Arduino for all manner of projects from reacting to environmental conditions to controlling complex robotic functions. The Arduino has also made learning electronics easier through practical applications.

Another aspect that has helped the rapid adoption of the Arduino platform is the growing community of contributors to a wealth of information made available through the official Arduino web site (`http://arduino.cc/en/`). When you visit the web site, you will find an excellent "getting started" tutorial as well as a list of helpful project ideas and a full reference guide to the C-like language for writing the code to control the Arduino (called a *sketch*).

The Arduino also provides an integrated development environment called the Arduino IDE. The IDE runs on your computer (called the *host*), where you can write and compile sketches and then upload them to the Arduino via USB connections. The IDE is available for Linux, Mac, and Windows. It's designed around a text editor that is especially designed for writing code and a set of limited functions designed to support compiling and loading sketches.

Sketches are written in a special format consisting of only two required methods—one that executes when the Arduino is reset or powered on and another that executes continuously. Thus, your initialization code goes in `setup()`, and your code to control the Arduino goes in `loop()`. The language is C-like, and you may define your own variables and functions. For a complete guide to writing sketches, see `http://arduino.cc/en/Tutorial/Sketch`.

You can expand the functionality of sketches and provide for reuse by writing libraries that encapsulate certain features such as networking, using memory cards, connecting to databases, doing mathematics, and the like. Many such libraries are included with the IDE. There are also some libraries written by others and contributed to Arduino.cc through open source agreements—some of which have been bundled with the IDE.

[2] `http://atmel.com/images/doc8161.pdf`

The Arduino supports a number of analog and digital *pins* that you can use to connect to various devices and components and interact with them. The mainstream boards have specific pin layouts, or *headers*, that allow the use of expansion boards called *shields*. Shields let you add additional hardware capabilities such as Ethernet, Bluetooth, and XBee support to your Arduino. The physical layout of the Arduino and the shield allow you to stack shields. Thus, you can have an Ethernet shield as well as an XBee shield, because each uses different I/O pins. You will learn about the use of the pins and shields as you explore how to apply the Arduino to IOT networks.

The Arduino is used best with a breadboard when developing or prototyping circuits. A breadboard is designed to allow you to plug in your electrical components and provide interconnectivity in columns so that you can plug the leads of two components into the same column and therefore make a connection. The board is split into two rows, making it easy to use an IC in the center of the board. Wires (called *jumpers*) can be used to connect the circuit on the breadboard to the Arduino. You will see an example of this later in this chapter.

The next sections examine some of the various Arduino boards—both Arduino branded and third party. Many more boards and variants are available, and a few new ones are likely to be out by the time this book is printed, but these are the ones that I use in my IOT projects and experiments. Any of these can be an excellent basis for your own projects.

Arduino Models

A growing number of Arduino boards are available. Some are configured for special applications, whereas others are designed with different processors and memory configurations. Some boards are considered official Arduino boards because they're branded and endorsed by Arduino.cc. Because the Arduino is open source and, more specifically, licensed using a Creative Commons Attribution Share-Alike license, anyone can build Arduino-compatible boards (often called Arduino *clones*). However, you must follow the rules and guidelines set forth by Arduino.cc.[3] This section examines some of the more popular Arduino-branded boards.

The basic layout of an Arduino board consists of a USB connection, a power connector, a reset switch, LEDs for power and serial communication, and a standard-spaced set of headers for attaching shields. The official boards sport a distinctive blue PCB with white lettering. With the exception of one model, all the official boards can be mounted in a chassis (they have holes in the PCB for mounting screws). The exception is an Arduino designed for mounting on a breadboard.

Arduino Zero

The Arduino Zero is the latest small-footprint Arduino board available from Arduino.cc. It has the same physical layout as the boards in the Arduino Uno family (`http://arduino.cc/en/Main/ArduinoBoardUno`) but has a much faster, 32-bit processor. The board is similar to the Leonardo board (see the "Arduino Leonardo" section) with 20 digital I/O pins, of which 18 can be used as analog pins. It has more memory with 256KB of flash memory and 32KB of SRAM. The clock speed is also faster at 48MHz. Figure 2-1 shows an early release of the Arduino Zero board.

[3] For a complete description of the Arduino.cc license policies and more information about building and selling your own Arduino-compatible board, see `http://arduino.cc/en/Main/FAQ`.

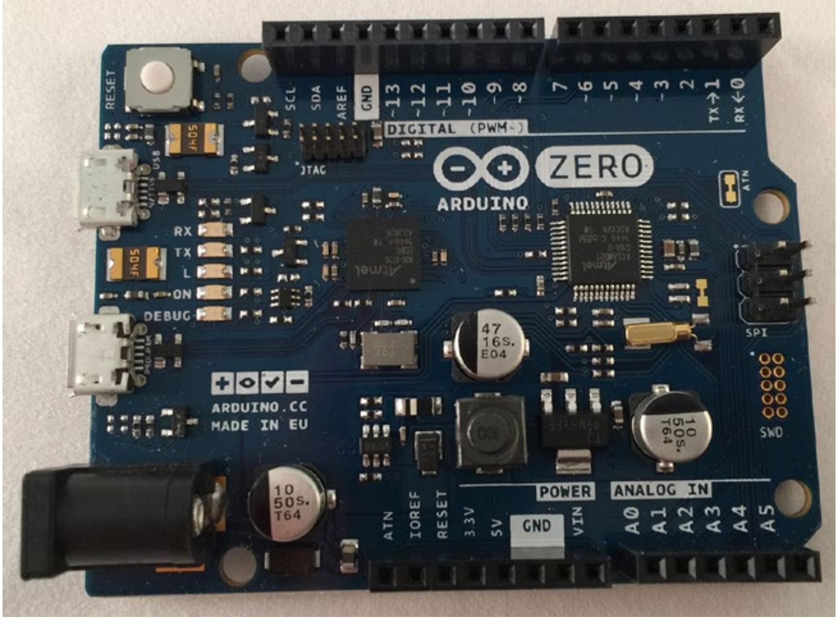


Figure 2-1. *Arduino Zero*

The board comes equipped with a USB host port so you can write sketches to access devices via USB such as a keyboard or mouse.

The Arduino Zero is an excellent choice for most Arduino projects and can solve a number of issues with memory and performance using the older boards. While not a replacement for poor or inefficient programming, the Arduino Zero can improve solutions that require more processing power and, more importantly, more memory.

■ **Tip** The Arduino Zero runs on 3.3V. Take extra care when using shields and connecting components to ensure you do not damage the board.

While this is the newest Arduino board in my portfolio of microcontrollers, I have used this board to test sketches that require more memory. Indeed, it has already proven to be an excellent choice for a node in my IOT solutions where I aggregate or augment the data. The added memory means I can also issue queries on the database server for looking up values for calculations.

You can find specific documentation for the Arduino Zero at `http://arduino.cc/en/Main/ArduinoBoardZero`.

Arduino Yún

The Arduino Yún is a very different Arduino board. While it is and supports use as a normal Arduino board (you can run the same sketches), the Yún has two processors: the Atmel ATmega32U4 is a Leonardo-compatible microcontroller, and the Atheros AR9331 runs a scaled-down version of Linux and the OpenWrt wireless stack.

The Yún has a USB host controller as well as both WiFi and Ethernet networking. Indeed, you can use the WiFi capabilities to form a wireless access port, making it possible for IOT solutions to host their own WiFi-connected devices. Like the Zero, this board runs on 3.3V, requiring you to select your components and shields carefully to ensure you do not damage the board. Figure 2-2 depicts the Arduino Yún.

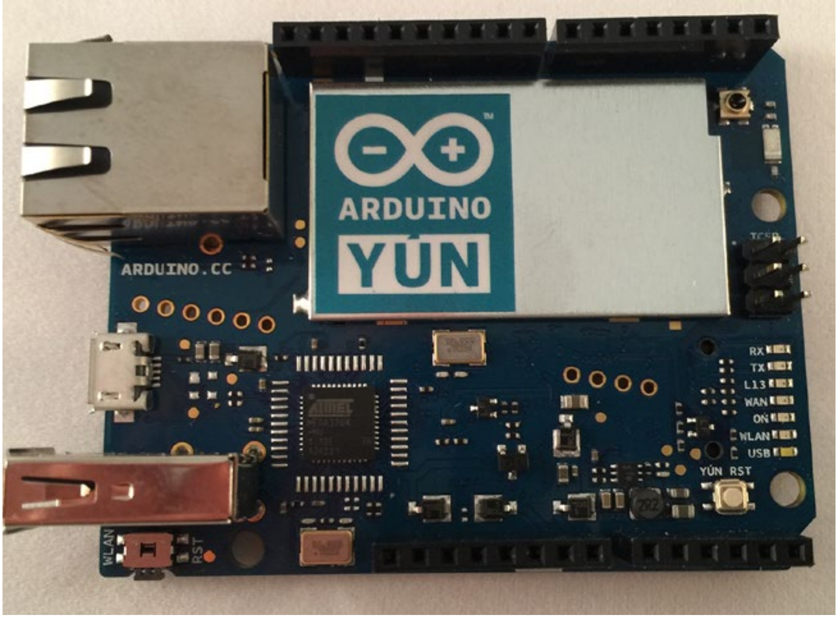


Figure 2-2. *Arduino Yún*

The Yún is not a board for everyone. However, if you need to be able to combine the use of Arduino shields with more powerful scripts written in Python, the Yún is perhaps a good choice for combining the power of Python with the versatility of Arduino-compatible components and shields. There are other boards that do this a bit better, but most do not include the WiFi access port capability. If you need or want to provide a powerful wireless access port feature in your solution that can run any of your Arduino sketches, the Yún is a good choice.

So far, I've used my Yún to create discrete WiFi networks for forensics and diagnosis of networking problems. Using the Yún as an access port permits me to do things I would not do on my own WiFi network. After all, if things go bad, I can just reset my Yún and try it again. It is far more risky to try that with your own wireless access router.

You can find specific documentation for the Arduino Yún at `http://arduino.cc/en/Guide/ArduinoYun`.

Arduino Leonardo

The Leonardo board is an evolution of the older, Uno-based boards. Although it supports the standard header layout, it doesn't support some of the older shields. However, it adds a faster processor and a USB controller that allows the board to appear as a USB device to the host computer. The newer ATmega32u4 processor has 20 digital I/O pins, of which 12 can be used as analog pins and 7 can be used as a pulse-width modulation (PWM) output. It has 32KB of flash memory and 2.5KB of SRAM.

The Leonardo has more digital pins than its predecessor but continues to support most shields. The USB connection uses a smaller USB connector. The board is also available with and without headers. Figure 2-3 depicts an official Leonardo board.

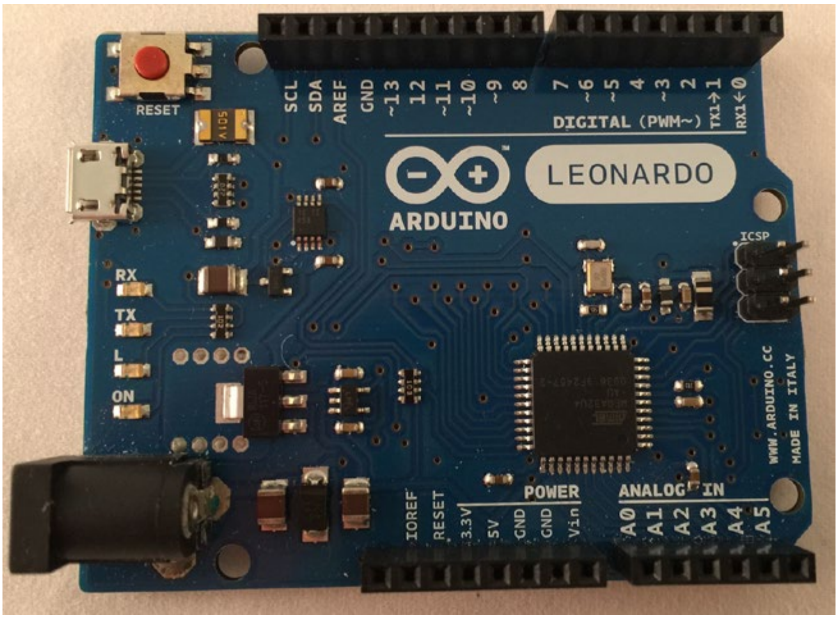


Figure 2-3. *Arduino Leonardo*

I have used the Leonardo to upgrade many of my older Arduino-based projects. While the board doesn't offer a lot more capability (but there is some, particularly memory) than some of the older boards, I am able to use the newer shields, which are starting to require the new header layout.

You can find specific documentation for the Arduino Leonardo at `http://arduino.cc/en/Main/ArduinoBoardLeonardo`.

Arduino Due

The Arduino Due is a new, larger, faster board based on the Atmel SAM3X8E ARM Cortex-M3 processor. The processor is a 32-bit processor, and the board supports a massive 54 digital I/O ports, of which 14 can be used for PWM output; 12 analog inputs; and 4 UART chips (serial ports); as well as two digital-to-analog (DAC) and two two-wire interface (TWI) pins. The new processor offers several advantages.

- 32-bit registers
- DMA controller (allows CPU-independent memory tasks)
- 512KB flash memory
- 96KB SRAM
- 84MHz clock

The Due has a larger form factor (called the *mega footprint*) but still supports the use of standard shields, but it also supports the mega format shields. The new board has one distinct limitation: unlike other boards that can accept up to 5V on the I/O pins, the Due is limited to 3.3V on the I/O pins. Figure 2-4 shows an Arduino Due board.

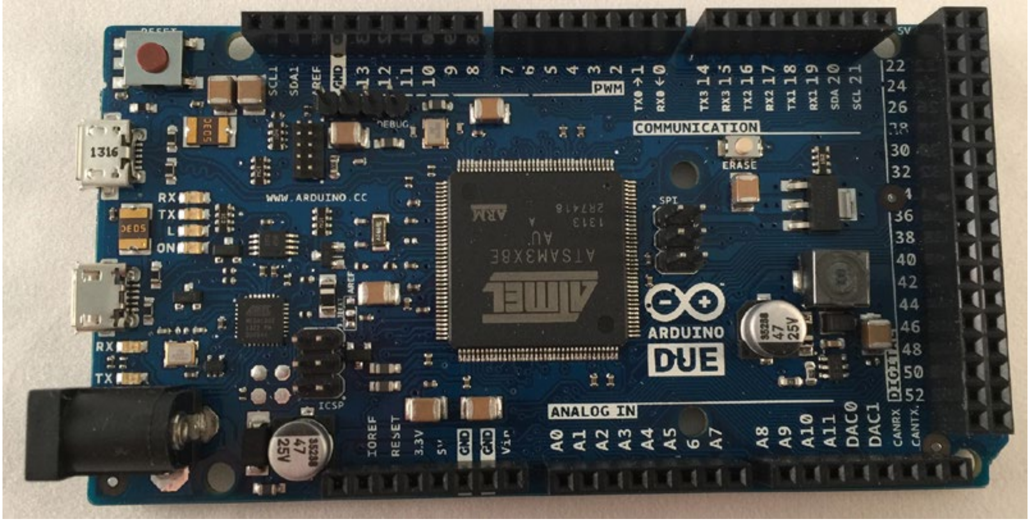


Figure 2-4. *Arduino Due*

The Arduino Due can be used for projects that require more processing power, more memory, and more I/O pins. I've used the Due in rare cases where I have needed a lot more I/O pins or more memory. However, with the new Zero, unless I need the added I/O pins, I'd likely use the Zero where physical size is an issue. If you don't have a space issue, look to the Due for your projects that require the maximum hardware performance.

You can find specific documentation for the Arduino Due at `http://arduino.cc/en/Main/ArduinoBoardDue`.

Arduino Mega 2560

The Arduino Mega 2560 is an older form of the Due. It's based on the ATmega2560 processor (hence the name). Like the Due, the board supports a massive 54 digital I/O ports, of which 14 can be used as PWM output; 16 analog inputs; and 4 UARTs (hardware serial ports). It uses a 16MHz clock and has 256KB of flash memory. Figure 2-5 shows the Arduino Mega 2560 board.

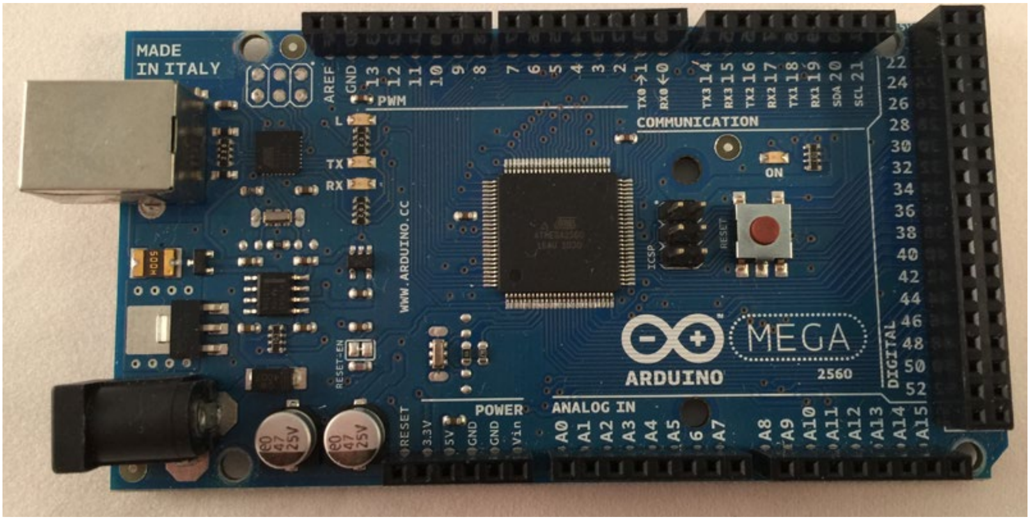


Figure 2-5. *Arduino Mega 2560*

The Mega 2560 is essentially a larger form of the standard Arduino (Uno, Duemilanove, and so on) and supports the standard shields. Interestingly, the Arduino Mega 256 is the board of choice for Prusa Mendel and similar 3D printers that require the use of a controller board named RepRap Arduino Mega Pololu Shield (RAMPS). Indeed, this is where most of my Mega 2560 boards are right now—in my 3D printers! That said, like the Due, you could use this board if you require additional I/O pins or more memory and size is not an issue (it is the same size and the Due).

You can find specific documentation for the Arduino Due at http://arduino.cc/en/Main/ArduinoBoardDue.

Arduino Clones

There are a growing number of third-party Arduino boards (also called *clones*). They range in form factor, components, and even features. Fortunately, most are 100 percent Arduino compatible, making it easy to use them in your Arduino-based projects. Even if the board is not 100 percent compatible with a particular Arduino board (for example, the Leonardo), third-party boards have been added to the Arduino integrated development environment requiring only that you select the specific board when compiling. You will see this later in this chapter.

The following sections describe some of the third-party Arduino boards that I use in my projects. Chances are you will encounter some of these and even more like them. As you will see, some have very different layouts, giving you more options when implementing your projects. Since these boards retain compatibility with the Arduino-branded boards, I describe them briefly by focusing on the unique features.

Sparkfun Redboard

The Sparkfun Redboard (http://sparkfun.com/products/12757) is an advanced version of the Arduino Uno. It uses the same boot loader, the addition of an FTDI interface (used in the older Duemilanove boards), and the R3 shield compatibility of the latest Arduino UNO R3. In effect, Sparkfun has reinvented the Arduino Uno by bringing back the best qualities of the older boards. And, yes, it is red. Figure 2-6 shows the Sparkfun Redboard.

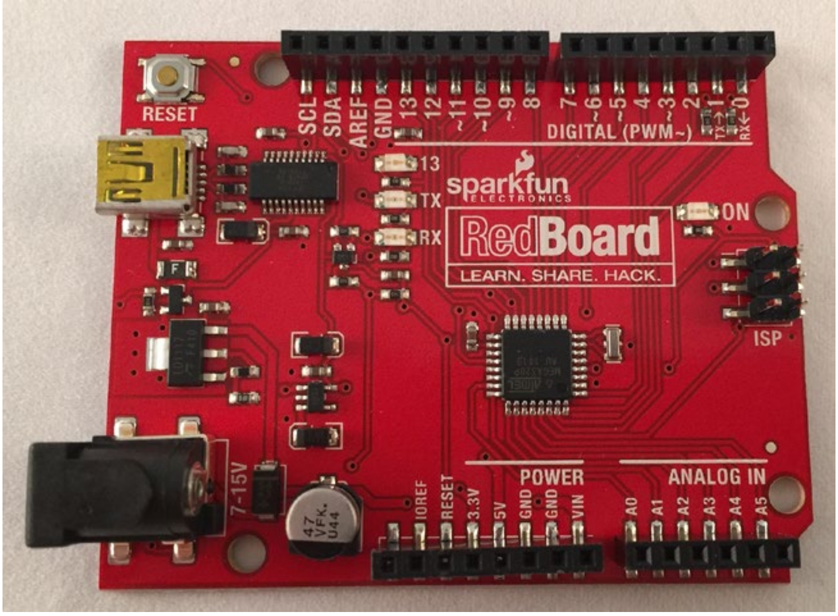


Figure 2-6. *Sparkfun Redboard*

What I like most about the Redboard is it is a bit cheaper than the Arduino-branded boards and is readily available from Sparkfun. Better still, Sparkfun stocks a number of shields and accessories that complement the Redboard. In fact, Sparkfun has an excellent kit called the Sparkfun Inventor's Kit (`http://sparkfun.com/products/12060`) that help you learn about Arduino programming and building circuits.

■ **Tip** If you have friends and family interested in learning more about Arduino, the Sparkfun Inventor's Kit would make an excellent gift.

You can find specific documentation for the Sparkfun Redboard at `https://learn.sparkfun.com/tutorials/redboard-hookup-guide`.

TinyCircuits TinyDuino

The TinyDuino is an interesting diversion from the standard Uno physical layout. As the name suggests, it is a small board. In fact, it is only slightly larger than a U.S. quarter![4] As you can see in Figure 2-7, the main board (center bottom) is about 21×21mm. That's small.

[4] Or as my grandmother used to say, "A twenty-and-five-cent piece."

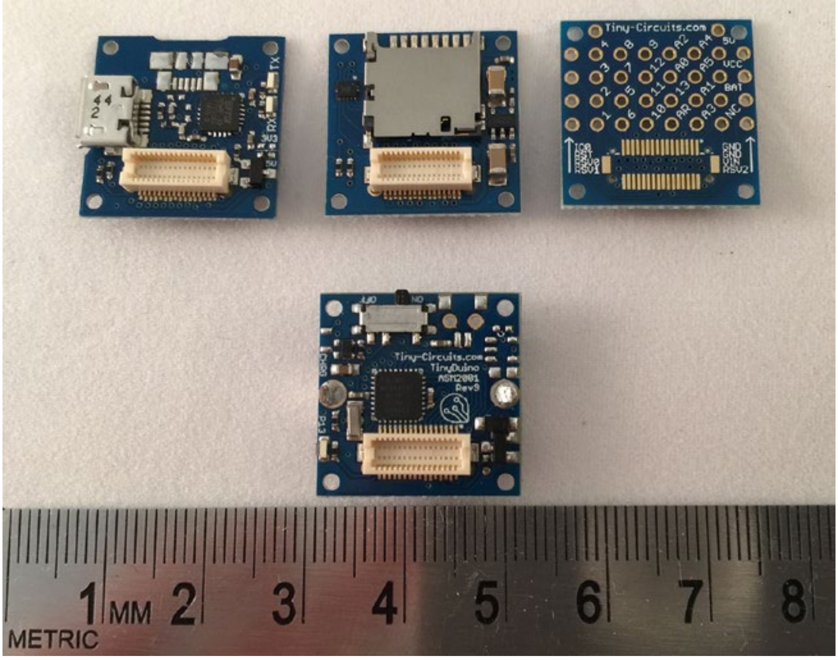


Figure 2-7. *TinyDuino*

Figure 2-7 shows four modules (also called *boards* or *core modules*); from left to right on the top row is a USB shield used to program the TinyDuino, a MicroSD card reader, and a prototyping board. The module in the center bottom is the main processor module.

The TinyDuino is compatible with the Arduino Uno with the same processor, memory, pins, and so on. There are some minor differences but nothing that is of any concern. In fact, the TinyDuino will run all of your sketches and can be programmed as if it were an Arduino Uno. You just cannot use Arduino shields with the TinyDuino.

Aside from the small size, the TinyDuino processor board has a battery holder that uses a CR1632 coin cell battery to power the board. This allows you to keep your Arduino solution very small and even run on battery power! Better still, there are additional versions of the main processor board that have a Lithium battery connector and even one without battery support that you can use to hardwire a power source.

The boards connect via a microconnector that permits you to stack them on top of one another. They can be stacked in any order with some exceptions. The processor board does not have a connector on the bottom, and the prototype boards do not have a connector on top. Regardless, even if you stack all four boards as shown, the TinyDuino stack is only about 20mm tall.

■ **Tip** You must have the USB board in order to program the TinyDuino, but once it's programmed, you can remove the board.

There is one other board that I consider a must-have if you need to use more than a few pins. The prototype boards have a number of pins broken out, but not all of them. The terminal module shown in Figure 2-8 allows access to all the Arduino pins and comes complete with screw terminals for every pin! That's cool. While it is a bit larger than the core processor module, it is still very small.

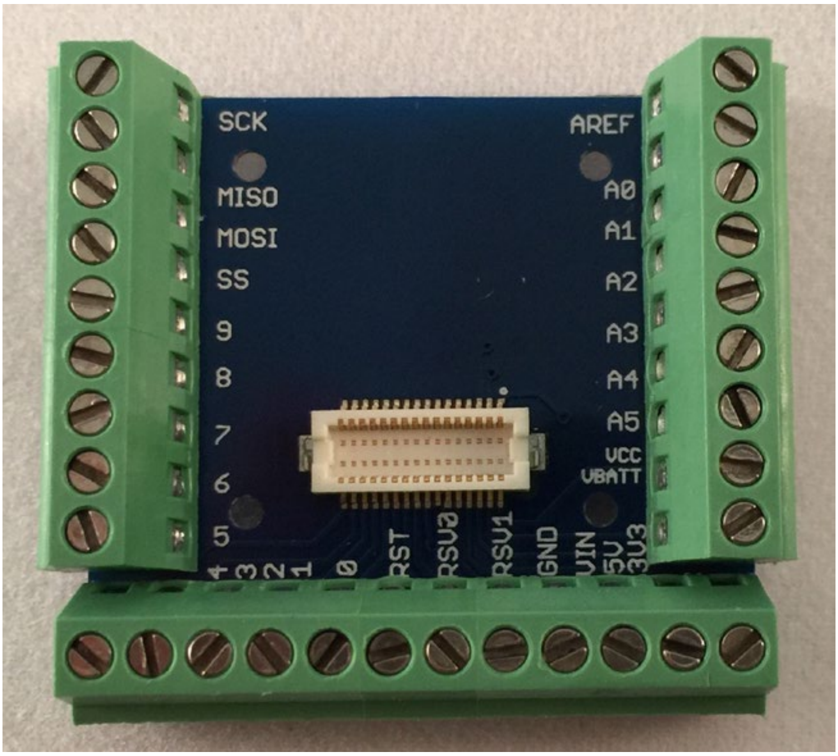


Figure 2-8. *TinyDuino terminal board*

Better still, TinyCircuits (http://tiny-circuits.com/), the maker of the TinyDuino, offers a growing selection of modules to add to your tiny Arduino kit. There are modules for LEDs, Bluetooth, WiFi, real-time clock (RTC), accelerometer, audio, dual seven-segment display, GPS, and even a micro-sized OLED screen. With so many modules available, you can build a really powerful Arduino solution in a fraction of the size of the normal Arduino boards.

Thus, the TinyDuino represents an excellent choice for solutions that need to be as small as possible. I've seen these boards used in wearables,[5] inside small toys, and even in wristwatch-sized Arduino devices. The small size and lower power make it ideal for hiding in small places.

You can find specific documentation for the TinyDuino at http://tiny-circuits.com/tinyduino_overview.

SpikenzieLabs Sippino

The Sippino from SpikenzieLabs (www.spikenzielabs.com) can be used on a solder-less breadboard. It costs less because it has fewer components and a much smaller footprint. Fortunately, SpikenzieLabs also provides a special adapter called a *shield dock* that allows you to use a Sippino with standard Arduino shields.

[5] TinyCircuits also makes a LilyPad-compatible module line the size of a U.S. dime (ten-cent piece) called the TinyLily (http://tiny-circuits.com/products/tiny-lily.html).

It’s based on the ATmega328 processor and has 14 digital I/O pins, of which 6 can be used as PWM output, and 6 analog input pins. The Sippino board has 32KB of flash memory and 2KB of SRAM. Figure 2-9 shows a Sippino with breadboard headers.

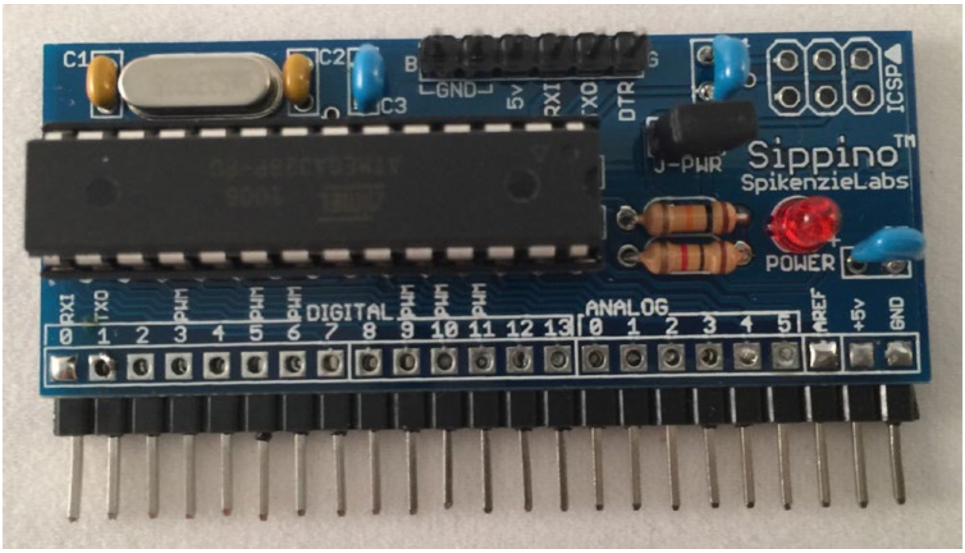


Figure 2-9. *Sippino*

The Sippino doesn’t have a USB connection, so you have to use an FTDI cable to program it. The good news is you need only one cable no matter how many Sippinos you have in your project. I have a number of Sippinos and use them in many of my Arduino projects where space is at a premium.

The shield dock is an amazing add-on that lets you use the Sippino as if it were a standard Uno or Duemilanove. Figure 2-10 shows a Sippino mounted on a shield dock.

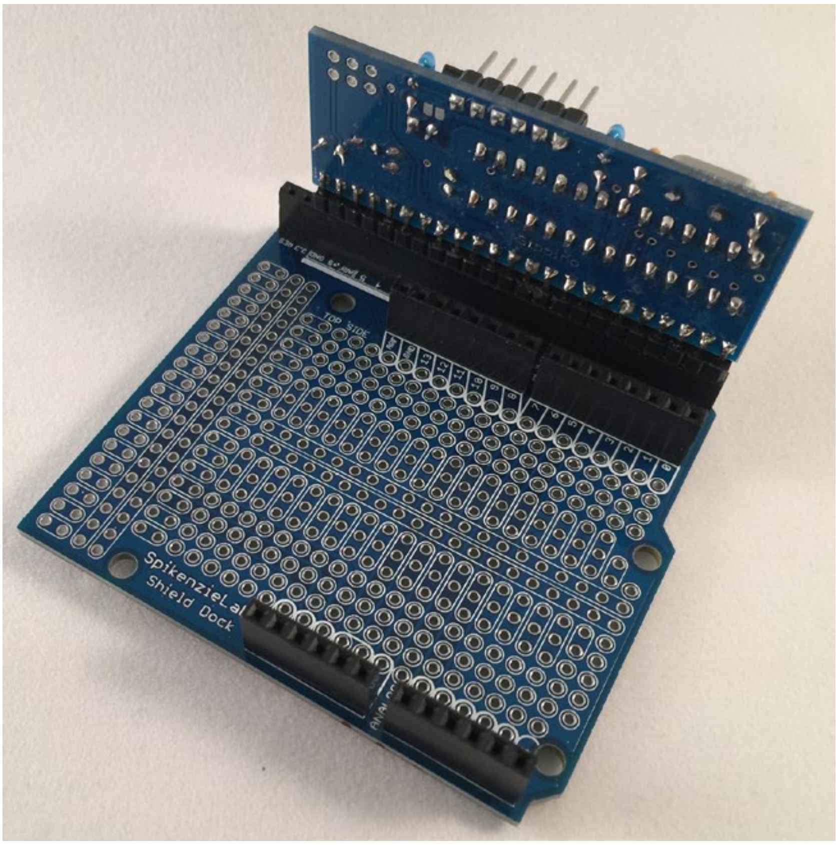


Figure 2-10. *Sippino shield dock*

I have used the Sippino in numerous projects from small geeky alarm clocks (think servos, analog gauges, and old-school glowing bulbs) to sensor nodes in sensor networks. The small size and card layout make it easy to hide the Sippino in a small enclosure. If you need or want something compatible with the older Arduino boards (or need to occasionally use it with an Arduino Uno–compatible shield) in a small package, the Sippino is a good choice.

Another reason I like these boards is they are sold as kits where you assemble the board yourself. The assembly is easy, and except for reading the values of the resistors (a skill every electronics hobbyist must master), the components are easy to align to the correct spot on the board.

You can find specific documentation for the Sippino and Shield dock at http://spikenzielabs.com.

Spikenzie Labs Prototino

The Prototino is another product of SpikenzieLabs. It has the same components as the Sippino, but instead of a breadboard-friendly layout, it's mounted on a PCB that includes a full prototyping area. Like the Sippino, it's based on the ATmega328 processor and has 14 digital I/O pins, of which 6 can be used as PWM output, and 6 analog input pins. The Prototino board has 32KB of flash memory and 2KB of SRAM.

The Prototino is ideal for building solutions that have supporting components and circuitry. In some ways it's similar to the Nano, Mini, and similar boards in that you can use it for permanent installations. But unlike those boards (and even the Arduino Pro), the Prototino provides a space for you to add your components directly to the board. I have used a number of Prototino boards for projects where I have added the components to the Prototino and installed it in the chassis. This allowed me to create a solution using a single board and even build several copies quickly and easily.

Like the Sippino, the Prototino doesn't have a USB connection, so you have to use an FTDI cable to program it. Figure 2-11 shows a Prototino board.

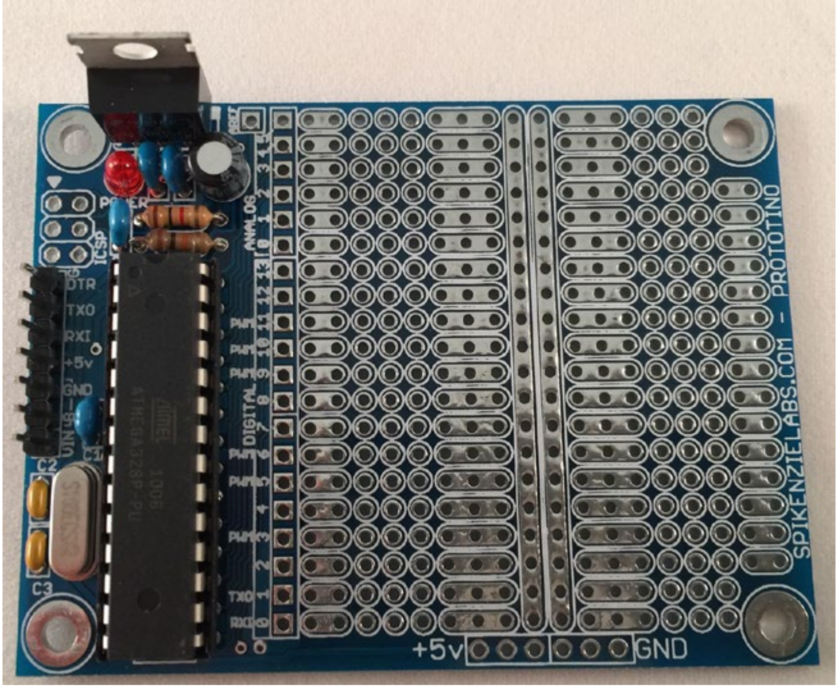


Figure 2-11. *Prototino*

I've used the Prototino in a number of projects where I had prototyped a circuit on a breadboard and wanted to quickly transfer it to a permanent board.[6] In fact, I started using the Prototino in many of my early projects because it was so easy to work with the breadboard layout of the solder pads.

Like the Sippino, the Prototino is available only as a self-assembly kit. You can find specific documentation for the Prototino at `http://spikenzielabs.com`.

Sparkfun ESP8266 WiFi Module

There is one last board I have in my collection of third-party Arduino boards. The ESP8266 WiFi module is one of the latest additions to the category. It is sometimes listed under the IOT category because of its unique features. Sparkfun's ESP8266 Thing has many cool features such as a LiPo battery connector, on/off switch (very handy), an external antenna connector (also handy for increasing range), a USB programming port, and of course all the available pins broken out. The board is also very small.

[6] Well, permanent until you unsolder it!

You can program the ESP8266 with the Arduino integrated development environment by adding a special add-on. See https://github.com/esp8266/Arduino for more details on how to configure the board manager. Figure 2-12 shows the Sparkfun ESP8266 board.

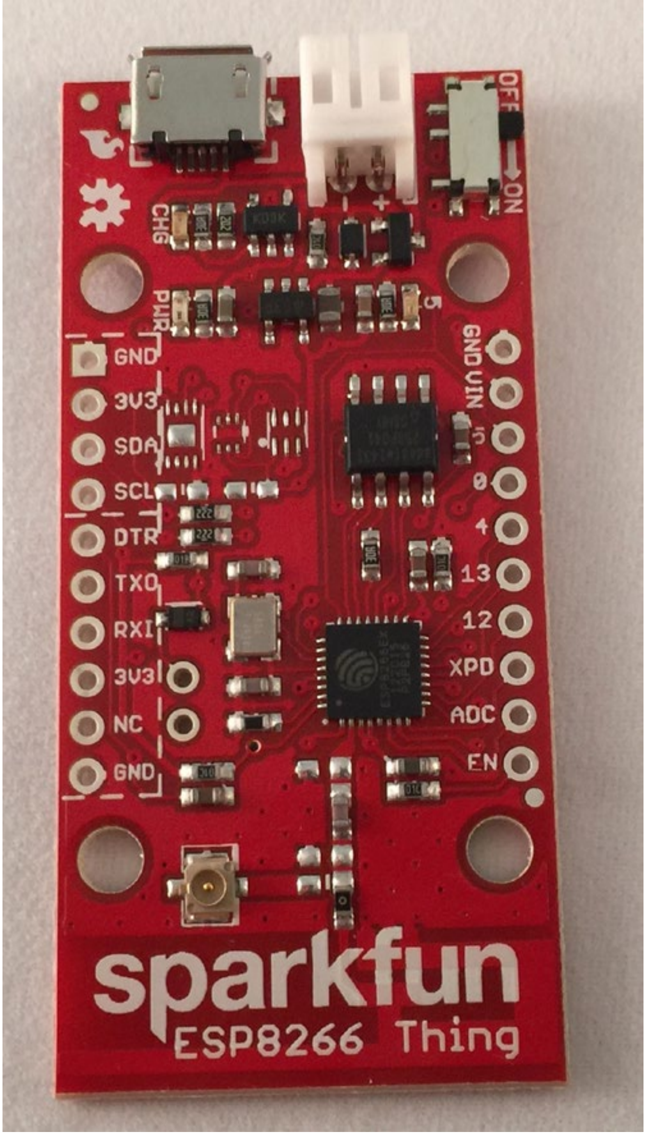


Figure 2-12. *ESP8266 Thing*

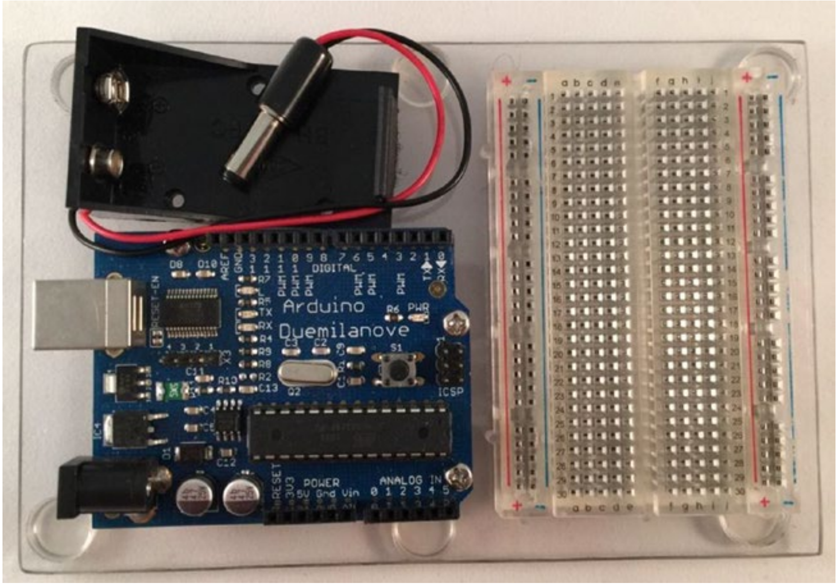
The ESP8266 represents yet another small physical layout board that you can use to create nodes for your IOT network (or single solution). I haven't used mine in any real projects yet, but the power of the module is appealing.

Sparkfun also sells an Arduino-sized board called the ESP8266 WiFi shield that you can use with your Arduino boards (`http://sparkfun.com/products/13287`). It can be used to provide WiFi capabilities to your Arduino projects. I have one of these too and am using it in place of the normal Arduino WiFi shield (see the "Recommended Accessory" sidebar) mainly because I wanted to experiment with the ESP8266 WiFi capabilities without the extra work to program the ESP8266 Thing.

You can find more information about the ESP8266 in Sparkfun's excellent resource library at `https://learn.sparkfun.com/tutorials/esp8266-thing-hookup-guide`.

RECOMMENDED ACCESSORY

If you plan to do any prototyping with your Arduino or other microcontroller board, I recommend getting or making your own experiment base (sometimes called a *breadboard holder*). You can easily make one from acrylic as I have done in the example shown here. In this case, I have an older Duemilanove board mounted next to a half-sized breadboard. I even had room to place a 9V battery holder for powering the experiment.



You can also buy them pre-drilled for most Arduino Uno–sized boards. Both Sparkfun (`http://sparkfun.com/products/11235`) and Adafruit (`http://adafruit.com/products/275`) stock excellent examples. You can also find these for other boards such as the Raspberry Pi.

Keeping the microcontroller and breadboard together makes it possible to move the experiment from one place to another or stop and resume at a later date.

Arduino Tutorial

This section is a short tutorial on getting started using an Arduino. It covers obtaining and installing the IDE and writing a sample sketch. Rather than duplicate the excellent works that precede this book, I cover the highlights and refer readers who are less familiar with the Arduino to online resources and other books that offer a much deeper introduction. Also, the Arduino IDE has many sample sketches that you can use to explore the Arduino on your own. Most have corresponding tutorials on the `http://arduino.cc` site.

Learning Resources

A lot of information is available about the Arduino platform. If you're just getting started with the Arduino, Apress offers an impressive array of books covering all manner of topics concerning the Arduino, ranging from getting started using the microcontroller to learning the details of its design and implementation. The following is a list of the more popular books:

- *Beginning Arduino* by Michael McRoberts (Apress, 2010)
- *Practical Arduino: Cool Projects for Open Source Hardware (Technology in Action*) by Jonathan Oxer and Hugh Blemings (Apress, 2009)
- *Arduino Internals* by Dale Wheat (Apress, 2011)

There are also some excellent online resources for learning more about the Arduino, the Arduino libraries, and sample projects. The following are some of the best:

- *Arduino.cc*: `http://arduino.cc/en/`
- *Adafruit tutorials*: `http://learn.adafruit.com/`
- *Make tutorials*: `http://makezine.com/category/electronics/arduino/`

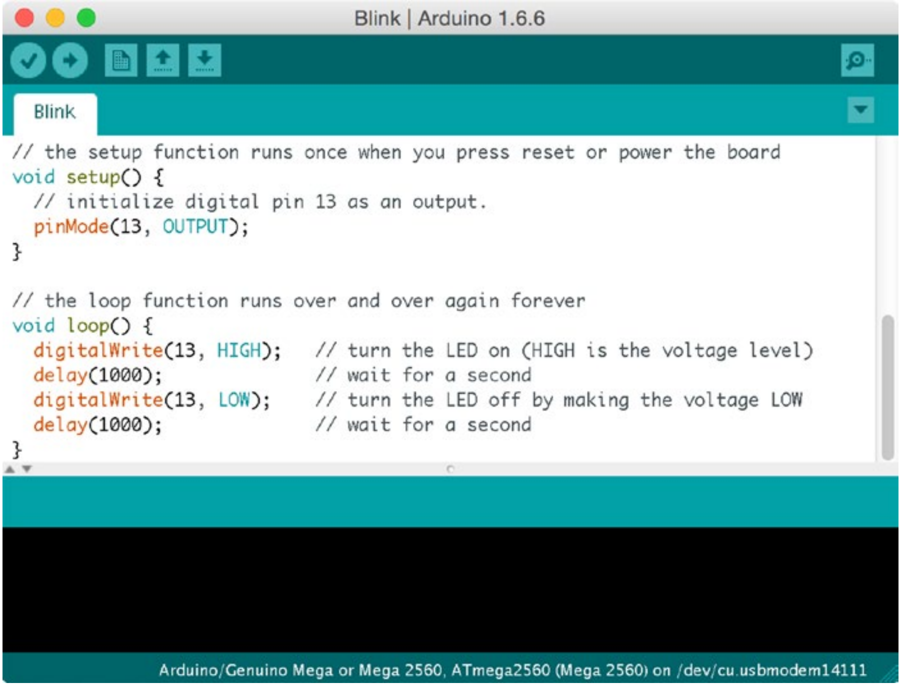
The Arduino IDE

The Arduino IDE is available for download for the Mac, Linux (32- and 64-bit versions), and Windows platforms. You can download the IDE from `http://arduino.cc/en/Main/Software`. There are links for each platform as well as a link to the source code if you need to compile the IDE for a different platform.

Installing the IDE is straightforward. I omit the actual steps of installing the IDE for brevity, but if you require a walk-through of installing the IDE, you can follow the Getting Started link on the download page or read more in *Beginning Arduino* by Michael McRoberts (Apress, 2010).

■ **Tip** See `http://arduino.cc/en/Guide/Howto` if you need help installing the drivers on Windows.

Once the IDE launches, you see a simple interface with a text editor area (a white background by default), a message area beneath the editor (a black background by default), and a simple button bar at the top. The buttons are (from left to right) Compile, Compile and Upload, New, Open, and Save. There is also a button to the right that opens the serial monitor. You use the serial monitor to view messages from the Arduino sent (or printed) via the Serial library. You will see this in action in your first project. Figure 2-13 shows the Arduino IDE.



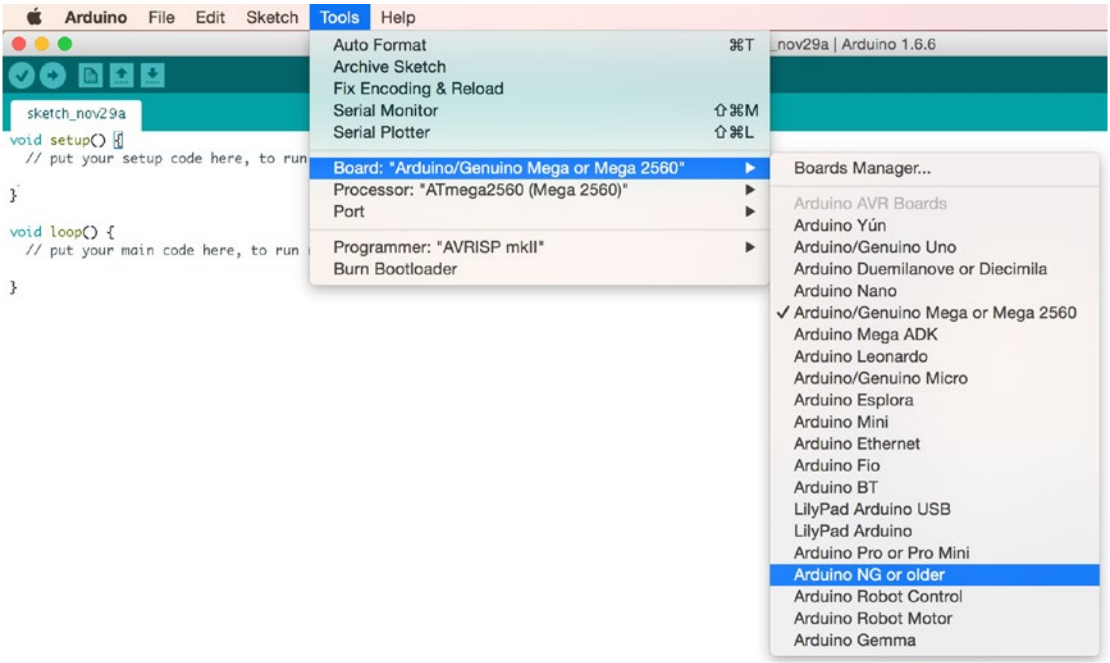
```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) on /dev/cu.usbmodem14111

Figure 2-13. *The Arduino IDE*

Notice that in Figure 2-13 you can see a sample sketch (called a *blink*) and the result of a successful compile operation. Notice also at the bottom it tells you that you're programming an Arduino Uno board on a specific serial port.

Because of the differences in processor and supporting architecture, there are some differences in how the compiler builds the program (and how the IDE uploads it). Thus, one of the first things you should do when you start the IDE is choose your board from the Tools ➤ Board menu. Figure 2-14 shows a sample of selecting the board on a Mac.

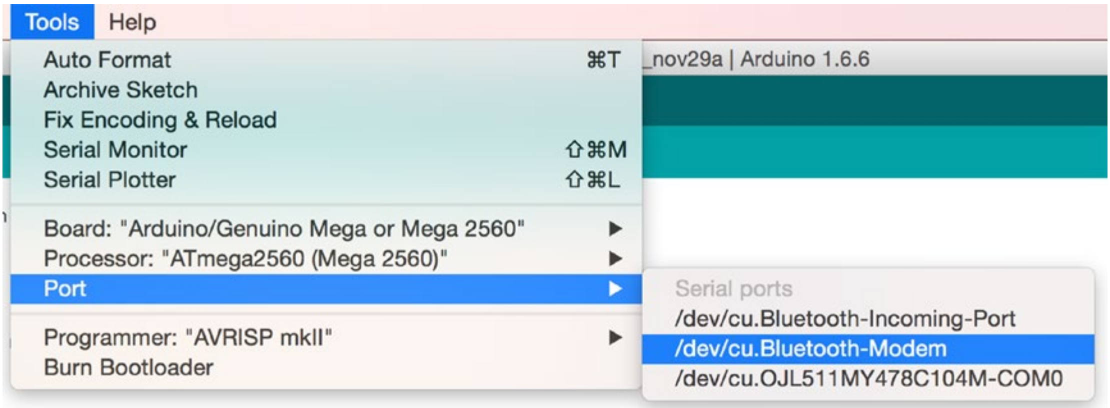


```
void setup() {
  // put your setup code here, to run

}

void loop() {
  // put your main code here, to run

}
```

Figure 2-14. *Choosing the Arduino board*

Notice the number of boards available. Be sure to choose the one that matches your board. If you're using a clone board, check the manufacturer's site for the recommended setting to use. If you choose the wrong board, you typically get an error during upload, but it may not be obvious that you've chosen the wrong board. Because I have so many different boards, I've made it a habit to choose the board each time I launch the IDE.

The next thing you need to do is choose the serial port to which the Arduino board is connected. To connect to the board, use the Tools ➤ Port menu option. Figure 2-15 shows an example on a Mac. In this case, no serial ports are listed. This can happen if you haven't plugged your Arduino into the computer's USB ports (or hub), if you had it plugged in but disconnected it at some point, or if you haven't loaded the drivers for the Arduino (Windows). Typically, this can be remedied by simply unplugging the Arduino and plugging it back in and waiting until the computer recognizes the port.



Board: "Arduino/Genuino Mega or Mega 2560"
Processor: "ATmega2560 (Mega 2560)"
Programmer: "AVRISP mkII"

Figure 2-15. *Choosing the serial port*

■ **Note** If you use a Mac, it doesn't matter which port you choose—either the one that starts with `tty` or the one that starts with `cu` will work.

Now that you have your Arduino IDE installed, you can connect your Arduino and set the board and serial port. You see the LEDs on the Arduino illuminate. This is because the Arduino is getting power from the USB. Thus, you don't need to provide an external power supply when the Arduino is connected to your computer. Next, let's dive into a simple project so you can see the Arduino IDE and learn how basic sketches are built, compiled, and uploaded.

Project: Hardware "Hello, World!"

The ubiquitous "Hello, World!" project for the Arduino is the blinking light. The project uses an LED, a breadboard, and some jumper wires. The Arduino turns on and off through the course of the `loop()` iteration. That's a fine project for getting started, but it doesn't relate to how sensors could be used.

Thus, in this section, you expand on the blinking light project by adding a sensor. In this case, you still keep things simple by using what is arguably the most basic of sensors: a pushbutton. The goal is to illuminate the LED whenever the button is pushed.

Wiring the Circuit

Let's begin by assembling an Arduino. Be sure to disconnect (power down) the Arduino first. You can use any Arduino variant that has I/O pins. Place one LED and one pushbutton in the breadboard. Wire the 5V pin to the breadboard power rail and the ground pin to the ground rail, and place the pushbutton in the center of the breadboard. Place the LED to one side of the breadboard, as shown in Figure 2-16.

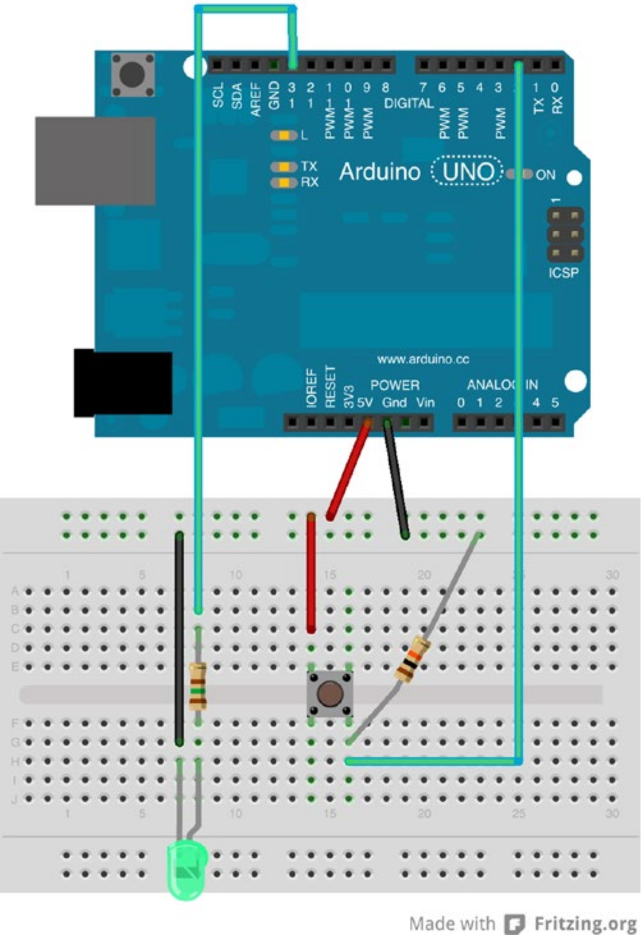


Figure 2-16. *Diagram of an LED with a pushbutton*

■ **Tip** If you power on your shiny new Arduino, you may see the LED on the board flash. This is because some Arduino boards come with the blink sketch preloaded.

You're almost there. Now wire a jumper from the power rail to one side of the pushbutton, and wire the other side of the pushbutton to (DIGITAL) pin 2 on the Arduino (located on the side with the USB connector). Next, wire the LED to ground on the breadboard and a 150 ohm resistor (colors: brown, green, brown, gold). The other side of the resistor should be wired to pin 13 on the Arduino. You also need a resistor to pull the button low when the button isn't pressed. Place a 10K ohm resistor (colors: brown, black, orange, gold) on the side of the button with the wire to pin 2 and ground.

The longest side of the LED is the positive side. The positive side should be the one connected to the resistor. It doesn't matter which direction you connect the resistor; it's used to limit the current to the LED. Check the drawing again to ensure that you have a similar setup.

■ **Note** Most Arduino boards have an LED connected to pin 13. You reuse the pin to demonstrate how to use analog output. Thus, you may see a small LED near pin 13 illuminate at the same time as the LED on the breadboard.

Writing the Sketch

The sketch you need for this project uses two I/O pins on the Arduino: one output and one input. The output pin will illuminate the LED, and the input pin will detect the pushbutton engagement. You connect positive voltage to one side of the pushbutton and the other side to the input pin. When you detect voltage on the input pin, you tell the Arduino processor to send positive voltage to the output pin. In this case, the positive side of the LED is connected to the output pin.

As you see in the drawing in Figure 2-17, the input pin is pin 2, and the output pin is pin 13. Let's use a variable to store these numbers so you don't have to worry about repeating the hard-coded numbers (and risk getting them wrong). Use the pinMode() method to set the mode of each pin (INPUT, OUTPUT). You place the variable statements before the setup() method and set the pinMode() calls in the setup() method, as follows:

```
int led = 13;     // LED on pin 13
int button = 2;   // button on pin 2

void setup() {
  pinMode(led, OUTPUT);
  pinMode(button, INPUT);
}
```

In the loop() method, you place code to detect the button press. Use the digitalRead() method to read the status of the pin (LOW or HIGH), where LOW means there is no voltage on the pin and HIGH means positive voltage is detected on the pin.

You also place in the loop() method the code to turn on the LED when the input pin state is HIGH. In this case, you use the digitalWrite() method to set the output pin to HIGH when the input pin state is HIGH and similarly set the output pin to LOW when the input pin state is LOW. The following shows the statements needed:

```
void loop() {
  int state = digitalRead(button);
  if (state == HIGH) {
    digitalWrite(led, HIGH);
  }
  else {
    digitalWrite(led, LOW);
  }
}
```

Now let's see the entire sketch, complete with the proper documentation. Listing 2-1 shows the completed sketch.

Listing 2-1. Simple Sensor Sketch

```
/*
  Simple Sensor - MySQL for the IOT

  For this sketch, we explore a simple sensor (a pushbutton) and a simple
  response to sensor input (a LED). When the sensor is activated (the
  button is pushed), the LED is illuminated.
*/

int led = 13;     // LED on pin 13
int button = 2;   // button on pin 2

// the setup routine runs once when you press reset:
void setup() {
  // initialize pin 13 as an output.
  pinMode(led, OUTPUT);
  pinMode(button, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the state of the sensor
  int state = digitalRead(button);

  // if sensor engaged (button is pressed), turn on LED
  if (state == HIGH) {
    digitalWrite(led, HIGH);
  }
  // else turn off LED
  else {
    digitalWrite(led, LOW);
  }
}
```

When you've entered the sketch as written, you're ready to compile and run it.

Compiling and Uploading

Once you have the sketch written, test the compilation using the Compile button in the upper-left corner of the IDE. Fix any compilation errors that appear in the message window. Typical errors include misspellings or case changes (the compiler is case sensitive) for variables or methods.

After you've fixed any compilation errors, click the Upload button. The IDE compiles the sketch and uploads the compiled sketch to the Arduino board. You can track the progress via the progress bar at the lower right, above the message window. When the compiled sketch is uploaded, the progress bar disappears.

Testing the Project

Once the upload is complete, what do you see on your Arduino? If you've done everything right, the answer is nothing. It's just staring back at you with that one dark LED—almost mockingly. Now, press the pushbutton. Did the LED illuminate? If so, congratulations: you're an Arduino programmer!

If the LED didn't illuminate, hold the button down for a second or two. If that doesn't work, check all of your connections to make sure you're plugged in to the correct runs on the breadboard and that your LED is properly seated with the longer leg connected to the resistor, which is connected to pin 13.

On the other hand, if the LED stays illuminated, try reorienting your pushbutton 90 degrees. You may have set the pushbutton in the wrong orientation.

Try the project a few times until the elation passes. If you're an old hand at Arduino, that may be a short period. If this is all new to you, go ahead and push that button and bask in the glory of having built your first sensor node!

WHAT ABOUT OTHER MICROCONTROLLERS?

Yes, there are other microcontrollers to choose from than the Arduino. Indeed, some predate the Arduino. If you have had experience with these, especially if you already own some of these boards and their accessories, you should consider using them because they offer many of the same benefits as the Arduino.

Some of the more popular microcontroller alternatives include the following. I include a link for more information about each.

- *Esquillo*: This is a relatively new IOT platform that features an onboard web-based development environment for developing in Squirrel featuring an interactive debugger, cloud support, and more. It also supports some Arduino shields. (See `http://esquilo.io/`.)
- *mbed*: This is a new IOT platform solution (currently in beta) that uses an ARM processor. (See `http://mbed.com/en/`.)
- *Photon (formerly Spark)*: This is an IOT development platform sporting WiFi and even an onboard web IDE. Oh, and it is Arduino compatible. (See `http://particle.io`.)
- *Propeller*: This is a powerful processor and wide selection of accessories available. Many older projects use this processor, which is programmed with a C-like programming language (See `http://parallax.com/catalog/microcontrollers/propeller`.)
- *Teensy*: This is a small microcontroller programmed with a special USB-based loader. Since it uses the same family of processors as the Arduino, it can be programmed with the Arduino IDE (See `http://pjrc.com/teensy/`.)

You can find many of these at popular online electronics stores such as Sparkfun (`http://sparkfun.com`) and Adafruit (`http://adafruit.com`).

Now that you've seen a number of Arduino boards, let's discuss some common and popular hardware add-ons for the Arduino.

Additional Arduino Hardware

Recall the Arduino supports add-on hardware via a daughter board that fits onto the Arduino headers (called a shield). There are many such shields available ranging from simple prototyping shields to complex motor controllers. I discuss some of the shields you are likely to use in IOT solutions in this section beginning with the Ethernet shield. There are far more Arduino shields available, but I keep the discussion to these to highlight those I use most for IOT solutions.

Arduino Ethernet Shield

The Arduino Ethernet shield gives your Arduino the capability to connect to a network and ultimately the Internet via an Ethernet network connection. The Arduino Ethernet shield is made by Arduino.cc and is compatible with most Arduino boards including the large Mega.

The board also comes equipped with a MicroSD card reader, making it an excellent choice for projects that need to store or read stored data locally. There is a reset button on the board that duplicates the reset button on the Arduino. Figure 2-17 shows the Arduino Ethernet shield.

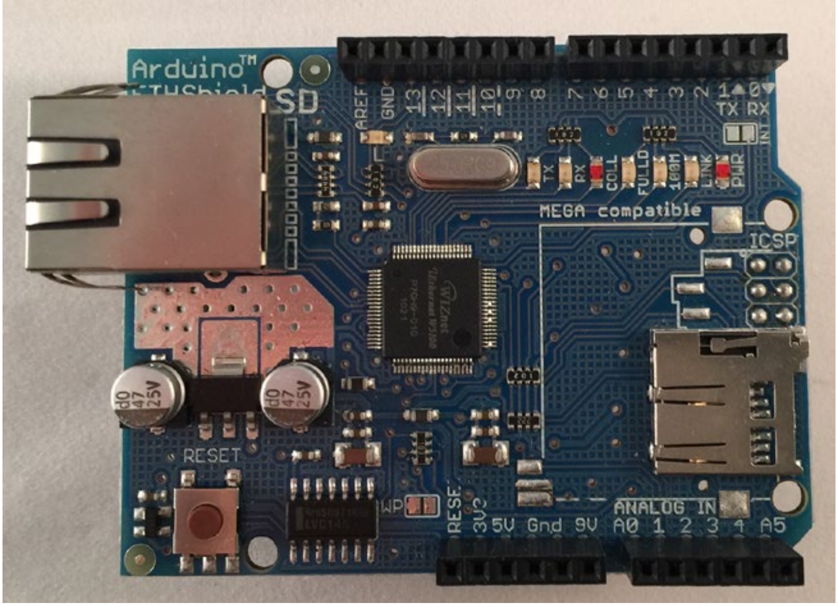


Figure 2-17. Arduino Ethernet shield

The board uses the built-in Ethernet library for interacting with network resources. There are a number example sketches available that you can use to learn how to use the shield as well as excellent documentation (`http://arduino.cc/en/Reference/Ethernet`).

The Arduino Ethernet shield is one of the most reproduced shields. I've found numerous third-party shields available at a fraction of the cost. Some it turns out are not 100 percent compatible, and some don't come with the MicroSD card reader. Interestingly, there is a variant of this board that permits you to use power over Ethernet (POE), allowing you to use the Ethernet cable and a special power supply on your switch or router to supply power to the Arduino.

If you need to make your Arduino solutions network aware and can use an Ethernet connection, the Arduino Ethernet shield is an excellent choice. If you want to save some money, you can choose one of the third-party versions; just make sure it is 100 percent compatible before you buy it.

A WARNING ABOUT ETHERNET SHIELD VARIANTS

You may be thinking that all Ethernet shields are created equal and work like network cards in desktop computers. That is, any one will do; just drop it in and go. But that is far from the case. What makes an Arduino-compatible Ethernet shield is not just that it works with your Arduino; it must also use the built-in Ethernet library.

For example, there are a number of Arduino Ethernet shields and even some Ethernet modules (think breakout boards) that use a different chipset such as the CC3000 or any one of several other chipsets. Many of these are not compatible with the Ethernet library. The way you can tell is if the shield comes with or requires you to download a library to use it.

This may not be a big deal for you for a single project, but there are at least two other concerns for IOT developers. First, if you have many nodes in your solution, you could potentially have to write a different sketch for each type of Ethernet shield used. Second, if you plan to use your Arduino to talk to a MySQL server, you must have a shield that uses the Ethernet library. This is because the special library designed to talk to MySQL (called the MySQL Connector/Arduino) is written for the Ethernet library. Using a different library that does not adhere to the Ethernet.Client class may make it incompatible with the connector. You will learn more about the connector in Chapter 6.

Arduino WiFi Shield 101

The Arduino WiFi shield 101 is the latest shield from Arduino.cc for connecting an Arduino to the Internet via a WiFi signal. There are older WiFi shields available, and you will see some of those in the following sections. What makes this shield interesting and new is it has onboard crypto-authentication, making it much easier to use on wireless networks using a more secure authentication supporting both the WEP and WPA2 Security Enterprise protocols. Because of these features, it is marketed for IOT solutions. However, unlike the older Arduino WiFi shield, it does not come with a MicroSD reader. Figure 2-18 shows the Arduino WiFi shield 101.

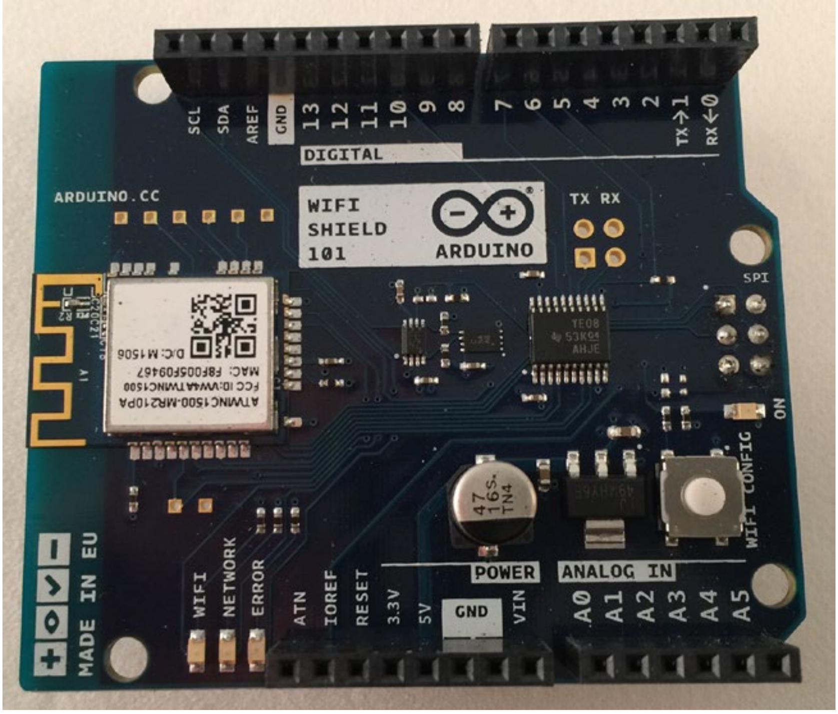


Figure 2-18. *Arduino WiFi shield 101*

There is one drawback, and it depends on how you will use the shield. Like some of the Arduino Ethernet shield clones (see the "A Warning About Ethernet Shield Variants" sidebar), it does not use the Ethernet library. You must download a new library to use it (`http://arduino.cc/en/Reference/WiFi101`). This may not be an issue if you have only a few nodes, but if you want to use the shield with existing sketches or sketches written for the Ethernet library, you may need to make some modifications to get it to work.

Fortunately, the new library has many examples you can use to write your sketches. I have not used the shield in my projects yet, but initial experimentation shows the new library is similar to the Ethernet library and thus should not be difficult to use in place of an Arduino Ethernet shield.

Arduino WiFi Shield

The Arduino WiFi shield is the older version of the newest 101 shield. It supports only simple authentication (no crypto features) but comes complete with a MicroSD card reader. What I like most about the shield is it is 100 percent compatible with the Ethernet library, making it easy to use the same core code in sketches for both the Arduino Ethernet and WiFi shields. There is a bit of a different startup code, but all your calls to the Ethernet library and its classes remain unchanged. Figure 2-19 shows the Arduino Ethernet shield.

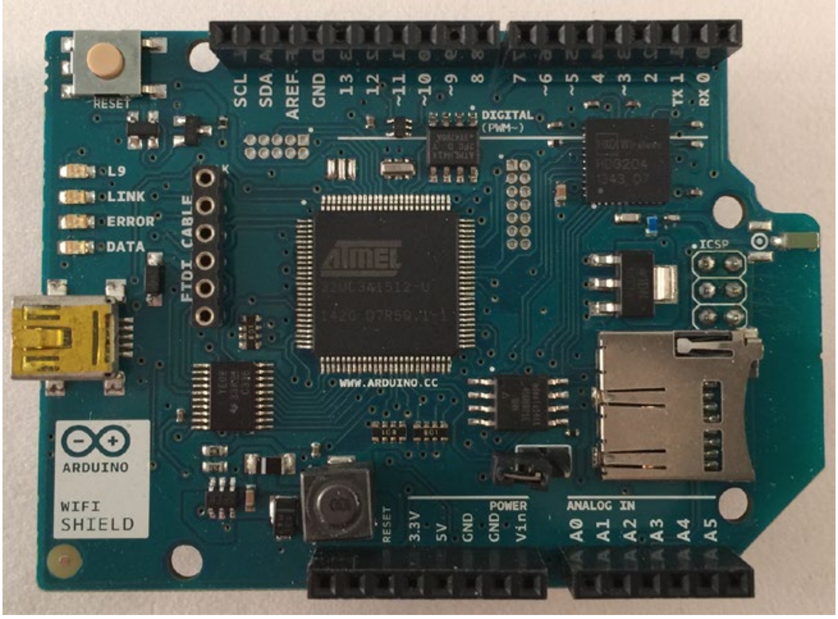


Figure 2-19. *Arduino*

Sparkfun WiFi Shield: ESP8266

The Sparkfun WiFi shield, ESP8266, is an interesting board from Sparkfun (`http://sparkfun.com/products/13287`). It has the ESP8266 WiFi system on chip (SoC) processor on board, giving a split personality. Recall from the earlier section on Arduino boards, the ESP8266 WiFi SoC is a programmable module, but by placing this chip on an Arduino shield layout, it permits you to use the shield as a simple WiFi connection using the AT command set with an Arduino board. Figure 2-20 shows the Sparkfun WiFi shield, ESP8266.

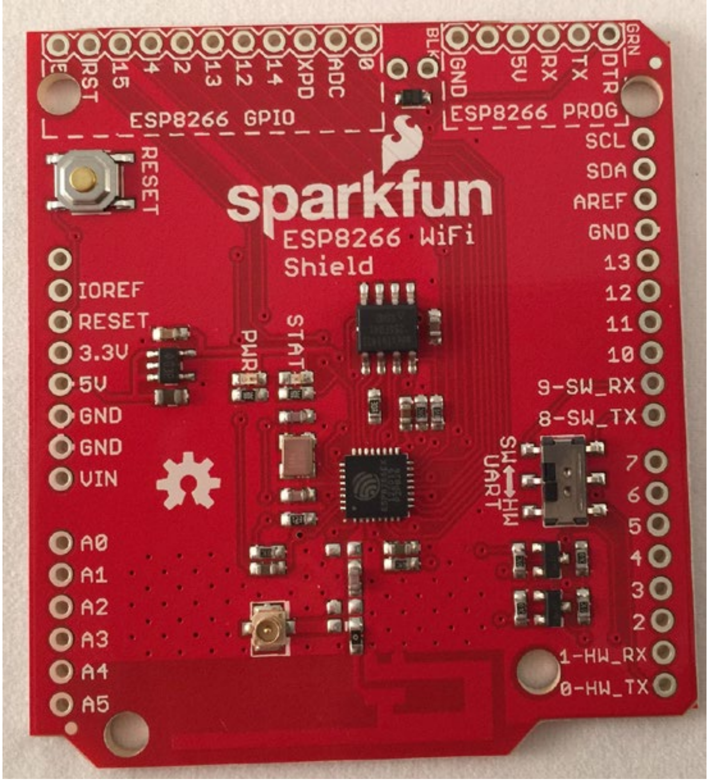


Figure 2-20. *Sparkfun WiFi shield, ESP8266*

While you can use this shield as a WiFi gateway for your Arduino, it is not programmed the same way. Instead of using the Ethernet library, it has its own library that you must download and use. This library (`https://github.com/sparkfun/SparkFun_ESP8266_AT_Arduino_Library`) allows you to use methods to access the AT command set of the ESP8266 to connect to the network. Fortunately, the methods are similar to the Ethernet library but not compatible. Thus, you may need to rewrite the networking portions of your existing sketches to use the shield.

The ESP8266 WiFi SoC comes programmed to work with the Arduino as a WiFi shield. However, it can be reprogrammed. Notice the ESP8266 connections at the top of the board. You can use these to program the ESP8266. All you need to do is solder a pin header on the board and connect it via an FTDI cable. This allows you to experiment with the ESP8266 chip by modifying it to add additional commands or features, making this shield a middle ground between an Arduino and the ESP8266 breakout board.

To learn more about this special board, see `https://learn.sparkfun.com/tutorials/esp8266-wifi-shield-hookup-guide`.

Adafruit WiFi Shield

The Adafruit WiFi shield is one of the few Arduino Ethernet clone boards that I've found to be a good alternative. The Adafruit WiFi shield uses the CC3000 chipset, which requires a new library. Fortunately, Adafruit has an excellent tutorial for using the library (`https://learn.adafruit.com/adafruit-shield-compatibility/cc3000-wifi-shield`).

The shield also has a MicroSD reader, making it a good choice for me since many of my Arduino nodes use the SD card to store data for logging and as a backup for database storage. Figure 2-21 shows the Adafruit WiFi shield.

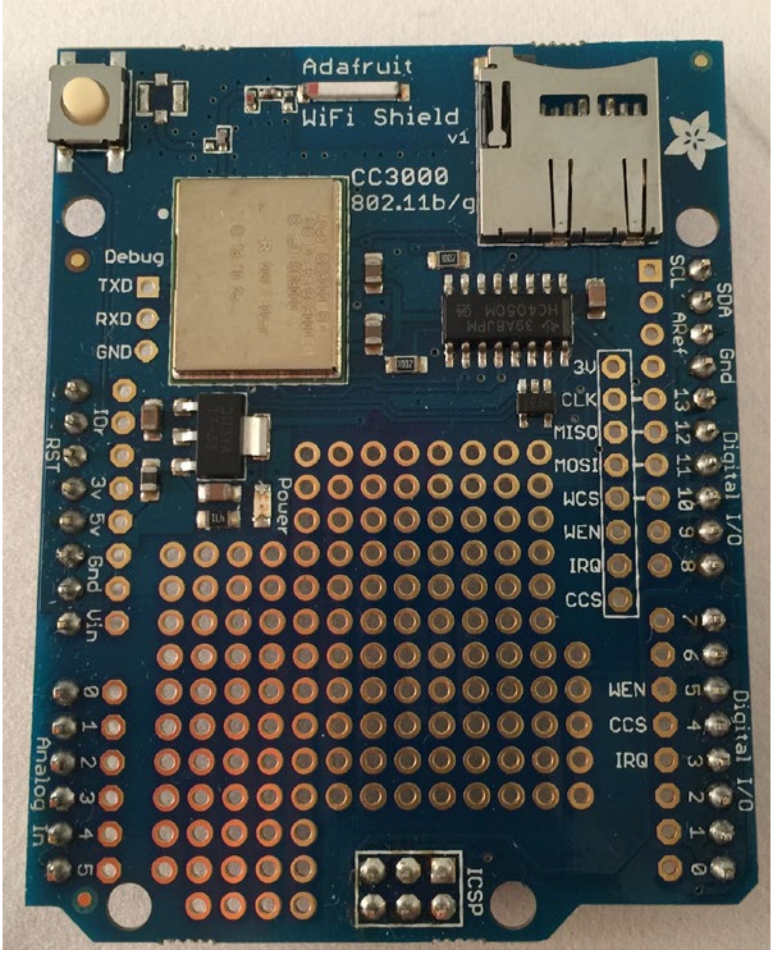


Figure 2-21. *Arduino*

Unfortunately, this shield is not compatible with the Ethernet library, which can mean rewriting the networking portion of your existing sketches. This makes it less attractive for me when developing IOT nodes because I like to use Ethernet connections when developing my sketches to eliminate the delay inherent in slower WiFi connections.

Sparkfun CryptoShield

This shield is an interesting shield. It is the first I’ve found that includes advanced cryptography features such as a real-time clock (RTC) module to keep accurate time, a trusted platform module (TPM) for RSA encryption/decryption, an AES-128 encrypted EEPROM for storing small amounts of encrypted data (such as user IDs and passwords for servers), support for SHA-256 and HMAC-256, and an ATECC108 that performs the Elliptic Curve Digital Signature Algorithm (ECDSA). Wow! That's a lot of cool stuff. Figure 2-22 shows the Sparkfun CryptoShield.

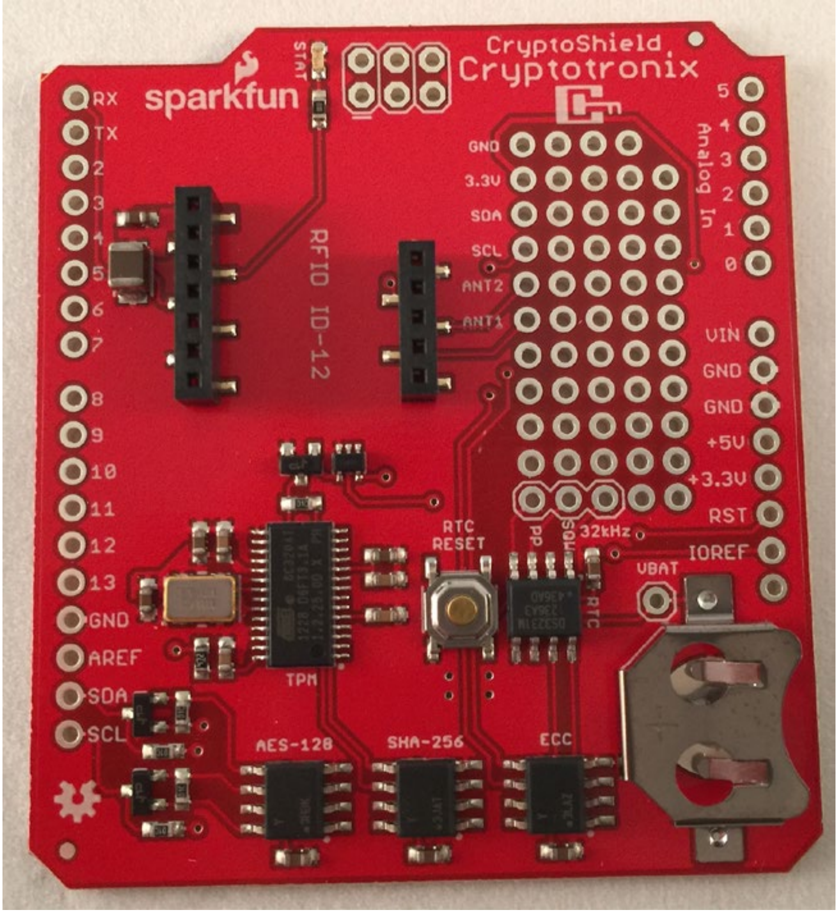


Figure 2-22. *Sparkfun CryptoShield for Arduino*

You can use this shield to encrypt data before you send it out and unencrypt it upon receipt. This may sound like overkill, but consider for the moment cases where you have nodes in your solution that must send data wirelessly or across unsecured network connections. In this case, adding hardware-level encryption means you can secure your data even if the network is not secure.

■ **Note** Sparkfun also makes a version of this board for the BeagleBone Black.

I've used this shield with great success in securing data from one Arduino node to another. I am planning to use this shield in my IOT solutions to help secure them from surveillance or theft.

Sparkfun MicroSD Shield

The last shield in my list of example Arduino shields is the Sparkfun MicroSD shield. It is simply a shield with a MicroSD card reader. There are other variants of this shield available, but I like the Sparkfun version because it also comes with a prototyping area, making it possible to build your own circuit on the shield and giving your solution a convenient platform and a MicroSD reader to boot. Figure 2-23 shows the Sparkfun MicroSD shield.

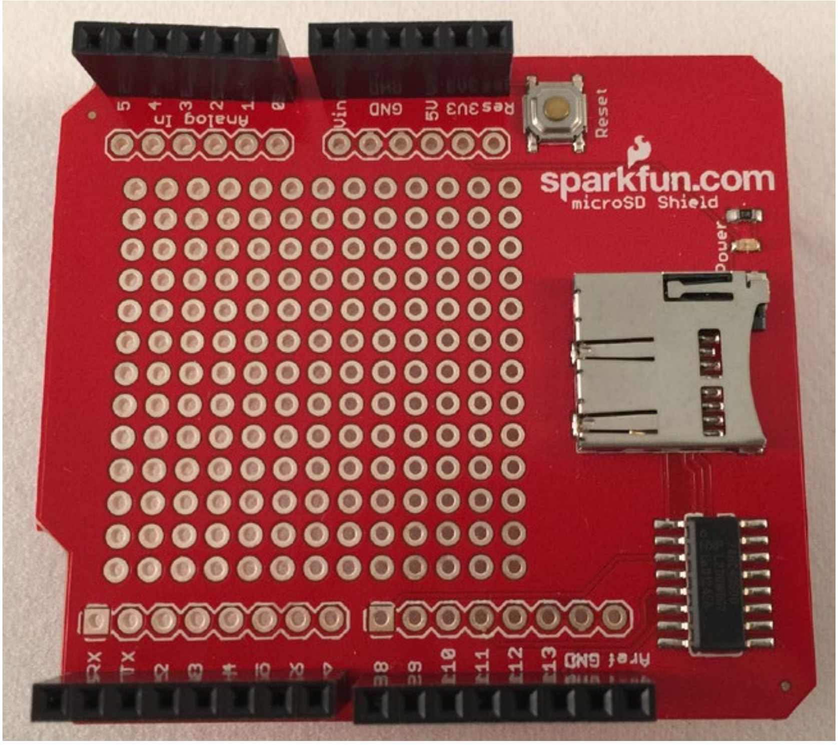


Figure 2-23. *Arduino microSD shield from Sparkfun*

XBee Modules

The next hardware option I will discuss is not specifically an Arduino shield or even meant to be used exclusively with the Arduino. As you will see, the XBee is a versatile module used for low-cost, low-power (as in resources) communication.

The XBee is a self-contained module that uses radio frequency (RF) to exchange data from one XBee module to another. XBee modules transmit on 2.4GHz or long-range 900MHz and have their own network protocols.

The XBee module itself is small—about the size of a large postage stamp—making it easy to incorporate into small projects such as sensor nodes. The modules are also low power and can use a special sleep mode to further reduce power consumption.

Although the XBee isn't a microcontroller, it does have a limited amount of processing power that you can use to control the module. One of these features, the sleep mode, can help extend battery life for battery-powered (or solar-powered) sensor nodes. You can also instruct the XBee module to monitor its data pins and transmit the data read to another XBee module. Aha! So, you can use XBee modules to link a sensor node to a data-aggregator node.

Although the XBee can be used to read sensor data, its limited processing power may mean it isn't suitable for all sensor nodes. For example, sensors that require algorithms to interpret or extrapolate meaningful data may not be suited for using an XBee alone. You may need to use a microcontroller or computer to perform the additional calculations.

Tip I include a more complete explanation of XBee modules in my book *Beginning Sensor Networks with Arduino and Raspberry Pi* (Apress, 2013). If you want to know more about the XBee hardware and specifications, see Chapter 2 in that book.

Several XBee modules are available. There are normal models and Pro models with more power and capabilities. There is also a wide variety of antenna options from onboard chip antennas to wire whip antennas to remote antenna connections. Figure 2-24 shows several of the XBee modules I use on a regular basis.

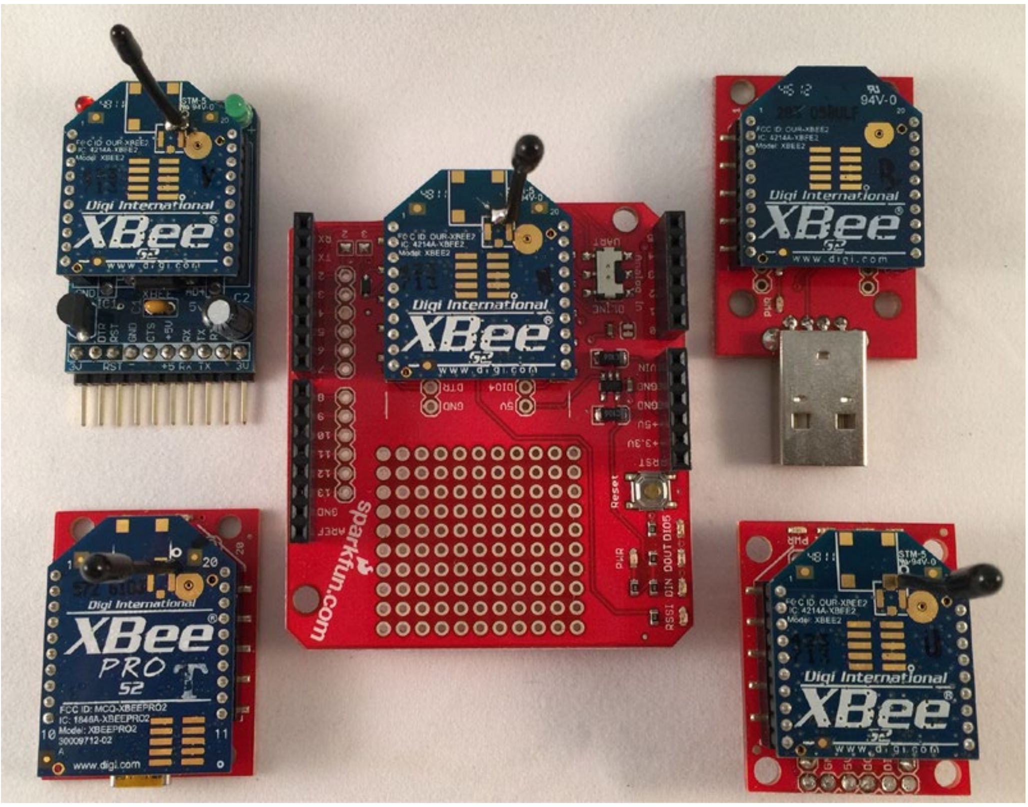


Figure 2-24. *XBee modules and various host boards*

Notice each of the XBee modules (five in all) is mounted on a different small board. These are examples of the types of hosts you can use to program and run your XBee modules. In the center is an XBee shield from Sparkfun (`https://www.sparkfun.com/products/12847`) that allows you to connect the XBee directly to your Arduino. The rest of the boards are those you can use to program the XBee. Starting from the upper-left corner, you can see the following boards:

- *Adafruit XBee Adapter Kit*: Provides an FTDI interface to programming the XBee (`www.adafruit.com/products/126`).
- *Sparkfun XBee Explorer USB Dongle*: Allows you to plug the XBee directly into a USB port (`https://www.sparkfun.com/products/11697`).
- *Sparkfun XBee Explorer Regulated*: Provides a power-regulated breakout board (XBees run on only 3.3V, not 5V), making it possible to use this board with other discrete components such as sensors (`https://www.sparkfun.com/products/11373`).
- *Sparkfun XBee Explorer USB*: Permits you to use a USB cable with a USB cable. It also has breakout pins for the XBee, making it possible to use this board with other discrete components such as sensors (`https://www.sparkfun.com/products/11812`).

You can find XBee radios for sale at most online electronics stores such as Sparkfun and Adafruit. Aside from my sensor networks book, Adafruit (`https://learn.adafruit.com/xbee-radios`) and Sparkfun (`https://learn.sparkfun.com/tutorials/xbee-shield-hookup-guide`) have excellent tutorials on using the XBee.

Now that you've learned about the popular Arduino microcontroller, its development environment, and additional hardware (shields), let's now discover more about low-powered (also called *low-cost*) computing platforms.

Low-Powered Computing Platforms

Low-powered computing platforms, sometimes called low-cost *computer boards* or *mini-computers*,[7] are built from inexpensive components designed to run a low-resource-intensive operating system. Most boards have all the normal features you would expect from a low-cost computer including video, USB, and networking features. However, not all boards have all these features.

The reason they are sometimes called low power isn't because of their smaller CPUs or memory capabilities; rather, it is because of their power requirements, which are typically between 5V and 24V. Since they do not require a massive, PC-like power supply, these boards can be used in projects that need the capabilities of a computer with a real operating system but do not have space for a full-sized computer, cannot devote the cost of a computer, or must run on a lower voltage.

■ **Tip** Because of the ambiguity and seeming lack of a standard nomenclature, I will refer to these boards as *low-cost computing boards* or simply *boards* when discussing them in this category.

[7] *Mini-computers* is not a very good name because some of these boards do not include video controllers or support for common laptop or desktop computer peripherals.

There are many varieties of low-cost computing boards. Some support the full features of a typical computer (and can be used as a pretty decent laptop alternative), while others have the bare essentials to make them usable as embedded computers. For example, some boards permit you to connect a network cable, keyboard, mouse, and monitor for use as a normal laptop or desktop computer while others have only networking and USB interfaces, requiring you to remotely access them.

One of the most powerful aspects of these boards is since they run a variant of the Linux operating system, many can be programmed with C, C++, Perl, Python, and similar programming languages. This makes it possible for you to develop your software with tools that you would find on any desktop and yet deploy it in your solution on a small, embedded computing device. Not only that, but also given the processors used are faster and some even have multiprocessing capabilities, your software will run much faster than it would on a microcontroller. How cool is that?

I have several of these boards[8] and have used them in a variety of ways. What I find most interesting about the boards I've used is some have support for Arduino shields (similar to the Arduino Yún), permitting you to mount and access an Arduino shield and connected circuitry running right on the board. I call these boards *Arduino hybrids*. I group the remaining boards into the computer boards category since they are largely full-featured computers. I discuss several examples of each category in the following sections.

Arduino Hybrids

An Arduino hybrid board is a microcontroller or a low-cost computing platform that contains Arduino headers can be used as a gateway in a sensor or IOT network where you can remotely log into it and yet access Arduino shields as if the board were an Arduino. These boards can be used in many more ways, but they excel at this role. You can set up the board on your network to give you remote access (via another computer or terminal) to the computer side while also connecting to and using the resources of the Arduino shields.

Some Arduino hybrid boards provide an onboard Arduino-compatible processor so that you can use Arduino sketches that interact with programs on the computer. These boards are a bit harder to use in the sense they don't usually support all Arduino shields, but they can be powerful when programmed using a language such as Python.

■ **Note** Some boards, such as the Raspberry Pi, can be expanded with special add-ons (such as the Raspberry Pi AlaMode[9]) that provide similar features as the Arduino hybrids.

I present two Arduino hybrid boards: the popular pcDuino3B and the Intel Galileo.

pcDuino3B

The pcDuino3B is one the more powerful boards I've used. It is effectively a full-featured small desktop computer. In fact, I've used my pcDuino3B as a laptop alternative simply by plugging in a mouse, keyboard, and monitor. Not only can you connect via WiFi to the Internet, but the board runs an older, specialized version of Ubuntu 12.04, which provides a full-featured desktop and many of the same applications you would use on a laptop or desktop. Except that there is a bare board sitting on the table, the speed of the processor makes the experience much like a laptop.

[8] As you will see, I have an enthusiast's passion. There's always room for one more board!

[9] http://makershed.com/products/alamode-for-raspberry-pi

The board has a powerful A20 processor (much faster than some other boards) with 1Gb of memory as well as an HDMI interface supporting an amazing 1080p resolution with its onboard video processor. It also has WiFi, Ethernet, audio, USB, a MicroSD drive, and even a SATA drive controller that permits the addition of spindle or solid-state hard drives for storage. Indeed, the feature list of this board is quite impressive.

As given the category, the board is compatible with the popular Arduino ecosystem such as Arduino shields. It also has its own header of I/O pins with 14 general-purpose I/O pins (GPIO for short), 2 PWM, 6 ADC, and 1 each of UART, SPI, and I2C pins. Figure 2-25 shows the latest pcDuino3B board.

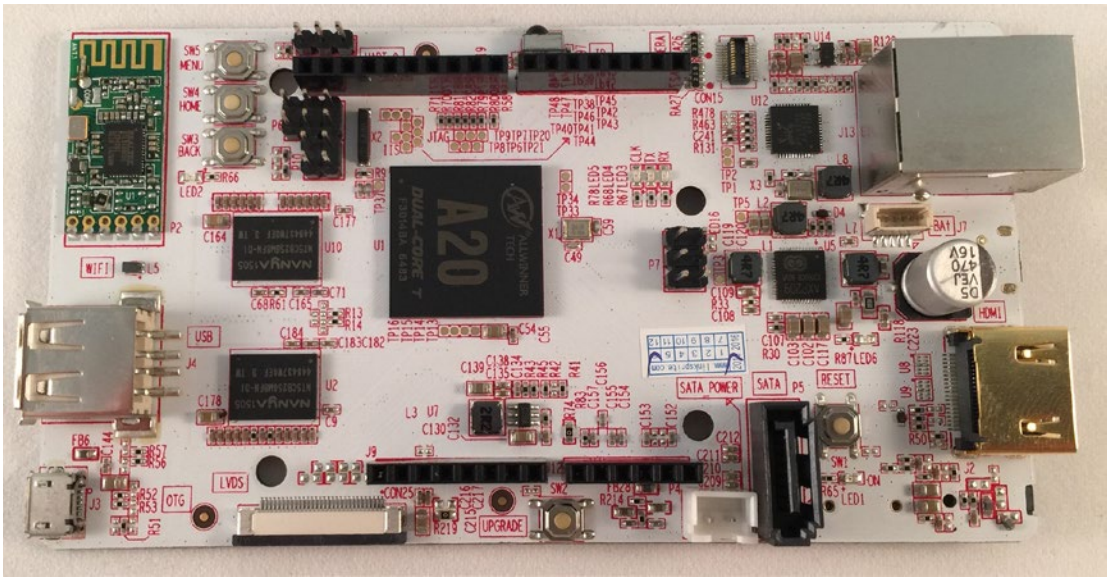


Figure 2-25. *pcDuino3B*

■ **Note** The pcDuino3B supports only Arduino Uno R3 3.3v shields.

I have often used the pcDuino3B as a traveling Arduino lab. Given its powerful computing features, the fact that the operating system is run from onboard memory storage and I can run the Arduino IDE from the desktop permits me to interact with the Arduino shields and hardware easily without having to lug around a separate computer, cable, Arduino, and so on. Best of all, it can be powered by USB.

I have also used the pcDuino3B as a database server (see Chapter 5) and a web server. Since it runs Ubuntu, there is little this small board cannot do. Perhaps the only downside is that it costs about twice as much as some of the other boards (but not nearly as much as the Intel boards). Thus, if you had to decide between this board and a cheaper board, it may come down to whether you need the power of the pcDuino3B versus the lower cost (and fewer features).

For more information and the full specifications of the pcDuino3B, see `http://linksprite.com/wiki/index.php5?title=PcDuino3B`.

Intel Galileo Gen 2

The Intel Galileo Gen 2 is based on the Intel Quark SoC X1000 application processor, which is a 32-bit CPU based on the venerable Intel Pentium CPU packaged as a SoC. It also has Arduino headers for use with Arduino Uno R3 shields (3.3V and some 5V shields).

Perhaps its most interesting feature is that since the CPU is an Intel chip, the board can run several operating systems (but not simultaneously) such as the new Windows 10 IOT Core[10] and most Linux variants. It comes with Yocto Linux (`http://yoctoproject.org/`) bootable from onboard memory but can be upgraded easily.

■ **Note** Yocto Linux does not currently support as wide an array of software tools as the Ubuntu on the pcDuino. However, more packages are being added to Yocto, so chances are if you cannot find the package on Yocto you're looking for, it may be added soon.

The board has a lot of features such as USB, Ethernet, an RTC, MicroSD (used for running the operating system but unlike the pcDuino3B has a limited onboard bootable operating system), 8MB flash memory, and a mini-PCI Express slot, which promises expansion unlike any other board (but yet to be realized in the mainstream). Figure 2-26 shows the Intel Galileo Gen 2 board.

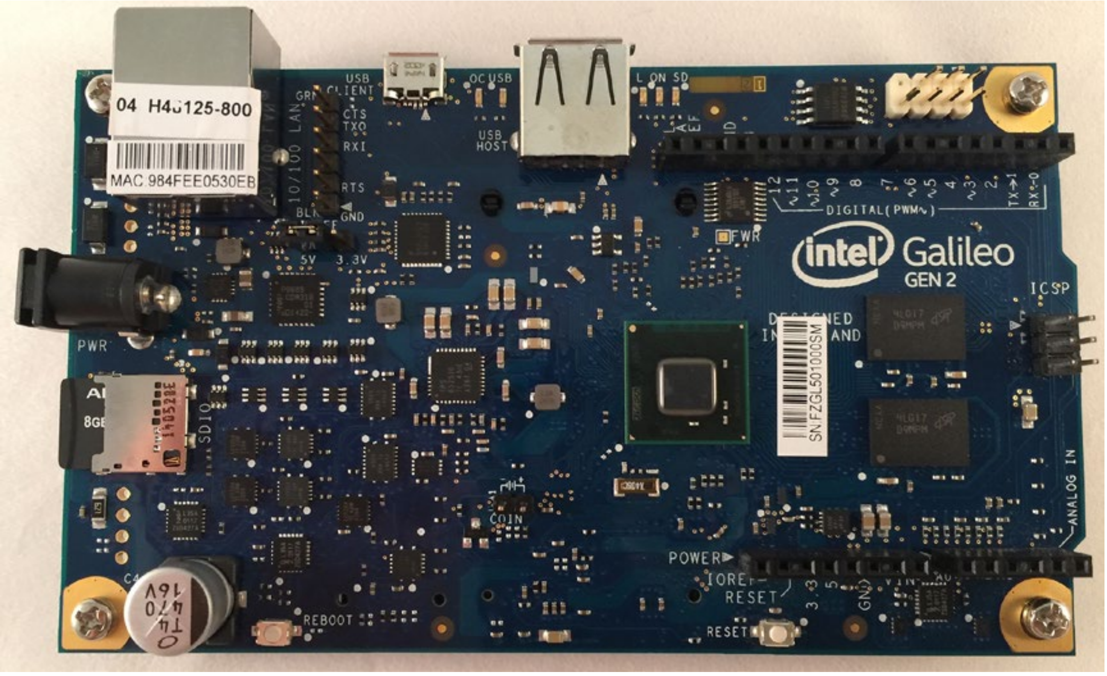


Figure 2-26. *Intel Galileo Gen 2*

[10] At this time, the Windows 10 IOT kit has not been officially released, but beta versions have proven interesting and promising.

The board requires external power supply and thus cannot be powered by USB but has support for power over Ethernet. While it has many interesting features, it is a relatively new addition to the market and does not seem to be catching on in popularity as much as some other boards. That said, there are several IOT expansion or experiment kits available such as the kit from SeeedStudio (http://seeedstudio.com/depot/Grove-starter-kit-plus-Intel-IoT-Edition-for-Intel-Galileo-Gen-2-and-Edison-p-1978.html).

Given the operating system is not as powerful as a full Linux implementation, you are most likely going to want to create a boot image and run a newer version of the operating system from the SD drive. You can find complete documentation on this endeavor on Intel's IOT site (https://software.intel.com/en-us/programming-blank-sd-card-with-yocto-linux-image-linux).

To date, I've used my Intel Galileo Gen 2 as a gateway for some of my Arduino shield–based circuits. This is because the board allows me to program it much like I would a normal Arduino with a special version of the Arduino IDE from Intel (http://intel.com/support/galileo/sb/CS-035101.htm). In addition, I've used the Galileo in place of an Arduino where I need more computing power or wanted easy access to the Arduino hardware from a remote computer (one I can remotely log into).

For more information about the Intel Galileo Gen 2, see https://software.intel.com/en-us/iot/hardware/galileo.

Computer Boards

The second category of low-cost computing boards are those that are intended to be embedded computing systems and do not include Arduino hardware compatibility (no shield headers or Arduino-compatible microcontroller on board).[11] However, all the boards I've used in this category support a variety of hardware pins you can use to connect sensors and other components and circuits. Thus, they are still good IOT development platforms. They just aren't Arduino hybrids.

That doesn't mean they aren't less powerful than Arduino hybrid boards (with the possible exception of the pcDuino3B); rather, they are generally more powerful and more versatile than Arduino hybrid boards. By virtue of their computer-like feel when using them, you may be surprised by their prowess.

The physical size of these boards is another characteristic that may take some getting used to. While most are very small, their size requires using both sides of the PCB for mounting components. Thus, most boards have connectors and components on both the top and the bottom. This isn't a big deal but may require some careful arranging if you are planning to mount these boards in an enclosure or may require some careful handling if you use them naked (just the bare board[12]). Fortunately, you can find custom-fit cases for most of these boards including several 3D-printable ones from http://thingiverse.com.

But they aren't just computers. Most provide GPIO pins for connecting sensors, LCDs, circuits, and more. In this respect, they are like the Arduino and can be used in many of the same situations. What makes them different is the greater processing power and connectivity that the computer aspects provide. That is, you can use a Raspberry Pi in place of an Arduino and gain the ability to remotely access that node and interact with it, which is difficult to do on the Arduino.

Boards in this category are typically computers that have dedicated hardware support for expansion, have more powerful operating system support (again, except for the pcDuino), and have networking capabilities built in. The most popular board in this category is the Raspberry Pi, which is featured predominantly later in this book. However, as you will see, there are other alternatives that are equally powerful. Let's start with a great alternative to the Raspberry Pi—the BeagleBone Black.

[11] The Intel Edison is a minor exception when used with the Arduino breakout host board.

[12] Cover that thing up, there may be children nearby!

BeagleBone Black

The BeagleBone Black is a small board designed for developers and hobbyists to experiment with hardware in embedded solutions using a Linux-based experience. In fact, the BeagleBone Black is designed to boot its Linux operating system in as little as ten seconds. A growing community of supporters are adding daily to the knowledge base and forum. Perhaps one of the best things about the BeagleBone Black is the startup guide at http://beagleboard.org/getting-started. It was written to get you up and running using your new "bone" in as little as five minutes. Thus, this board has a lot of appeal for someone familiar with Linux who wants to get started quickly. Figure 2-27 shows the BeagleBone Black.

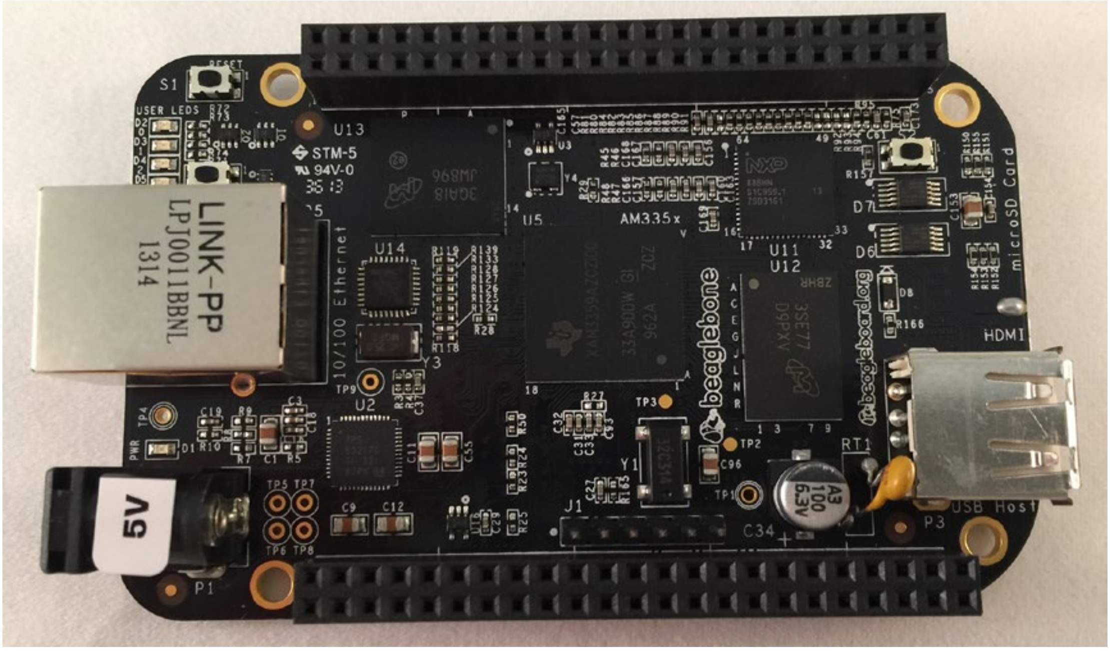


Figure 2-27. *BeagleBone Black*

The board is also a capable low-cost computer with an A8 ARM processor, 512MB RAM, 4Gb of flash storage, USB host and client (for power and terminal access), a graphics chip with HDMI output, and Ethernet. There is also a power connector for sustained operation.

The footprint of the BeagleBone Black is rather unique. It can fit inside a larger metal tin like some breath mints are packaged. You would need to take a pair of tin snips to cut access for the USB, power, and Ethernet, but it will fit. In other words, it's a bit smaller than most boards.

It also has two rows of 46-pin GPIO headers for use in connecting hardware such as sensors, LCD panels, and more. In fact, you would use these headers in a similar way as you would on an Arduino or Raspberry Pi. You can also connect daughter boards called *capes* for added functionality. There are several capes available from vendors like Sparkfun, which offers the following capes:

- *4.3" LCD panel*: http://sparkfun.com/products/12085
- *7" LCD panel*: http://sparkfun.com/products/12086
- *ProtoCape for prototyping circuits*: http://sparkfun.com/products/12774
- *CryptoCape for hardware encryption*: http://sparkfun.com/products/12773

I've found the BeagleBone Black to be a capable board that can be used almost anywhere you need a more powerful processor or need to do additional processing that exceeds the capabilities of a microcontroller-based solution. I've used the BeagleBone Black in a number of experiments with hardware and find it a viable alternative to the more popular Raspberry Pi. However, in some ways the Linux feel and fit of the BeagleBone Black appeals to me more, but that is a personal choice. That said, I still have way more Raspberry Pi boards than bones.

You can learn more about the BeagleBone Black hardware including compatible hardware accessories, creating a bootable Linux image, and more at `http://elinux.org/Beagleboard:BeagleBoneBlack`.

Raspberry Pi 2 Model B

The Raspberry Pi 2 Model B is the latest iteration of the Raspberry Pi. It has all the features of the original Raspberry Pi but with a faster processor and more USB ports. The Raspberry Pi is a popular board with developers mainly because of its low cost and ease of use. Given the popularity of the Raspberry Pi, I cover it greater detail in Chapter 6, including a short tutorial on how to get started using it. Thus, I will briefly cover the highlights here and reserve a more detailed discussion on using the board for Chapter 6.

The Raspberry Pi 2B hardware includes a 900MHz A7 ARM CPU, 1GB RAM, video graphics with HDMI output, 4 USB ports (up from just 2 on older boards), Ethernet, a camera interface (CSI), a display interface (DSI), a MicroSD card, and 40 GPIO pins. Figure 2-28 shows the Raspberry Pi 2B board.

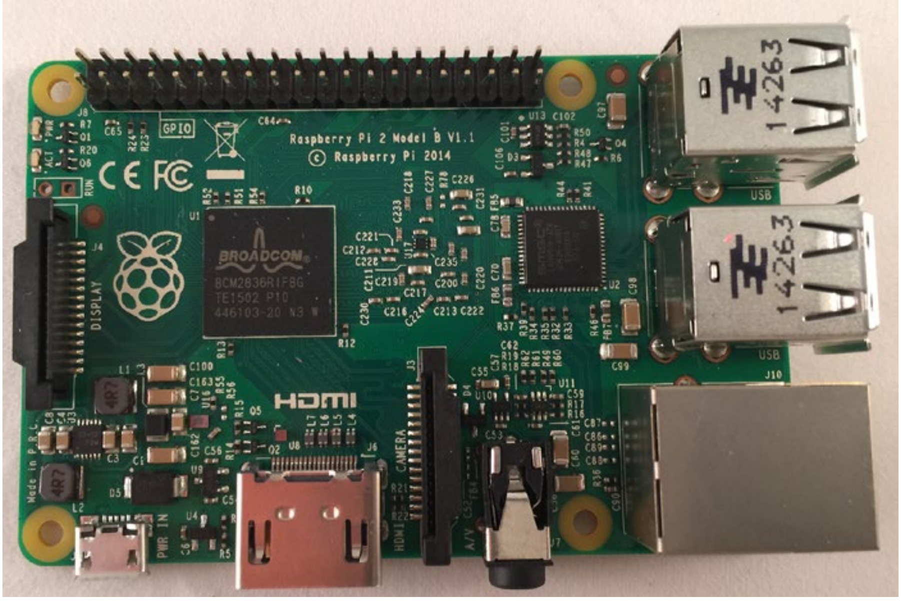


Figure 2-28. *Raspberry Pi Model 2B*

The camera interface is really interesting. You can buy a camera module like the ones at Adafruit (`http://adafruit.com/categories/177`) and connect it to the board for use as a remote video-monitoring component. I've used this feature extensively by turning a couple of my Raspberry Pi boards into 3D-printing hubs where I can send print jobs over the network to print and check the progress of the prints remotely. The software that makes this possible is called OctoPrint (`http://octoprint.org/`), and I cover it in great detail in my book *Maintaining and Troubleshooting Your 3D Printer* (Apress, 2014). See `http://apress.com/9781430268093?gtmf=s` for more details.

The LCD interface is also interesting because there is now a 7" LCD touch panel that connects to the DSI port (http://element14.com/community/docs/DOC-78156/l/raspberry-pi-7-touchscreen-display). I have yet to acquire one of these (they are in short supply), but I have some really neat ideas for a wall-mounted console for monitoring my IOT solutions. I have also seen a number of interesting Raspberry Pi tablets built using the new LCD touch panel. You can learn about one promising example (made by Adafruit, so I expect it to be excellent) at http://thingiverse.com/thing:1082431.[13]

Aside from that, the Raspberry Pi has been my go-to board for all manner of requirements, from a more powerful sensor node to a data aggregation node to hosting a database and web server. There are also many examples from the community on how to employ the Raspberry Pi in your projects. For more information about the Raspberry Pi, see Chapter 6.

Raspberry Pi B

The Raspberry Pi B model is an older version of the Pi 2B discussed earlier with fewer features. I include it in this list because this board is plentiful and can sometimes be found at a discount compared to the newer Raspberry Pi boards. Even so, it has 512MB of RAM, two USB ports, and an Ethernet port. As a plus, accessories such as cases are also plentiful and cheaper than those for the newer models.

I have found only a few cases where I needed the more powerful Pi 2B. Thus, if you can find some of these older boards, you can save some money (that is, if you do not need the extra USB ports or other features of the P2i 2B). Figure 2-29 shows an example of an earlier version of the Raspberry Pi B.

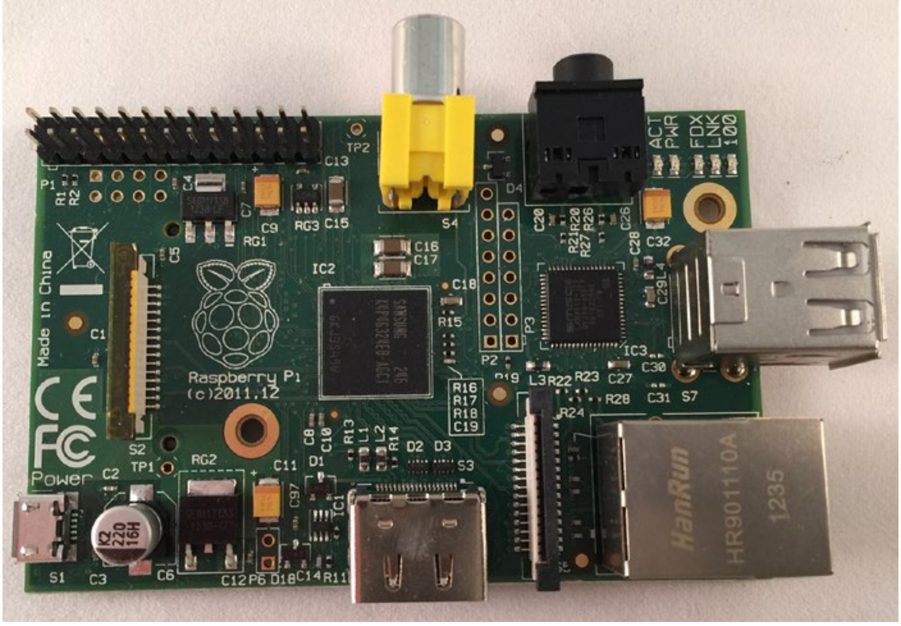


Figure 2-29. *Raspberry Pi Model B*

[13] The case is a 3D-printed affair, so you'll need to find someone to print it for you if you do not have a 3D printer. A full tutorial on how to build it is included.

I often use my older Raspberry Pi B boards (I have too many it seems) whenever I need the power of the Raspberry Pi as an embedded computing node but do not need the extra features for additional USB devices, especially for cases where I can use Ethernet instead of WiFi.

I have also used this older board to prototype projects since the risk (cost) of damaging the older board is not as great as the newer boards. If I fry the older board, I just pull another one out or order another couple of used ones. If I were to damage my newest Raspberry Pi 2B, I'm out a bit more money and will likely have to wait for a new one (even today they can sometimes be hard to find).

For more information about the Raspberry Pi B, see https://www.raspberrypi.org/products/model-b/.

Intel Edison (with Sparkfun Blocks)

The Intel Edison is another low-cost computing board, but instead of being a tiny computer, it is more of an embedded solution than the other boards. I include it here for those who need the power of a low-cost computer in as small and as versatile a package as possible. The board is a mere 35×25×4mm in size. The Edison has an Intel Atom 500MHz dual-core, dual-threaded CPU and an Intel Quark 100MHz microcontroller. A large RFID shield hides all its components. The Edison also runs a version of the Linux operating system named Yocto, which is stored in firmware, making for very fast bootup.

Unlike the similar board, the Intel Galileo, the Intel Edison uses a single module with a micro board-to-board connector designed to allow you to stack additional boards beneath the Edison—similar to the TinyDuino. And unlike the Galileo, you can use the latest version of the Arduino IDE to program the Edison directly. You can also write programs in C, C++, or Python to run locally and access the hardware GPIO pins.

Since the Intel Edison is an embedded platform, it doesn't have USB, video, and network connections, but it does have WiFi (802.11a/b/g/n) and Bluetooth (4.0 and 2.1 EDR). You get the additional ports by using add-ons boards. After all, it is meant as an embedded IOT solution rather than a computing platform. It is also a lot smaller and therefore easier to encase into remote nodes or smaller devices.

Figure 2-30 shows the Intel Edison together with a number of add-on boards stacked beneath it. As you can see, it makes for a tidy package. Note the raised board with the Intel Edison branding. That's the Edison! Everything else in the photo is an additional component.

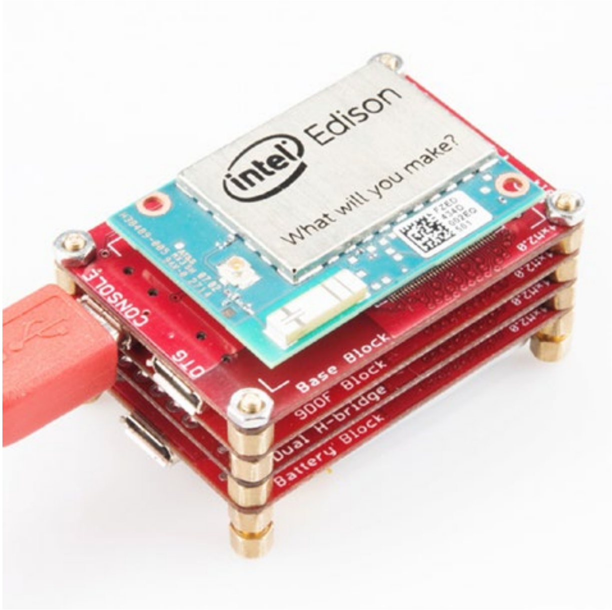


Figure 2-30. *Intel Edison (courtesy of http://sparkfun.com)*

Since the board comes as a SoC module, you will need to buy a host board to use it or connect it to your computer for programming or console access. Intel sells one such board, but there are several other choices from Sparkfun, as described here:

- *Intel Edison Mini Breakout*: Slightly larger than the Edison, the mini breakout provides USB ports for communication via a UART console (think terminal) and power. Intel normally sells this mini breakout board with an Edison module. See `http://sparkfun.com/products/13025`.
- *Intel Arduino Expansion Board*: This is a larger host board with Arduino headers for mounting Arduino shields. It provides the same USB ports as the previous host board and, like that one, is available as a kit. If you want to program the Edison with the Arduino IDE, this host board is a good choice. See `http://sparkfun.com/products/13097`.
- *Sparkfun Base Block*: This is a smaller host board that has the same USB ports but also comes with a pass-through micro board-to-board connector so that you can use additional boards. You can also use this board to program the Edison like an Arduino. This board is available without the Edison from Sparkfun. See `http://sparkfun.com/products/13045`.
- *Sparkfun Console Block*: This is similar to the Base Block but without the power port, thus providing only console access. See `http://sparkfun.com/products/13039`.

The modular nature of the Edison makes it possible to build your solution with only the hardware you need based on a powerful main core. This is especially true with the addition of Sparkfun's Edison Blocks. Sparkfun has created stackable modules (called *blocks*) that you can use to build a hardware stack to meet your embedded computing needs. There are quite a number of blocks available for just about whatever you want to do.

In fact, there are blocks that break out the GPIO pins in either Raspberry Pi or breadboard layouts, supply battery power, and provide a tiny OLED-based display; there's even an Arduino header block. The really cool aspect of using blocks is how they stack, which allows you to not only tailor your hardware for what you need but also to keep the hardware as small as possible to match the diminutive size of the Edison. A sample list of Sparkfun's Edison blocks is shown here:

- *Intel Edison Block ADC*: `http://sparkfun.com/products/13046`
- *Intel Edison Block Base*: `http://sparkfun.com/products/13045`
- *Intel Edison Block Dual H-Bridge*: `http://sparkfun.com/products/13043`
- *Intel Edison Block PWM*: `http://sparkfun.com/products/13042`
- *Intel Edison Block Console Basic*: `http://sparkfun.com/products/13040`
- *Intel Edison Block Console*: `http://sparkfun.com/products/13039`
- *Intel Edison Block GPIO*: `http://sparkfun.com/products/13038`
- *Intel Edison Block Battery*: `http://sparkfun.com/products/13037`
- *Intel Edison Block MicroSD*: `http://sparkfun.com/products/13041`
- *Intel Edison Block Arduino*: `http://sparkfun.com/products/13036`
- *Intel Edison Block OLED*: `http://sparkfun.com/products/13035`
- *Intel Edison Block I2C*: `http://sparkfun.com/products/13034`
- *Intel Edison Block 9 Degrees of Freedom*: `http://sparkfun.com/products/13033`

For more information about the Intel Edison and Sparkfun's Edison blocks, see Sparkfun's excellent guide at https://learn.sparkfun.com/tutorials/edison-getting-started-guide.

JUST HOW LOW-COST ARE THESE BOARDS?

You may be wondering about how much these boards cost. The following are the average prices for each board in the order they were mentioned. I do not include the Sparkfun Edison blocks because, as you will see, the price will depend on which blocks you buy, but generally the blocks range in price from about $15 to $35, with some kits available at a discount. I also omit the older Raspberry Pi Model B, but used prices are somewhat below the $30 mark for those boards.

- *pcDuino3B* : $60.00
- *Intel Galileo Gen 2*: $75.00
- *Raspberry Pi 2B*: $42.00
- *Intel Edison with Mini Breakout*: $75.00
- *Intel Edison with Arduino Host*: $100.00

Notice the more expensive boards are the Intel options. Given it's mostly proprietary hardware, that isn't too surprising. The real bargain is the Raspberry Pi, which partly explains its popularity. However, for the price and features, the pcDuino3B is another of my favorite boards.

Now that you've seen a variety of boards you can use to host your sensors including microcontrollers and low-cost computer boards, let's look at another key hardware component—the sensor.

Sensors

With all this talk of sensors and what sensor networks are and how they communicate data, you may be wondering what exactly sensors are and what makes them sense. This section and its subsections answer those questions and more. Let's begin with the definition of a sensor.

A *sensor* is a device that measures phenomena of the physical world. These phenomena can be things you see, such as light, smoke, water vapor, and so on. They can also be things you feel, like temperature, electricity,[14] water, wind, and so on. Humans have senses that act like sensors, allowing us to experience the world around us. However, there are some things your sensors can't see or feel, such as radiation, radio waves, voltage, and amperage. Upon measuring these phenomena, it's the sensors' job to convey a measurement in the form of either a voltage representation or a number.

There are many forms of sensors. They're typically low-cost devices designed for a single purpose and with a limited capability for processing. Most simple sensors are discrete components; even those that have more sophisticated parts can be treated as separate components. Sensors are either analog or digital and are typically designed to measure only one thing. But an increasing number of sensor modules are designed to measure a set of related phenomena.

[14] Shocking, isn't it?

Analog Sensors

Analog sensors are devices that generate a voltage range, typically between 0V and 5V. An analog-to-digital circuit is needed to convert the voltage to a number. Most microcontrollers have this feature built in, and the Arduino is a fine example. The Arduino has a limited set of pins that operate on analog data and incorporate analog-to-digital (A/D) conversion circuits.

But it isn't that simple (is it ever?). Analog sensors work like resistors and, when connected to microcontrollers, often require another resistor to "pull up" or "pull down" the voltage to avoid spurious changes in voltage known as *floating*. This is because voltage flowing through resistors is continuous in both time and amplitude. Thus, even when the sensor isn't generating a value or measurement, there is still a flow of voltage through the sensor that can cause spurious readings. Your projects require a clear distinction between OFF (zero voltage) and ON (positive voltage). Pull-up and pull-down resistors ensure that you have one of these two states. It's the responsibility of the A/D converter to take the voltage read from the sensor and convert it to a value that can be interpreted as data.

When sampled (when a value is read from a sensor), the voltage read must be interpreted as a value in the range specified for the given sensor. Remember that a value of, say, 2 volts from one analog sensor may not mean the same thing as 2 volts from another analog sensor. Each sensor's data sheet shows you how to interpret these values.

When you use a microcontroller like the Arduino, the A/D converters conveniently change the voltage into a value that uses 10 bits, resulting in an integer value between 0 and 1,023. For example, a sensor may measure phenomena in a range consisting of 200 points on a scale. The lowest value typically represents 0 and the highest 1,023. The Arduino in this case can be programmed to convert the value read from the A/D converter into a value on the sensor's scale.

As you can see, working with analog sensors is a lot more complicated than using the DHT-22 digital sensor from the previous section. With a little practice, you will find that most analog sensors aren't difficult to use once you understand how to attach them to a microcontroller and how to interpret their voltage on the scale in which the sensor is calibrated to work.

Digital Sensors

Digital sensors like the DHT-22 are designed to produce a string of bits using serial transmission (one bit at a time). However, some digital sensors produce data via parallel transmission (one or more bytes[15] at a time). As described previously, the bits are represented as voltage, where high voltage (say, 5 volts) or ON is 1 and low voltage (0 or even -5 volts) or OFF is 0. These sequences of ON and OFF values are called *discrete values* because the sensor is producing one or the other in pulses—it's either ON or OFF.

Digital sensors can be sampled more frequently than analog signals because they generate the data more quickly and because no additional circuitry is needed to read the values (such as A/D converters and logic or software to convert the values to a scale). As a result, digital sensors are generally more accurate and reliable than analog sensors. But the accuracy of a digital sensor is directly proportional to the number of bits it uses for sampling data.

The most common form of digital sensor is the pushbutton or switch. What, a button is a sensor? Why, yes, it's a sensor. Consider for a moment the sensor attached to a window in a home security system. It's a simple switch that is closed when the window is closed and open when the window is open. When the switch is wired into a circuit, the flow of current is constant and unbroken (measuring positive volts using a pull-up resistor) when the window is closed and the switch is closed, but the current is broken (measuring zero volts) when the window and switch is open. This is the most basic of ON and OFF sensors.

[15] This depends on the width of the parallel buffer. An 8-bit buffer can communicate 1 byte at a time, a 16-bit buffer can communicate 2 bytes at a time, and so on.

Most digital sensors are actually small circuits of several components designed to generate digital data. Unlike analog sensors, reading their data is easy because the values can be used directly without conversion (except to other scales or units of measure). Some may suggest this is more difficult than using analog sensors, but that depends on your point of view. An electronics enthusiast would see working with analog sensors as easier, whereas a programmer would think digital sensors are simpler to use.

So, what do you do with the data once it's measured? The following section briefly describes some aspects of sensor data and considerations for storing that data.

Storing Sensor Data

Storing sensor data depends on how the data is interpreted and ultimately how it will be used. If you plan to use a computer—or, better, a database—to store the data, you should store it in a way that makes sense.

For example, storing a sequence of voltages from an analog signal may be considered preserving the data in its purest form, but without context or an A/D converter, the data may be meaningless. Storing the digital conversion of the voltage may not be wise either, because you have to remember the scale and range in order to derive the values intended to be represented. Thus, it makes much more sense to store the resulting conversion to scale. Fortunately, when you're using digital sensors, the only thing you need to remember is what unit of measure is being used (Celsius, Fahrenheit, feet, meters, and so on). Therefore, it's best to save the final form of the measurement.

But where do you store this information? Commercial sensor networks store the data in an embedded database or file-storage device, transmit it to another system for storage, or store it on removable digital media. Older sensor networks (like a polygraph or EKG machine) store the data as hard copy using graphs (making them very obsolete).

There are a number of simple storage devices and technologies you can use to build your own sensor networks, ranging from local devices for the Arduino to modern hard drives on the Raspberry Pi. These storage mechanisms are listed here and discussed in more detail in Chapter 3.

Let's take a look at some of the sensors available and the types of phenomena they measure.

Examples of Sensors

All sensor networks begin with one sensor and a means to read and interpret the data. This chapter has presented a lot of information about sensors. You may be thinking of all manner of useful things you can measure in your home or office or even in your yard or surroundings. You may want to measure the temperature changes in your new sun room, detect when the mail carrier has tossed the latest circular in your mailbox, or perhaps keep a log of how many times your dog uses his doggy door. I hope that by now you can see these are just the beginning when it comes to imagining what you can measure. You should be thinking about what kind of sensor network you want to build; you can use this book as a means to learn how to build it.

What types of sensors are available? The following list describes some of the more popular sensors and what they measure. This is just a sampling of what is available. Perusing the catalogs of online electronics vendors such as Mouser Electronics (`http://mouser.com`), SparkFun Electronics (`sparkfun.com`), and Adafruit Industries (`http://adafruit.com/`) will reveal many more examples.

- *Accelerometers*: These sensors measure motion or movement of the sensor or whatever it's attached to. They're designed to sense motion (velocity, inclination, vibration, and so on) on several axes. Some include gyroscopic features. Most are digital sensors. A Wii Nunchuck (or WiiChuck) contains a sophisticated accelerometer for tracking movement. Aha: now you know the secret of those funny little thingamabobs that came with your Wii.

- *Audio sensors*: Perhaps this is obvious, but microphones are used to measure sound. Most are analog, but some of the better security and surveillance sensors have digital variants for higher compression of transmitted data.
- *Barcode readers*: These sensors are designed to read barcodes. Most often, barcode readers generate digital data representing the numeric equivalent of a barcode. Such sensors are often used in inventory-tracking systems to track equipment through a plant or during transport. They're plentiful, and many are economically priced, enabling you to incorporate them into your own projects.
- *RFID sensors*: Radio frequency identification uses a passive device (sometimes called an *RFID tag*) to communicate data using radio frequencies through electromagnetic induction. For example, an RFID tag can be a credit-card-sized plastic card, a label, or something similar that contains a special antenna, typically in the form of a coil, thin wire, or foil layer that is tuned to a specific frequency. When the tag is placed in close proximity to the reader, the reader emits a radio signal; the tag can use the electromagnet energy to transmit a nonvolatile message embedded in the antenna, in the form of radio signals that are then converted to an alphanumeric string.[16]
- *Biometric sensors*: A sensor that reads fingerprints, irises, or palm prints contains a special sensor designed to recognize patterns. Given the uniqueness inherit in patterns such as fingerprints and palm prints, they make excellent components for a secure access system. Most biometric sensors produce a block of digital data that represents the fingerprint or palm print.
- *Capacitive sensors*: A special application of capacitive sensors, pulse sensors are designed to measure your pulse rate and typically use a fingertip for the sensing site. Special devices known as pulse oximeters (called *pulse-ox* by some medical professionals) measure pulse rate with a capacitive sensor and determine the oxygen content of blood with a light sensor. If you own modern electronic devices, you may have encountered touch-sensitive buttons that use special capacitive sensors to detect touch and pressure.
- *Coin sensors*: This is one of the most unusual types of sensors.[17] These devices are like the coin slots on a typical vending machine. Like their commercial equivalent, they can be calibrated to sense when a certain size of coin is inserted. Although not as sophisticated as commercial units that can distinguish fake coins from real ones, coin sensors can be used to add a new dimension to your projects. Imagine a coin-operated WiFi station. Now, that should keep the kids from spending too much time on the Internet!
- *Current sensors*: These are designed to measure voltage and amperage. Some are designed to measure change, whereas others measure load.
- *Flex/Force sensors*: Resistance sensors measure flexes in a piece of material or the force or impact of pressure on the sensor. Flex sensors may be useful for measuring torsional effects or as a means to measure finger movements (like in a Nintendo Power Glove). Flex-sensor resistance increases when the sensor is flexed.

[16] http://en.wikipedia.org/wiki/Radio-frequency_identification

[17] www.sparkfun.com/products/11719

- *Gas sensors*: There are a great many types of gas sensors. Some measure potentially harmful gases such as LPG and methane and other gases such as hydrogen, oxygen, and so on. Other gas sensors are combined with light sensors to sense smoke or pollutants in the air. The next time you hear that telltale and often annoying low-battery warning beep[18] from your smoke detector, think about what that device contains. Why, it's a sensor node!
- *Light sensors*: Sensors that measure the intensity or lack of light are special types of resistors: light-dependent resistors (LDRs), sometimes called *photo resistors* or *photocells*. Thus, they're analog by nature. If you own a Mac laptop, chances are you've seen a photo resistor in action when your illuminated keyboard turns itself on in low light. Special forms of light sensors can detect other light spectrums such as infrared (as in older TV remotes).
- *Liquid-flow sensors*: These sensors resemble valves and are placed in-line in plumbing systems. They measure the flow of liquid as it passes through. Basic flow sensors use a spinning wheel and a magnet to generate a Hall effect (rapid ON/OFF sequences whose frequency equates to how much water has passed).
- *Liquid-level sensors*: A special resistive solid-state device can be used to measure the relative height of a body of water. One example generates low resistance when the water level is high and higher resistance when the level is low.
- *Location sensors*: Modern smartphones have GPS sensors for sensing location, and of course GPS devices use the GPS technology to help you navigate. Fortunately, GPS sensors are available in low-cost forms, enabling you to add location sensing to your sensor network. GPS sensors generate digital data in the form of longitude and latitude, but some can also sense altitude.
- *Magnetic-stripe readers*: These sensors read data from magnetic stripes (like that on a credit card) and return the digital form of the alphanumeric data (the actual strings).
- *Magnetometers*: These sensors measure orientation via the strength of magnetic fields. A compass is a sensor for finding magnetic north. Some magnetometers offer multiple axes to allow even finer detection of magnetic fields.
- *Proximity sensors*: Often thought of as distance sensors, proximity sensors use infrared or sound waves to detect distance or the range to/from an object. Made popular by low-cost robotics kits, the Parallax Ultrasonic Sensor uses sound waves to measure distance by sensing the amount of time between pulse sent and pulse received (the echo). For approximate distance measuring,[19] it's a simple math problem to convert the time to distance. How cool is that?
- *Radiation sensors*: Among the more serious sensors are those that detect radiation. This can also be electromagnetic radiation (there are sensors for that too), but a Geiger counter uses radiation sensors to detect harmful ionizing. In fact, it's possible to build your own Geiger counter using a sensor and an Arduino (and a few electronic components).

[18] I for one can never tell which detector is beeping, so I replace the batteries in all of them.

[19] Accuracy may depend on environmental variables such as elevation, temperature, and so on.

- *Speed sensors*: Similar to flow sensors, simple speed sensors like those found on many bicycles use a magnet and a reed switch to generate a Hall effect. The frequency combined with the circumference of the wheel can be used to calculate speed and, over time, distance traveled. Yes, a bicycle computer is yet another example of a simple sensor network: the speed sensor on the wheel and fork provides the data for the monitor on your handlebars.
- *Switches and pushbuttons*: These are the most basic of digital sensors used to detect if something is set (ON) or reset (OFF).
- *Tilt switches*: These sensors can detect when a device is tilted one way or another. Although simple, they can be useful for low-cost motion-detection sensors. They are digital and are essentially switches.
- *Touch sensors*: The touch-sensitive membranes formed into keypads, keyboards, pointing devices, and the like are an interesting form of sensor. You can use touch-sensitive devices like these for sensor networks that need to collect data from humans.
- *Video sensors*: As mentioned previously, it’s possible to obtain small video sensors that use cameras and circuitry to capture images and transmit them as digital data.
- *Weather sensors*: Sensors for temperature, barometric pressure, rainfall, humidity, wind speed, and so on, are all classified as weather sensors. Most generate digital data and can be combined to create comprehensive environmental sensor networks. Yes, it’s possible to build your own weather station from about a dozen inexpensive sensors, an Arduino (or a Raspberry Pi), and a bit of programming to interpret and combine the data.

Computer Systems

A complete discussion of IOT hardware (at least hardware for the nodes in the network) would be incomplete without mentioning computer systems. Desktop or server computers can be used in your IOT solutions in a number of ways, from providing a reliable platform for a database server, web server, firewall to the Internet, or cloud gateway (or all of these).

I typically try to avoid using computer systems in my IOT solutions mainly to keep cost down but also because low-cost computer boards and microcontroller boards are more than adequate for the solutions I’ve built (or dream about building).

However, if you do want to use a computer system in your IOT solution, you should do so. Just consider the added cost (even small shoebox computers are many times the cost of a Raspberry Pi), physical size and mounting or location, and additional power requirements (they’re not easy to run off of batteries or solar power).

Finally, make sure you take necessary precautions in securing your computer systems so that they are not vulnerable to intrusion or invite attacks. Hackers can generally wreak much more havoc with a computer system than, say, an Arduino. However, don’t dismiss low-cost computer boards as safe; they aren’t. Most are also good targets for hackers given they often run powerful Linux operating systems.

Summary

The hardware available for IOT solutions includes any discrete electronic component used to create circuits for sensors, power, and many other needs. Within that vast category of electronics are the components used to support features for IOT solutions. More specifically, options are available for building a network of nodes that observe, record, and display information about the world around us.

In this chapter, you examined a long list of microcontrollers focusing on the Arduino line of microcontroller boards, low-cost (low-power) computing boards such as the Raspberry Pi, and the various communication hardware you can use to connect the nodes from using an Ethernet or WiFi network connecting to the Internet to low-cost, low-power wireless communication modules (XBee radios) for connecting sensor nodes.

You also saw a brief tutorial on how to use the Arduino programming environment and even saw an example of a simple sensor-based Arduino project (recall a switch is a simple sensor).

Finally, you took a brief look at sensors including an overview of the types and various sensors available for you to use in your IOT solution.

In the next chapter, you'll examine how the data in an IOT solution can be stored (be that on a local device such as memory-attached file-based solutions), sent to other nodes in a network, or saved to a database server.