Part I

Introduction to Arduino

In This Part

CHAPTER 1

Introduction to Arduino

Electronics enthusiasts have always been present. For decades, they have been creating interesting devices and products. Amateur radio enthusiasts typically made their own radio sets using schematics found in magazines or simply from their own design. How many of us built a radio system to discover electronics, only to be hooked? With a few dollars' worth of components, you could create your own radio and listen to glorious long-wave transmissions on a small low-quality speaker, and yet it was better than what could be bought in the shops because it was homemade. If you wanted better sound, you could buy a better speaker. More volume? There were kits for that, too. Audiophiles built their own amplifiers and accessories depending on their needs. Electronics shops proposed books for all levels, from beginner to expert. Kits were also available using the simplest of components all the way to entire computer systems. It was a time in which you could build just about anything, and everything. You could, quite literally, walk into an electronics shop, buy a DIY computer, and spend a few hours soldering memory chips onto a printed circuit board. That's how I started.

In the 1990s, things changed slightly. Most hobbyists had a PC on their desk and could use them to create schematics, simulate parts of a system, and even print circuit board with transparent layouts, making the entire process much easier. However, something was missing. Almost all the devices that could be made were not programmable. Microprocessors were available but were either too expensive or too complicated. At the time, the 68000 microprocessor was one of the most reliable components available and was relatively cheap but complex. The microprocessor by itself was useless; it had to be hooked up to external memory. To run a program on every boot, it had to also have read-only memory. If you wanted interrupts, again, you had to add a chip into the design. The end result was complicated and out of the reach of some enthusiasts. To do without this complexity, enthusiasts that wanted programmable devices tended to use what was already on their desk: a personal computer.

Most PCs at the time used the ISA bus, as shown in Figure 1-1. ISA was a simple bus that allowed components to be added to the processor and general computer system. It was a simple system that allowed users to insert add-on cards into their computer, and it was extremely easy to use. It wasn't hard to create a circuit board that could be plugged into an ISA slot, and complete prototyping boards existed, enabling enthusiasts and engineers to test a solution before making their own board. Some of these boards even included breadboards, a simple system allowing users to place their components and wires without the need to solder. This sparked a small revolution, and many enthusiasts turned to this type of board to do what previously could not be done: create programmable systems. An ISA board could have digital inputs and outputs, analog inputs and outputs, radios, communication devices—just about anything was possible. All this would be controlled by the computer's CPU, using simple programming languages such as C or Pascal. My ISA card kept my student apartment nice and warm by reading data from a thermometer and turning on electric heaters, acting like a thermostat. It also served as an alarm clock, programmed depending on my classes the next day. Although I did manage to miss a few morning classes, in all fairness it was usually my fault; the ISA card worked perfectly on a tight budget.

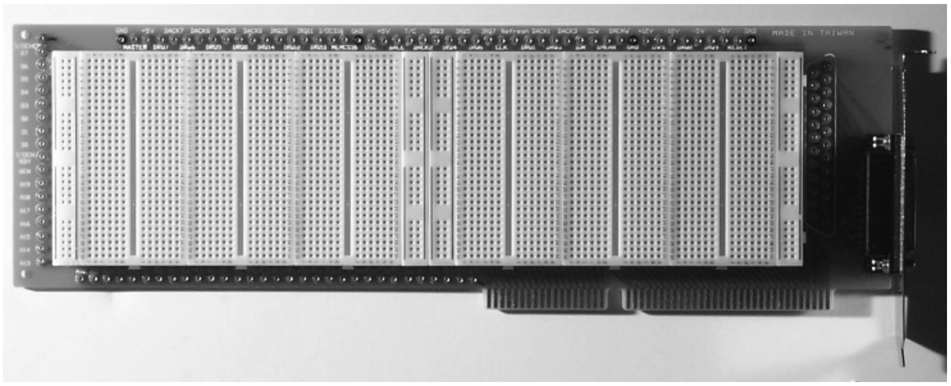

Figure 1-1: ISA prototyping board

Computers became faster, and systems evolved. The industry changed, and so did the expansion ports. Just as enthusiasts became experts on the ISA bus, the industry invented a new system: the VESA Local Bus (VLB). The VLB bus was an extension to ISA, only adding a second connector for memory-mapped I/O and Direct Memory Access (DMA), but it announced a change. Computers were indeed getting faster, and some computer bus systems couldn't keep up. Even VLB couldn't keep up, and after only a year, PCI became the reference. The PCI bus is an advanced bus but requires components and logic to identify itself. It suddenly became increasingly difficult to create homemade boards. Some users decided to use other industry-standard ports, such as the parallel port or RS-232, but most stopped creating such systems. Those that did continue mainly used analog systems or nonprogrammable digital systems. Instead of having a programmable microcontroller, the system was designed using logic gates. For example, a bulb could turn on if both inputs A and B were true, or if input C was false. These tasks became more and more complicated as the number of inputs increased.

Analog systems, such as radios and amplifiers, did not have a form of programming. They were designed with a specific task in mind. Configuration was analog; with a small screwdriver, the designer could "tweak" values with potentiometers, variable resistances. It wasn't possible to program the device to multiply an input signal by a specific value; instead, potentiometers were added to counter the effect of tolerances in the components. Designs therefore added an additional phase, calibration. Specific input signals were fed into devices, and a specific output was expected.

Processors did exist that could be used, and some projects did use them, but integrating a processor into a design generally meant that several components needed to be used. Memory chips, I/O controllers, or bus controllers had to be used, even after a decade of technological advancements, and circuits became more and more complicated. Even when designs worked, programming them proved to be a challenge. Most programming was done via EEPROM devices, short for Electronically Erasable Programmable Read-Only Memory. These devices could contain a computer program and could be programmed using an external programmer attached to a computer. They were called erasable read-only because the contents could indeed be wiped and replaced, but doing so required removal of the circuit and subjecting it to ultra-violet light for 20 minutes. One small error in a program could often take 30 minutes or more to correct.

Atmel AVR

Atmel is an American semi-conductor company, founded in 1984, and the name Atmel is an acronym for Advanced Technology for Memory and Logic. Right from the start, Atmel designed memory chips that used less power than competing designs, but it soon decided to create programmable devices. In 1994, Atmel entered the microprocessor market, creating an extremely fast 8051-based microcontroller. In 1995, Atmel was one of the first companies to license the ARM architecture, giving it access to advanced processor technology.

Atmel didn't use only ARM technology, it also created its own processor, the AVR, in 1996 (see Figure 1-2). What does AVR stand for? Well, that is one of the many mysteries of Atmel. Designed by Alf-Egil Bogen and Vegard Wollan, some say it stands for Alf and Vegard's RISC processor. We will never know, and at the time, people were not interested in understanding the name, but rather getting their hands on this advanced piece of technology. Today, more and more people are curious as to the origin of this curious processor, Atmel continues to tease the public with videos of the inventors explaining the name, only to have the big reveal scrambled by mobile telephone interference.

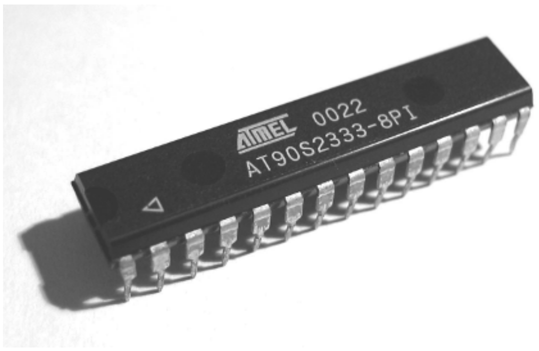

Figure 1-2: Atmel AVR Microprocessor

Previously, programming the read-only memory of a device required some tedious tasks, like subjecting the chip to UV light, or complicated erase techniques. This all changed with Atmel's 8-bit AVR. The AVR was the first microcontroller family to use on-chip flash memory for program storage. It also included Random Access Memory (RAM) directly on the chip, essentially containing everything needed to run a microcontroller on a single chip. Suddenly, all the complicated design could be replaced with a single component. Even better, programming the chip could be done in minutes, using minimal hardware. Some Atmel designs allowed users to plug the microcontroller directly into a USB port and to program it using Atmel's software. From compilation to program execution took less than a minute.

Several learning platforms existed: Parallax's BASIC Stamp and PIC devices were in use, but Atmel's AVR made its appearance and added another alternative for electronics enthusiasts. Previously, on digital systems, the logic was defined before creating the board. Inputs and outputs were connected to logic gates, and the functionality was designed into the product. Now, with the AVR series, enthusiasts and engineers had a new possibility. Instead of designing functionality electronically, systems could be designed to interact with the outside world using computer programming. This simplified electronics; instead of using multiple logic gates, everything was connected directly to the microcontroller, which could then be programmed to react to events from the outside world. Programs could be flashed and re-flashed, and devices could be programmed and re-programmed, opening the gates to a whole new world of electronics. In theory, a device could be made that would adapt to almost every situation possible. The technology existed; all that was left was for someone to create the device.

The Arduino Project

The Arduino project started in 2005, and was a project for the students at the Interaction Design Institute Ivrea in Ivrea, Italy. Students were taught to use a BASIC Stamp, a small microcontroller device programmable in PBASIC (a variation of the BASIC programming language), but the price for this device (almost $75) was considered to be too expensive for students, not only on acquisition, but also to replace damaged units.

Arduino started as a project for design students, targeted as a replacement for the BASIC Stamp. The Atmel 8-bit AVR was chosen for its simplicity and low price, and had the extra advantage of requiring few external components. It also has an impressive amount of inputs and outputs, making it a perfect choice for future designs.

Students and teachers worked together on a new design, one that used the Atmel AVR and that could easily accept external cards. When the original design was completed, researchers worked to make the design lighter, less expensive and easily usable by students, enthusiasts, and engineers. The first Arduino board was born. Improvements on the Arduino's original design, such as replacing the DB-9 serial connector with USB, has helped expand the platform's appeal.

There are two sides to every Arduino. There is, of course, the hardware, but this is only part of an Arduino project. Every Atmel microcontroller used for Arduino comes with a specific firmware, a small program embedded on every device that looks for a program to run or helps install a program using a serial device.

The final design was released as open source and was designed and sold by Arduino. Releasing Arduino as an Open Source Hardware project was an interesting move. Because it was open source, it attracted more and more users to look into their projects. Because the Arduino already had an excellent input/output design, users began to create boards that could be added to the original Arduino. When Arduino designed a new board, it kept the original input/output layout, enabling existing add-ons to be used with new designs.

Originally designed for education, the Arduino project became famous with electronics enthusiasts, and its boards were sold by more and more distributors.

Arduino not only created the hardware—an embedded device that does not have corresponding software and support programs might still be difficult to use—but also spent a lot of time developing its own language and Integrated Development Environment (IDE). The end result is a nice IDE that can work on Windows, MacOS, and Linux and converts the Arduino language (a high level variant of C/C++) to AVR code. The Arduino development environment hides away all the complications linked to embedded systems and mixing software—such as setting up an environment, linkers, pesky command lines—and lets the developer program using simple C language functions through the Arduino Programming Language.

The ATmega Series

Atmel has placed its AVR design into different groups, depending on various factors. There are numerous AVR microcontrollers, and knowing which one to use is essential for projects. Some ATmega devices have more memory, or more digital and analog inputs and outputs, or have a specific package size.

The ATmega Series

The Atmel megaAVR is the muscle of the AVR series. They are designed for applications requiring large amounts of code, with flash memory ranging from 4 k all the way to 512 k, enough for the most demanding of programs. Atmel megaAVR devices come in various sizes, ranging from 28 pins all the way to 100 pins. These devices have an impressive amount of embedded systems: analog to digital converters, multiple serial modes, and watchdog timers, to name but a few. They also have a large amount of digital input and output lines, making them ideal for devices that communicate with numerous components.

There are close to 100 ATmega devices, ranging in flash memory size and package size, and some models have advanced features such as internal LCD Controllers, CAN controllers, USB controllers, and Lightning controllers. ATmega chips are found in almost every Arduino board produced.

You can find more information on the ATmega series on Atmel's website at: `http://www.atmel.com/products/microcontrollers/avr/megaavr.aspx`.

The ATtiny Series

The Atmel tinyAVR series has small-package devices designed for applications that require performance and power efficiency. These devices live up to their name "tiny"; the smallest tinyAVR is 1.5 mm by 1.4 mm. The word "tiny" is only a reference to their size. Their power is comparable to the larger AVRs; they have multiple I/O pins that can be easily configured and a Universal Serial Interface that can be configured as SPI, UART, or TWI. They can also be powered with as little as 0.7 V, making them highly energy-efficient. They can be used in single-chip solutions or in glue logic and distributed intelligence in larger systems.

There are more than 30 ATtiny devices, and they come with between 0.5 k and 16 k of flash memory, and range from 6-pin packages to 32-pin packages. You can find more information on the ATtiny series on Atmel's website at: `http://www.atmel.com/products/microcontrollers/avr/tinyavr.aspx`.

While the ATtiny series are powerful devices given their size, no Arduino uses this device as its microcontroller.

Other Series

Atmel also has different AVR series: The XMEGA series deliver real-time performance, with added encryption using AES and DES modules, and includes an interesting technology, the XMEGA Custom Logic, reducing the need for external electronics.

Atmel also produces a 32-bit version of its AVR microcontroller: the UC3. Supporting fixed-point DSP, a DMA controller, Atmel's famous Peripheral Event System and advanced power management, the UC3 is a formidable microcontroller. You can find more information on Atmel's AVR website at: `http://www.atmel.com/products/microcontrollers/avr/default.aspx`.

The Different Arduinos

The original Arduino was designed for one specific task, and it fit that task perfectly. With the success of the original Arduino board, the company decided to create more designs, some of them for very specific tasks. Also, because the original Arduino design was open source, several companies and individuals have developed their own Arduino-compatible boards, or have followed in the open source tradition, and have proposed their modifications to Arduino. Arduino has begun a certification program to ensure compatibility with boards that use different processors, with the Intel Galileo being the first to receive such a certification. Anyone is free to make their own Arduino-based derivative, but the name and logo of Arduino are trademarked. As such, you'll find a number of boards with names ending in "uino", implying compatibility.

WARNING **Beware of counterfeits! Some companies propose Arduino boards that are cheaper than the original Arduino series, but these boards tend to have less reliable hardware. Arduino boards are cheap but still use good quality electronic components, whereas counterfeit boards may well use components that will not last as long. Paying a few extra dollars for a board helps Arduino finance more research to create new Arduino boards and software, and ensures a better user experience. You can read more about how to spot counterfeit boards at: `http://arduino.cc/en/Products/Counterfeit`.**

Arduino made the board design open source, but it still produces its own boards. These boards are known as official boards. Other companies also make Arduino-compatible boards.

Arduino Uno

The Arduino Uno is the "standard" Arduino board and the most readily available. It is powered by an Atmel ATmega328, with a total of 32 KB of flash memory, 2 KB of SRAM, and 1 KB of EEPROM memory. With a total of 14 digital I/O pins and 6 analog I/O pins, this is a very capable device, able to run most programs. An on-board ATmega16u2 chip manages serial communication. It is one of the least expensive boards and the most used. When starting a new project, if you do not know what Arduino to use, start with the Uno, as shown in Figure 1-3.

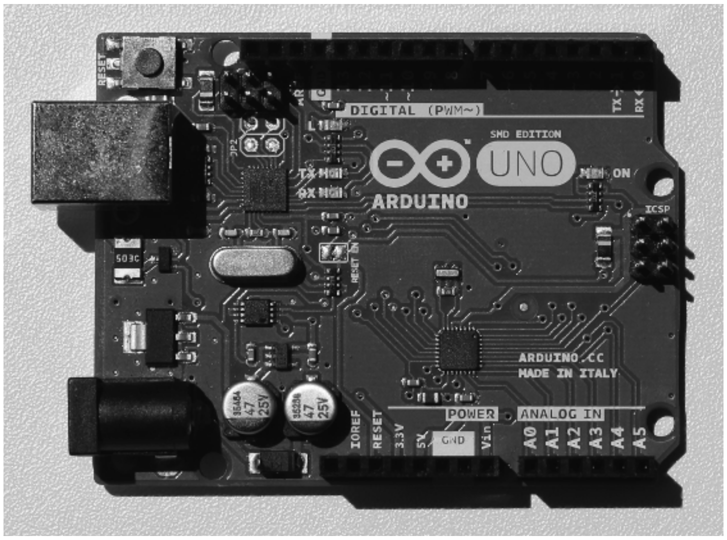

Figure 1-3: The Arduino Uno

Arduino Leonardo

The Arduino Leonardo is slightly different to the Uno. Based on the ATmega32u4, this microcontroller has enhanced USB capabilities and therefore does not require a dedicated microchip for USB serial communication like the Uno. One advantage to this is cost; one less microchip means a cheaper solution. It also means that a developer can use the microcontroller as a native USB device, increasing flexibility in the communication with a computer. The Leonardo can effectively emulate a keyboard and mouse via USB HID, as shown in Figure 1-4.

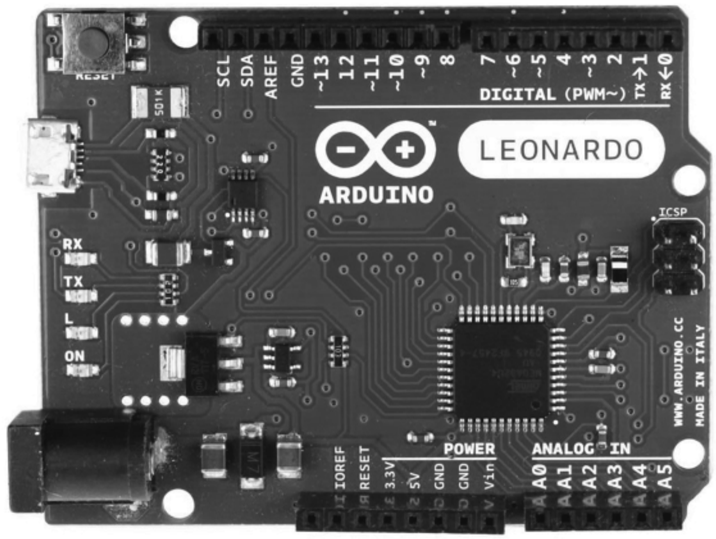

Figure 1-4: The Arduino Leonardo

Arduino Ethernet

The Arduino Ethernet, based on the ATmega328 found in the Uno, can connect to an Ethernet network, a functionality needed in a number of projects. Physically, the Arduino Ethernet has the same 14-digital inputs/outputs as the Arduino Uno, with the exception that 4 are used to control the Ethernet module and on-board micro-SD card reader, limiting the amount of pins available.

It is interesting to note that the Arduino Ethernet has an optional POE module, short for Power Over Ethernet. This option enables the Arduino Ethernet to be powered directly from an Ethernet connection, without the need for an external power source provided that there is a POE supply on the other end of the Ethernet cable. Without POE, the Arduino must be powered by an external source

Another difference from other Arduino boards is the lack of a USB connector. Because most of the space is taken up with an Ethernet connector, this device instead supports a 6-pin serial programming header and is compatible with numerous programming devices (including a device from Arduino, the USB-Serial adapter). The Arduino Ethernet is shown in Figure 1-5.

Arduino Mega 2560

The Arduino Mega 2560 is only slightly larger than the Arduino Uno, but it has more input and output pins. It has a total of 54 digital I/O pins and 16 analog inputs. It also has a large amount of flash memory: 256 KB, capable of storing larger programs than the Uno. It also has generous SRAM and EEPROM: 8 KB and 4 KB, respectively. It also has 4 hardware UART ports, making it an ideal platform for communicating with multiple devices serially.

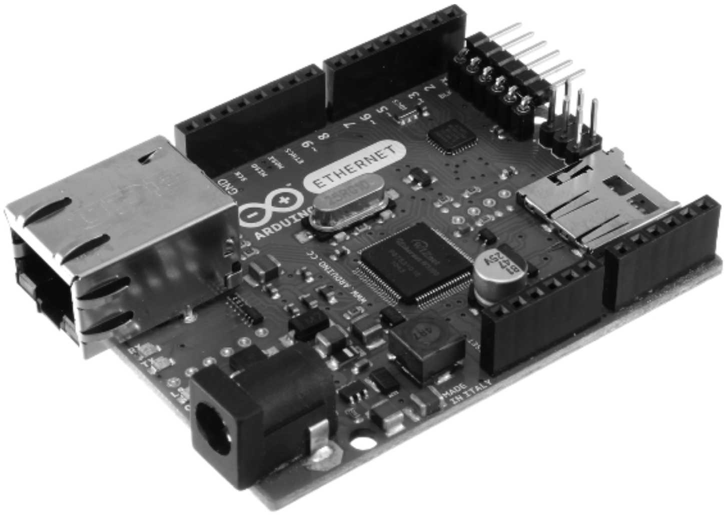

Figure 1-5: The Arduino Ethernet

Arduino Mega boards are used when large amount of inputs and outputs are required. It is shown in Figure 1-6.

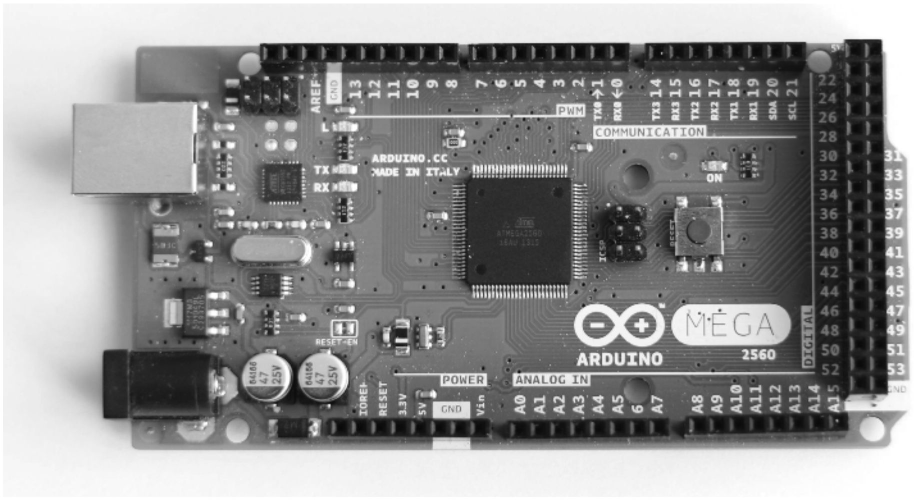

Figure 1-6: The Arduino Mega 2560

Arduino Mini

The Arduino Mini is a tiny device, useful for applications where space is reduced to the absolute minimum (see Figure 1-7). It has 14 digital I/O pins and 4 analog input pins. (Four more are available but are not broken out.) The device has the strict minimum: it does not have a USB connector; it has no power regulator; and it has no headers. Programming is done via an external USB or RS232 to TTL serial adapter. It is shown in Figure 1-7.

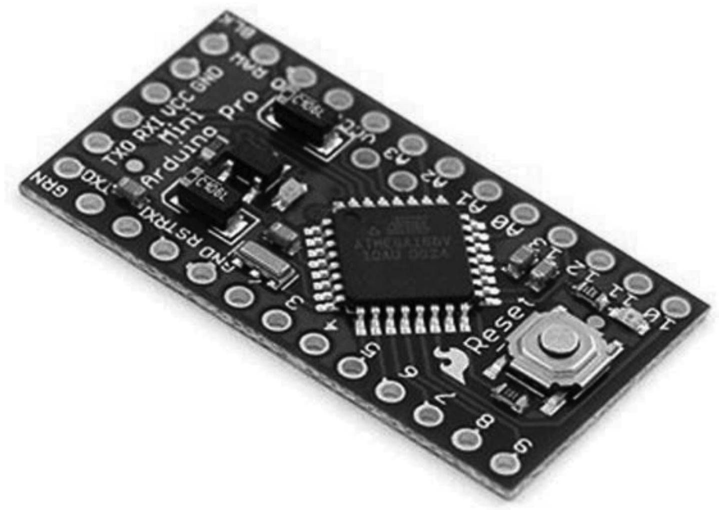

Figure 1-7: The Arduino Mini

Arduino Micro

The Arduino Micro lives up to its name; it is one of the smallest Arduino boards available. Despite its small size, it still has a large amount of input and output pins; it has 20 digital input/output pins, of which 7 can be used as PWM outputs. It also has 12 analog inputs.

The Micro is not designed to have shields but it does have an interesting layout, as shown in Figure 1-8. It can be placed directly onto a breadboard.

Arduino Due

The Arduino Due differs from all other Arduino designs in that it is not based on an AVR, but rather uses a microcontroller based on an ARM Cortex-M3, the Atmel SAM3X8E. This advanced microcontroller is clocked at 84 MHz and is a full 32-bit device. It has a large amount of digital and analog I/O: 54 digital pins (12 of which can be used as PWM) and 12 analog inputs. The board has 4 UARTs, an SPI header, a Twin-Wire Interface, and even includes a JTAG header.

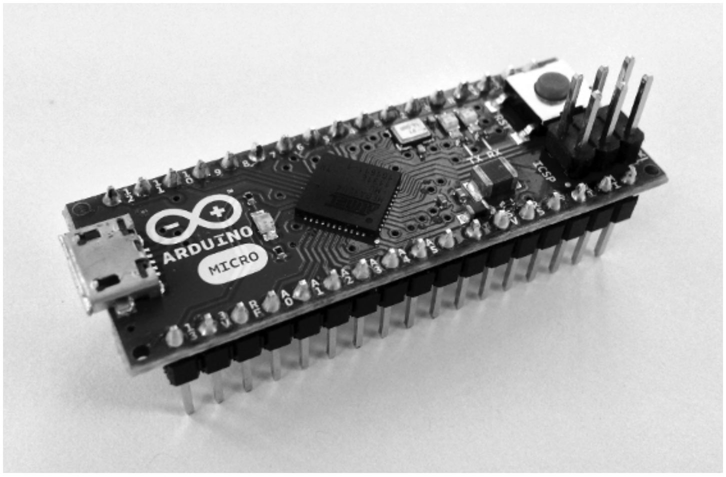

Figure 1-8: The Arduino Micro

The Arduino Due has more strict power supply requirements, and the microcontroller itself is powered under 3.3 V. Be careful not to apply 5 V to any of the pins: otherwise, you will damage the board. When choosing a shield for the Due, make sure the shield supports 3.3 V. You can identify if a shield is Due compatible by making sure it conforms to the Arduino R3 layout.

The Arduino Due is an incredibly powerful Arduino. The Due has 512 KB of flash memory and a total of 96 KB of SRAM. It can handle the largest programs at a fast speed. If you have a lot of calculations to perform, this is the Arduino that you need (Figure 1-9).

LilyPad Arduino

The LilyPad Arduino is an interesting device. It strays from the typical Arduino build because it is not rectangular, but round. Secondly, it does not support shields. What it is designed for, however, is to be a small device that is perfect for wearable computing, or e-fabric. The round shape means that connectors are evenly distributed, and its small scale (2 inches in diameter) makes it perfect for wearable devices. This device is easily hidden, and multiple manufacturers have designed devices especially for the LilyPad: Wearable LEDs, light sensors, even battery supply boxes that can be sewn into fabric.

To make the LilyPad as small and as light as possible, some sacrifices were made. The LilyPad does not have a voltage regulator, so it is vitally important to deliver at least 2.7 volts, but more important, no more than 5.5 volts; otherwise, the LilyPad will be destroyed (see Figure 1-10).

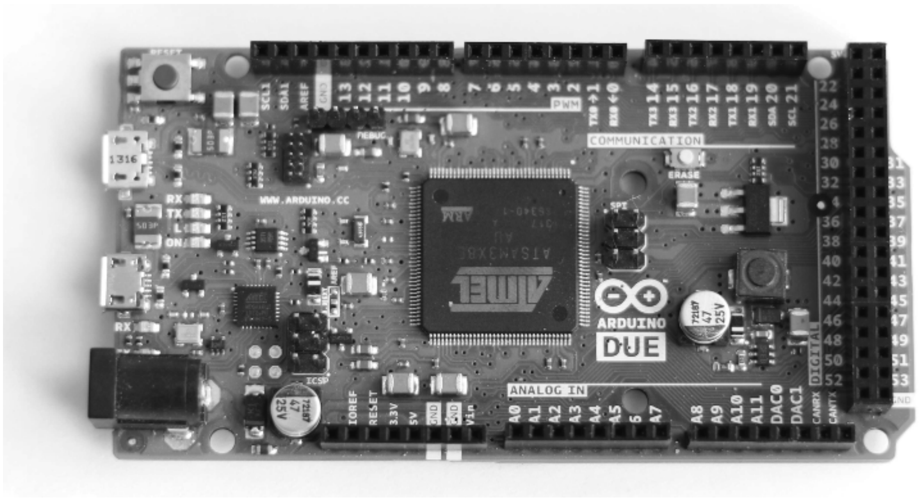
Figure 1-9: The Arduino Due

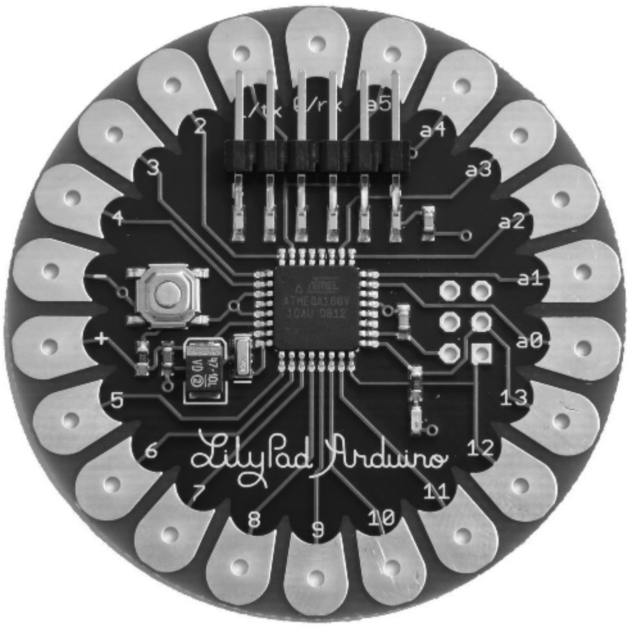
Figure 1-10: The LilyPad Arduino

Arduino Pro

The Arduino Pro exists in two versions, based either on the ATmega168 or the ATmega328. The 168 version operates at 3.3 V with an 8 MHz clock, and the 328 version runs on 5 V at 16 MHz. Both versions have 14 digital inputs/outputs and 6 analog inputs. It has a JST battery power connector, a power switch to select between power modes, and space reserved for a power jack, if needed. It does not have a USB connector but instead uses a FTDI cable for programming.

The Arduino Pro is different from most other Arduinos in that while it is a prototyping board it is designed to be embedded in projects. It does not come with headers—indeed, it does not have any headers at all, as shown in Figure 1-11. All the digital and analog inputs and outputs are placed at the exterior of the board, retaining shield layout, ready to be soldered to wire or connectors if necessary. Instead of being used for prototyping, the Arduino Pro is aimed at semipermanent installation in finished products. The Arduino Pro was not designed by Arduino but was designed and is manufactured by SparkFun Electronics.

Arduino Robot

The Arduino Robot is, simply put, an Arduino on wheels. There are two Arduino boards on the Robot—one controls the on-board motors, and the other contains sensors. The Control board controls the Motor board and gives it instructions on how to operate.

The Control board is powered by an ATmega32u4, with 32 KB of flash, 2.5 KB of SRAM, and 1 KB of EEPROM. It also has an external I2C EEPROM device, providing more storage. It has a compass, a speaker, three LEDs, a five-button key pad, and an LCD screen. It also has three solder points for external I2C devices. It also has I/O capability, with five digital I/Os, six PWMs, and four analog inputs. There is space for eight analog inputs (for distance sensors, ultrasound sensors, or other sensors) and six digital I/O pins for other devices (four of which can be used for analog input).

The Motor board is a fully independent board, powered by an ATmega32u4, the same microcontroller as on the Control board. The Motor board contains two wheels powered independently, five IR sensors, and I2C and SPI ports. It also contains the power supply; it is powered by four rechargeable AA batteries, and contains a jack port to recharge the on-board batteries. The board can also be powered by an on-board USB connector, but in this configuration, for safety reasons, the motors are disabled (Figure 1-12).

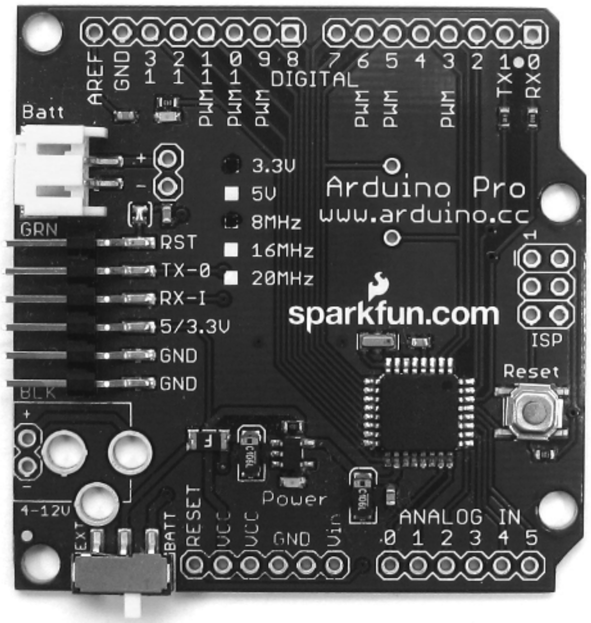

Figure 1-11: The Arduino Pro

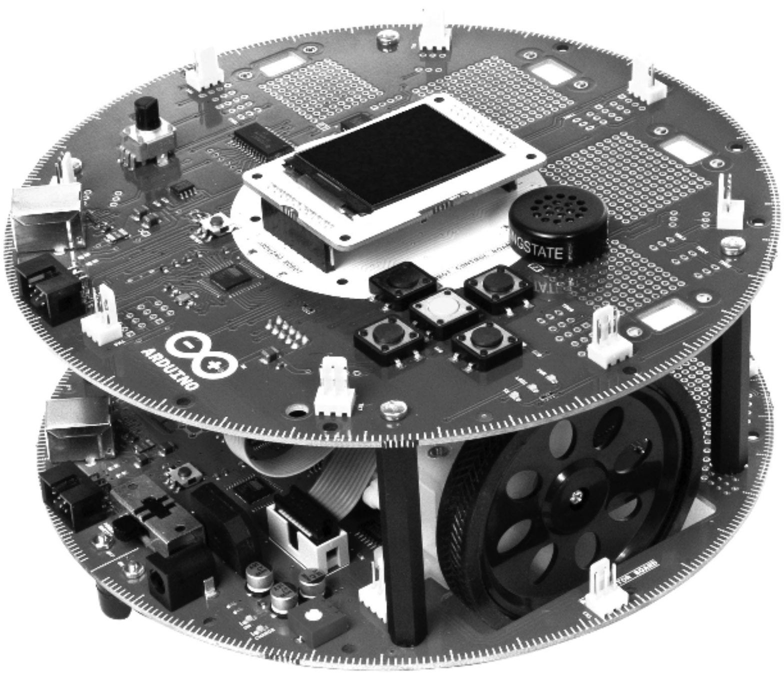

Figure 1-12: The Arduino Robot

Arduino Esplora

The Arduino Esplora is a strange device. Where most Arduinos are designed to sit on a table or be placed under fabric, the Esplora is designed to be held in your hand. Based on an ATmega32u4, it is not shield compatible and does not have any solder points for inputs and outputs. Instead, it looks and feels like a game pad; it has thumb inputs in the form of four digital switches, one analog joystick, and a linear potentiometer. For more feedback, the Esplora has a buzzer and an RGB LED. It also features more advanced devices; it has an on-board microphone, a temperature sensor, a connector for an LCD screen, and a three-axis accelerometer.

The Esplora has 32 KB of flash; 4 KB are used by the bootloader. It has 2.5 KB of SRAM, and 1 KB of EEPROM. It is a capable device, and it makes up for its lack of connectors with four TinkerKit connectors: two inputs and two outputs, as shown in Figure 1-13.

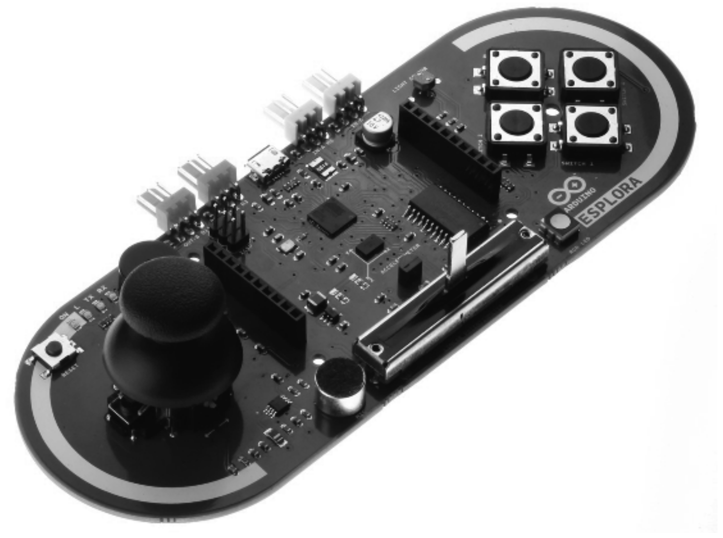

Figure 1-13: The Arduino Esplora

Arduino Yún

The Arduino Yún is based on an ATmega32u4, but it also has an Atheros AR9331 on the same board. The Atheros processor has a complete Linux distribution, based on OpenWRT, famous for Linux-based wireless routers.

The Arduino Yún has built-in Ethernet and WiFi, and also has a micro-SD slot. The Yún is different from other Arduinos and shields in that it has advanced network functionality; the Arduino can send commands to OpenWRT and then continue processing its sketch (Figure 1-14). The two processors work independently, the Bridge library facilitates communication between the two processors.

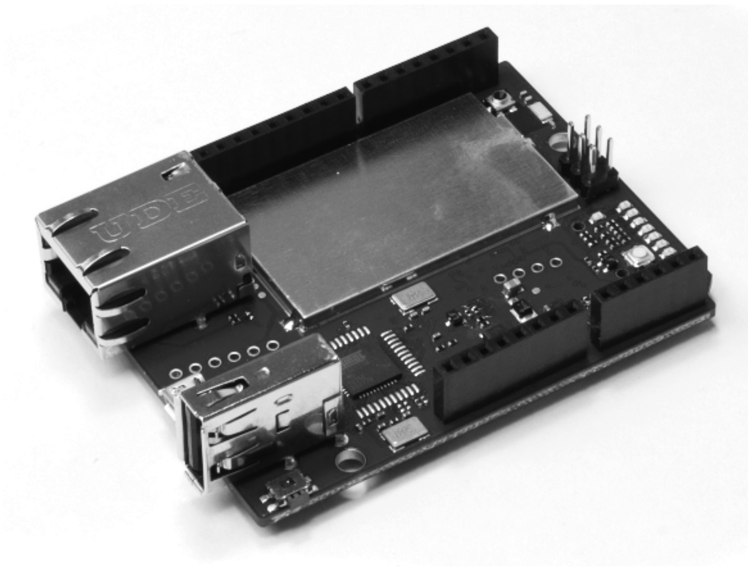

Figure 1-14: The Arduino Yún

Arduino Tre

The not-yet-released Arduino Tre promises to be a phenomenal beast. Up until now, the fastest Arduino was the Arduino Due, based on an ARM-compatible microcontroller. The Tre, created by Arduino and BeagleBoard, combines the power of a full computer with the flexible input and output of an Arduino.

The Tre has a Cortex-A8 class processor, the Sitara AM335X processor, running at 1 GHz. This processor has access to 512 MB of RAM and has an HDMI port capable of displaying Full HD (1920 x 1080). All this power is interfaced by an Atmel ATmega32u4 using the Arduino programming environment that enthusiasts have come to love.

Arduino Zero

The Arduino Zero is a brand new Arduino using Atmel's SAM D21 microcontroller. It has 256 KB of flash memory, 32 KB of RAM, and runs at 48 MHz. The Arduino Zero is designed to handle future requirements from the Maker community, by creating a design that is powerful, robust, and flexible enough to be used in robotics and wearable projects, as well as the IoT. It is also the first design to have an advanced debugger interface.

Your Own Arduino?

Arduino has always created open-source designs, and all the boards listed previously have schematic files available directly from the Arduino website, under a Creative Commons Attribution Share-Alike license. Put simply, this means that you are free to study the Arduino schematics to make your own or to make modifications either for personal use or professional use on the condition that you give credit to Arduino for the original design and release your own design under the same license.

With the exception of the Arduino Due, all Arduino boards are based on the Atmel AVR. These chips can be bought from electronic distributors with the Arduino firmware pre-installed, or if you have the proper tools, you can buy blank chips and load the firmware yourself.

Shields

An Arduino by itself is a capable device and already includes numerous input and outputs, but its power only starts there. Because Arduino designs are open source, numerous companies have developed shields, printed circuit boards that are placed on top of the Arduino board that connect to the Arduino's pins. There shields add functionality by using different inputs and outputs, either digital I/O or through serial communication.

What Is a Shield?

A *shield* is a printed circuit board that can be placed on the top of most Arduino boards. It connects to the Arduino's processor through male header pins. Adding a shield to an Arduino does not necessarily expand the possibilities of an Arduino, but most do.

For most prototyping projects, you connect wires to the Arduino's headers and connect them to a breadboard. This is easy enough for a lot of applications, like outputting data to two or three LEDs. For more complex applications, a breadboard isn't practical due to the complexity of the wiring, or the size of the components. Micro-SD card readers are extremely small and cannot be placed onto a breadboard. Soldering wires to a micro-SD reader isn't particularly easy, so your choices are limited. Writing data to a micro-SD card is something that can happen a lot, so it's fortunate several companies have developed shields with a micro-SD reader. If your application requires data logging, all you have to do is to connect the shield to the top of the Arduino, add a few lines of code, and you are ready to go. It is that simple.

As said previously, not all shields add functionality. Some shields exist to help prototyping— allowing you to solder components onto the shield—without having to make your own PCB. Prototyping on a breadboard is an excellent way to test that your design works, but after the design is proven, it is time to make a better board. For example, if you were creating a doorbell application, it would be complicated to hide a breadboard behind the ringer. Instead, you could solder those components onto a prototyping board, saving space and making your design much more resistant to shock or tampering. The added advantage of this type of board is that you do not need to create your own printed circuit board or do any complicated routing.

The Different Shields

Shields exist for a wide variety of applications: storage on SD cards, network connectivity by Ethernet or WiFi robotics control, enabling displays like LCD and TFT screens, to name but a few.

Most shields can be stacked, so you are not limited to using only one at a time. However, some shields may require input and outputs that will subsequently be unavailable to other designs. Be careful when you choose your shields!

Arduino Motor Shield

When using motors, special care has to be taken. When turned off, motors can induce voltage spikes, and components need to be added to a design account for this possibility. Also, typically, USB power is insufficient for motors. The Arduino Motor Shield takes care of this and enables the programmer independent control of two DC motors, or one stepper motor. This shield can either be powered from the Arduino or rely on an external power supply.

Arduino Wireless SD Shield

The Wireless SD shield is designed for an Xbee module but works with any radio modem with the same footprint. The on-board micro-SD slot allows the shield to act as a data logger. It also has a small prototyping area for adding components.

Arduino Ethernet Shield

The Arduino Ethernet shield does exactly as the name implies; it adds Ethernet connectivity through a W5100 controller, supporting up to four simultaneous socket connections. This module also includes a micro-SD slot for data-logging.

The Arduino Ethernet Shield has an optional POE module. On a POE network, the module (and the parent Arduino) can be powered directly over Ethernet.

Arduino WiFi Shield

The Arduino WiFi Shield includes an HDG104 Wireless LAN controller, enabling an Arduino to access 802.11b/g networks. It can connect to open and encrypted networks. This module also includes a micro-SD slot for data-logging.

Arduino GSM Shield

The Arduino GSM shield connects to the Internet through a GPRS network, at a maximum of 85.6 KBps. It also has voice capabilities; by adding an external microphone and speaker circuit, it can make and receive voice calls. It can also send and receive SMS messages. The modem, an M10 by Quectel, is configured using AT commands, handled in software by the GSM library.

The Arduino GSM Shield comes with a Bluevia SIM card; which allows for machine-to-machine roaming data connections in blocks of 10 or 20 megabytes. However, the GSM shield will work with a SIM card from a different provider.

Your Own Shield

In some cases, you will want to make your own electronics. For prototyping, a breadboard is sufficient, but when you need something more robust and more professional, it is time to make your own shield. There are several software options to assist you, but one of the best is the Fritzing application. In Fritzing, you can create breadboard designs, translate them into electronic schematics, and generate a shield layout directly. Fritzing also has its own shield creation system; just upload your schematic to its website and receive a professionally built shield.

What Can You Do with an Arduino?

This is one of the most commonly asked questions, but the answer is both simple and complicated. Put simply, you can do almost anything you can imagine. The most difficult part of any Arduino project is identifying a need. Maybe you have an aquarium at home and would like to control the lighting in a specific way? Maybe you would like to add a parking assist device onto your car. Some people just want to add some automation to their house, opening and closing motorized shades at the push of a button. Some people come up with even more amazing and fun projects: a remote-controlled lawn mower, even a chess playing robot. The possibilities are almost unlimited. There are a few things that an Arduino cannot do, but that list is becoming shorter every time a new Arduino-compatible board is released.

Arduino is an excellent way to learn about software development and electronics because it is a low-cost, robust device that is easy to program.

Some people use Arduino for hobbyist electronics, with projects ranging from the simple to the incredibly absurd. I know of one person who has entirely automated his house using 10 Arduino Megas, each room communicating with the others to better estimate electrical consumption, heating, and personal comfort.

Arduino is also used professionally because the components are low-cost and highly reliable and have the added flexibility of being open source. When an initial design is completed, developers can make a board much smaller to be included in toys, small embedded systems, and even industrial machines. Several 3-D printers are based on Arduino for their ease of use and reliability.

What You Will Need for This Book

Each chapter has a list of elements required to complete. However, when creating an Arduino project, a few items are required every time. Following is a list:

- **A power supply**—The Arduino Uno accepts an input voltage of 6 to 20 V, with 7 to 12 V being recommended. Any standard AC-to-DC center-positive adapter should work fine, preferably one that can supply up to or over 1 amp of current.
- **Multimeter**—Almost any model. You do not need to buy the most expensive, far from it, but it should test DC voltage, DC amperage and continuity, with optional resistance calculation, and AC voltage and amperage if you plan to interface your Arduino to main's power.
- **Breadboard**—The size depends on your project. Consider a medium-sized board; if it is too small you might not fit all your components (or it might be too cramped, possibly creating short circuits), and large breadboards can cost more and require more space. (I use 680-point breadboards for most of my examples and projects.)
- **Resistors**—A common element of every project. There are numerous values, but there are some values that will be used more often. There are kits on the market that propose 10 of every value, or you can go with the most common, the choice is yours. To start out, ten 220-ohm, ten 1-kilohm, and ten 10-kilohm resistors should suffice.
- **LEDs**—A great way of knowing the output of a pin. Coupled with a resistor, it can instantly show the state of your project.
- **Other electronic components**—Sometimes it is handy to have a small collection of capacitors, switches, and diodes on hand. Each example in this book has a complete list of the required components.

Summary

This chapter briefly talked about some of what an Arduino can do, but there is no way of knowing exactly what everyone will do with it. As I said, your only limitation will be your imagination, and I would love to hear about what you have done with an Arduino! You can contact me on my website at `http://packetfury.net`. I look forward to hearing about your projects!

In the next chapter, you will learn more about programming an Arduino, including how to install the Arduino IDE, how to connect an Arduino to your computer, and uploading your first sketch.