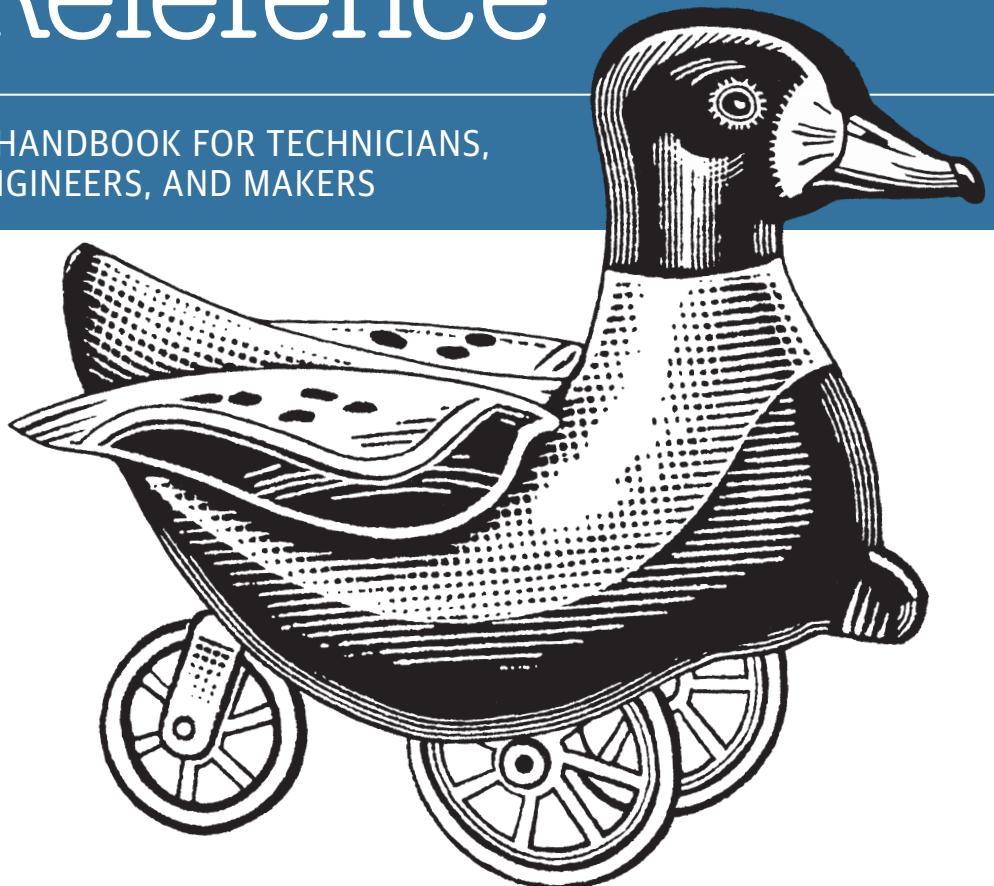


Arduino

A Technical Reference

A HANDBOOK FOR TECHNICIANS,
ENGINEERS, AND MAKERS



J. M. Hughes

Arduino: A Technical Reference

A Handbook for Technicians, Engineers, and Makers

J. M. Hughes

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Arduino: A Technical Reference

by J. M. Hughes

Copyright © 2016 John Hughes. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safaribooksonline.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editor: Dawn Schanafelt

Interior Designer: David Futato

Production Editor: Colleen Lobner

Cover Designer: Randy Comer

Copyeditor: Rachel Head

Illustrator: John M. Hughes

Proofreader: Kim Cofer

and Rebecca Demarest

Indexer: Ellen Troutman-Zaig

May 2016: First Edition

Revision History for the First Edition

2016-05-05: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781491921760> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Arduino: A Technical Reference*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-92176-0

[LSI]

Table of Contents

Preface.....	xiii
1. The Arduino Family.....	1
A Brief History	1
Types of Arduino Devices	2
Arduino Galleries	4
Arduino-Compatible Devices	7
Hardware-Compatible Devices	7
Software-Compatible Devices	8
The Arduino Naming Convention	9
What Can You Do with an Arduino?	10
For More Information	12
2. The AVR Microcontroller.....	13
Background	14
Internal Architecture	14
Internal Memory	17
Peripheral Functions	17
Control Registers	18
Digital I/O Ports	18
8-Bit Timer/Counters	19
16-Bit Timer/Counters	21
Timer/Counter Prescaler	21
Analog Comparator	21
Analog-to-Digital Converter	22
Serial I/O	24
USART	24
SPI	25

TWI	25
Interrupts	26
Watchdog Timer	29
Electrical Characteristics	29
For More Information	29
3. Arduino-Specific AVR Microcontrollers.....	31
ATmega168/328	32
Memory	32
Features	32
Packages	34
Ports	34
Pin Functions	34
Analog Comparator Inputs	34
Analog Inputs	34
Serial Interfaces	35
Timer/Clock I/O	36
External Interrupts	37
Arduino Pin Assignments	38
Basic Electrical Characteristics	38
ATmega1280/ATmega2560	39
Memory	39
Features	41
Packages	41
Ports	42
Pin Functions	42
Analog Comparator Inputs	42
Analog Inputs	43
Serial Interfaces	44
Timer/Clock I/O	45
External Interrupts	46
Arduino Pin Assignments	46
Electrical Characteristics	49
ATmega32U4	49
Memory	49
Features	51
Packages	51
Ports	51
Pin Functions	52
Analog Comparator Inputs	53
Analog Inputs	53
Serial Interfaces	54

Timer/Clock I/O	55
External Interrupts	56
USB 2.0 Interface	57
Electrical Characteristics	58
Arduino Pin Assignments	59
Fuse Bits	60
For More Information	62
4. Arduino Technical Details.....	63
Arduino Features and Capabilities	63
Arduino USB Interfaces	64
Arduino Physical Dimensions	66
Full-Size Baseline Arduino PCB Types	67
Mega Form-Factor Arduino PCB Types	68
Small Form-Factor Arduino PCB Types	69
Special-Purpose PCB Types	73
Arduino Pinout Configurations	73
The Baseline Arduino Pin Layout	74
The Extended Baseline Pin Layout	76
The Mega Pin Layout	81
Nonstandard Layouts	83
For More Information	87
5. Programming the Arduino and AVR Microcontrollers.....	89
Cross-Compiling for Microcontrollers	90
Bootloaders	92
The Arduino IDE Environment	94
Installing the Arduino IDE	95
Configuring the Arduino IDE	96
Cross-Compiling with the Arduino IDE	98
The Arduino Executable Image	101
The Arduino Software Build Process	101
Sketch Tabs	103
Arduino Software Architecture	104
Runtime Support: The main() Function	106
An Example Sketch	107
Constants	110
Global Variables	111
Libraries	112
Using Libraries in Sketches	112
Adding a Library to the Arduino IDE	116
Creating Custom Libraries	118

Arduino Source Code	119
6. Life Without the Arduino IDE	121
IDE Alternatives	121
PlatformIO	122
Ino	124
The AVR Toolchain	125
Installing the Toolchain	127
make	130
avr-gcc	131
binutils	132
avr-libc	135
Building C or C++ Programs from Scratch	137
Compiling with avr-gcc or avr-g++	137
Multiple Source Files and make	138
AVR Assembly Language	140
The AVR Programming Model	141
Creating AVR Assembly Language Programs	143
AVR Assembly Language Resources	146
Uploading AVR Executable Code	146
In-System Programming	147
Programming with the Bootloader	148
Uploading Without the Bootloader	149
JTAG	151
AVRDUDE	152
Using an Arduino as an ISP	154
Bootloader Operation	154
Replacing the Bootloader	156
Summary	156
7. Arduino Libraries	157
Library Components	158
Contributed Libraries	211
8. Shields	215
Electrical Characteristics of Shields	216
Physical Characteristics of Shields	217
Stacking Shields	219
Common Arduino Shields	220
Input/Output	221
I/O Extension Shields	222
I/O Expansion Shields	226

Relay Shields	230
Signal Routing Shields	232
Memory	235
Communication	237
Serial I/O and MIDI	237
Ethernet	239
Bluetooth	241
USB	243
ZigBee	244
CAN	246
Prototyping	249
Creating a Custom Prototype Shield	253
Motion Control	253
DC and Stepper Motor Control	254
PWM and Servo Control	255
Displays	257
Instrumentation Shields	263
Adapter Shields	266
Miscellaneous Shields	268
Uncommon Arduino Shields	272
Sources	274
9. Modules and I/O Components.....	275
Modules	276
Physical Form Factors	277
Interfaces	277
Module Sources	280
Module Descriptions	282
Grove Modules	309
Sensor and Module Descriptions	310
Sensors	312
Temperature, Humidity, and Pressure Sensors	312
Tilt Sensors	318
Audio Sensors	319
Light Sensors	320
Magnetic Sensors	324
Vibration and Shock Sensors	324
Motion Sensors	325
Contact and Position Sensors	327
Range Sensors	331
Communications	332
APC220 Wireless Modules	332

315/433 MHz RF Modules	332
ESP8266 Transceiver	333
Output Devices and Components	334
Light Sources	335
Relays, Motors, and Servos	339
Analog Signal Outputs	342
User Input	343
Keypads	343
Joysticks	344
Potentiometers and Rotary Encoders	345
User Output	345
Text Displays	345
Graphical Displays	347
Support Functions	347
Clocks	348
Timers	350
Connections	351
Working with Naked Jumper Wires	351
Module Connection Systems	351
Building Custom Connectors	352
Choosing a Connection Method	354
Sources	355
Summary	355
10. Creating Custom Components.	357
Getting Started	360
Custom Shields	365
Physical Considerations	366
Stacking Shields	367
Electrical Considerations	369
The GreenShield Custom Shield	369
Objectives	370
Definition and Planning	370
Design	371
Prototype	379
Final Software	385
Fabrication	393
Final Acceptance Testing	397
Operation	399
Next Steps	400
Custom Arduino-Compatible Designs	400
Programming a Custom Design	401

The Switchinator	401
Definition and Planning	402
Design	403
Prototype	416
Software	420
Fabrication	423
Acceptance Testing	427
Next Steps	428
Resources	428
11. Project: A Programmable Signal Generator.....	431
Project Objectives	433
Definition and Planning	434
Design	435
Functionality	436
Enclosure	437
Schematic	438
Prototype	441
Control Inputs and Modes	441
Display Output	443
DDS Module	444
Software	446
Source Code Organization	447
Software Description	448
The DDS Library	456
Testing	457
Final Assembly	460
Pull-up Resistor Array	460
Input Protection	461
Chassis Components	462
DC Power	465
Final Testing and Closing	466
Reducing the Cost	466
Cost Breakdown	468
Resources	469
12. Project: Smart Thermostat.....	471
Background	471
HVAC Overview	472
Temperature Control Basics	473
Smart Temperature Control	476
Project Objectives	477

Definition and Planning	477
Design	478
Functionality	478
Enclosure	480
Schematic	482
Software	482
User Input/Output	485
Control Output	488
Prototype	489
DHT22 Sensor	491
Rotary Encoder	491
Real-Time Clock Module	493
LCD Shield	493
Software	493
Source Code Organization	494
Software Description	494
Testing	497
Final Version	498
Assembly	498
Testing and Operation	501
Cost Breakdown	502
Next Steps	503
Resources	503
13. Model Rocket Launcher: A Design Study.....	505
Overview	505
The Design Cycle	506
Objectives	508
Selecting and Defining Functional Requirements	510
Creating the Preliminary Design	514
Design Feasibility	517
Preliminary Parts List	520
Prototype	521
Final Design	522
Electrical	522
Physical	527
Software	529
Testing and Operation	532
Cost Analysis	533
A. Tools and Accessories.....	535

B. AVR ATmega Control Registers.....	549
C. Arduino and Compatible Products Vendors.....	575
D. Recommended Reading.....	581
E. Arduino and AVR Software Development Tools.....	585
Index.....	589

Preface

Since its introduction in 2005 the Arduino has become one of the most successful (some might argue the most successful) open source hardware projects in the world. Boards based on the open designs released by the Arduino team have been fabricated in countries around the world, including Italy, Brazil, China, the Netherlands, India, and the United States. One can purchase a fully functional Arduino-compatible board for around \$15, and the Arduino development environment is readily available for download and is completely free. Originally based on the 8-bit AVR family of microcontrollers (the AVR is itself an interesting device with an equally interesting history), the Arduino has moved into the realm of 32-bit processing with the addition of the Due model with an ARM processor, the Yún with an on-board module running the OpenWrt version of Linux, and the upcoming Zero model. Arduinos have been used for everything from interactive art to robotics, and from environmental sensors to the smarts in small “CubeSat” satellites built by small teams and launched for a fraction of what a full-size satellite would cost.

I bought my first Arduino (a Duemilanove) many years ago, more out of curiosity than anything else. I had worked with microprocessor and microcontroller development systems since the early 1980s, starting with the 6502, 6800, and 8051, and then moving on to the 8086, the Z80, the 80186, and the 68000 family. Early on I usually programmed these devices in assembly language or PL/M, since these were really the only rational choices at the time for embedded systems. Later it became feasible to use C or Ada, as the capabilities of the microprocessors improved and the software tools matured. In each case, however, I came to expect having loads of reference material available: datasheets, hefty manuals, handbooks, and reference design documentation that would arrive along with the development circuit board and its accessories. It usually showed up in a large, heavy box.

When my new Arduino arrived and I opened the very small box I found that there was only a circuit board, a plug-in power pack, a few LEDs and resistors, some jumper wires, and a solderless breadboard block. No manuals, no handbooks, and no datasheets. Not even a CD with documents and software on it. Nothing more than a few sheets of paper with a manifest of what was in the box and a URL to a web page where I could read some “how to get started” material and find links to the software I needed. I was, to say the least, surprised.

I was also ignorant. When I bought the Arduino I didn’t know its full backstory, nor was I aware of its intended audience. It turns out that it was originally meant primarily for people with little or no hardcore technical background who just wanted to experiment (in a playful sense) with something cool and make things go. In other words, artists and tinkerers, not engineers who have a penchant for technical details and an addiction to project plans, specifications, and, of course, manuals.

Once I understood this, it all made a lot more sense. Reading Massimo Banzi’s book *Getting Started with Arduino* (O’Reilly) gave me a better understanding of the Arduino philosophy, and it was a good starting place in my quest for additional details. Also, unlike semiconductor manufacturers with their development kits, the folks on the Arduino team aren’t in the business of selling chips—they’re working to inspire creativity. The AVR microcontroller was chosen because it was inexpensive and it could be easily integrated into their vision for a device that could be readily applied to a creative endeavor. The AVR has enough computational “horsepower” and sufficient built-in memory to do complex and interesting things, unlike earlier generations of microcontrollers that usually needed expensive development tools and provided only scant amounts of internal on-chip memory.

Simplicity and low cost aside, the real secret to the success of the Arduino is the firmware bootloader on the AVR chip, coupled with a simple and easy-to-use integrated development environment (IDE) and the code libraries supplied with it—all provided free under open source and Creative Commons licensing. The Arduino philosophy is to make it as easy as possible to use an Arduino. By clearing away the bulk of the technical details and simplifying the development process, the Arduino invites the user to experiment, try new things, and yes, play. For the first time in a long time I found myself actually having a lot of fun just connecting things in different combinations to see what I could make it do. I wish that the Arduino had been around when I was teaching introductory embedded systems design—it would have helped reduce a lot of frustration for the people in the class trying to wade through assembly language listings, memory maps, and flowcharts.

Since my first Arduino arrived I’ve found many sources for useful and interesting add-on components for the Arduino family, some of them quite amazing in terms of

both price and capabilities.¹ In fact, I became something of an Arduino pack rat, buying inexpensive shields and modules and building up a sizable collection of various bits. But, sadly, I have to say that many times I've opened a package with a nifty new gizmo in it, only to discover that there is no documentation of any kind. Not even a simple wiring diagram.

As an engineer, it is particularly frustrating to me to purchase something interesting, only to have it show up with no documentation. I then have to embark on a quest to determine if any documentation actually does exist, and in a form that I can read (I can't read Chinese). Sometimes that search is fruitless, and I'm forced to resort to component datasheets and reverse engineering the circuit board to figure out the wiring. Other times the information I'm seeking does exist, but it is scattered across multiple websites in various bits and pieces. This situation is slowly improving, but it can still be a real pain. After years of collecting stacks of notes, web page links, and datasheets, I finally decided to get organized and assemble it in one place.

So, what does this book have that can't be found somewhere on the Internet? Not that much, to be perfectly honest, but hopefully what it will do is reduce a lot of the potential frustration and wasted time—and of course there are all the bits that I've discovered on my own. The official technical data comes from manufacturers such as Atmel, the Arduino team, and numerous other sources, both well known and obscure. Some overseas vendors do have at least rudimentary websites, whereas others have very nice web pages with links to other sites for the documentation. This book contains as much of the essential data as I could find or reverse engineer, all in one convenient place, with as many of the holes plugged as I could manage. I wanted to save others the frustration I've experienced trying to run down some mundane little technical detail about the USB interface, or figure out why a shield wasn't working correctly, or why that really neat sensor I bought from someone through eBay didn't seem to work at all.

The result of my frustrations is this book, the one I've been wanting for working with the Arduino boards and shields. I really wanted something physical that I could keep near at hand on my workbench. It isn't always practical to have to refer to a web page to look something up, and to make things even more interesting, sometimes access to the Internet just isn't available (like when you're trying to debug a remote data logging device on a mountaintop with just a little netbook for company and no wireless service for 60 miles in any direction). I wanted something that I could use to quickly

¹ Entering “Arduino” into the search field on eBay will return a multitude of things like ultrasonic range sensors, temperature and humidity sensors, various Arduino clones, Bluetooth and ZigBee shields, and much more. But, unfortunately, some of the items come with little or no documentation, and even if there is some it may not be particularly up-to-date or accurate. That doesn't mean you shouldn't consider these sellers (the prices are usually excellent and the build quality is generally very good), but as always when buying things online, caveat emptor.

look up the answer to a question when working with the Arduino and associated add-on components, no matter where I was. As far as I know, such a thing didn't exist until now. I hope you find it as useful as I have as I assembled my notes for this book.

Intended Audience

This book is intended for those people who need or want to know the technical details. Perhaps you've gone as far as you can with the introductory material and the "99 amazing projects" type of books, and you now need to know how to do something novel and unique. Or, you might be a working engineer or researcher who would like to incorporate an Arduino into your experimental setup in the lab. You might even be someone who wants to install an Arduino into an RC airplane, use one as part of a DIY weather station,² or maybe do something even more ambitious (a CubeSat, perhaps?).

Ideally you should have a basic knowledge of C or C++, some idea of how electrons move around in a circuit, and some experience building electronic devices. If I may be so bold, I would suggest that you have a copy of my book *Practical Electronics: Components and Techniques* (also from O'Reilly) on hand, along with some reference texts on programming and electronics (see Appendix D for some suggestions).

What This Book Is

This book is a reference and a handbook. I have attempted to organize it such that you can quickly and easily find what you are looking for. I have included the sources of my information whenever possible, and those insights that are the result of my own investigations are noted as such.

What This Book Is Not

This book is not a tutorial. That is not its primary purpose. I don't cover basic electronics, nor is there any discussion of the dialect of the C++ language that is used to create the so-called "sketches" for programming an Arduino. There are some excellent tutorial texts available that cover general electronics theory and programming, and I would refer the reader to those as a place to get started with those topics.

This book is also not an Arduino-sanctioned guide to the products produced by the Arduino team. It is based on information gleaned from various sources, some more

² The books *Environmental Monitoring with Arduino* and *Atmospheric Monitoring with Arduino* (O'Reilly), both by Emily Gertz and Patrick Di Justo, offer some good ideas for doing just this with cheap and readily available sensors and an Arduino.

obscure than others, along with my own notes and comments based on my experiences with the Arduino. As such, I am solely responsible for any errors or omissions.

About Terminology

The distinctions between processors, microprocessors, and microcontrollers arose sometime in the early 1980s as manufacturers were trying to distinguish their products based on size and amount of external circuitry required for the devices to do something useful. Full-size mainframe processors and the smaller microprocessors like those found in desktop PCs both typically require some external components (sometimes a lot of them) in order to be useful. A microcontroller, on the other hand, has everything it needs to do its job already built in. Also, a microprocessor will typically support external memory, whereas a microcontroller may have only limited support (if any at all) for adding additional memory to what is already on the chip itself.

Throughout this book I will use the terms “microcontroller” and “processor” interchangeably. Although “microcontroller” might be considered to be technically more correct, in my mind it is still a processor of data, just a smaller version of the huge machines I once worked with in the distant past. They all do essentially the same thing, just at different scales and processing speeds.

What's in This Book

[Chapter 1](#) presents a brief history of the Arduino in its various forms. It also introduces the AVR microcontrollers used in the Arduino boards, and discusses the differences between software-compatible and hardware-compatible products based on the Arduino.

The Atmel AVR microcontroller is the subject of [Chapter 2](#). This is intended as an overview of what is actually a very sophisticated device, and so this chapter provides a quick tour of the highlights. This includes the timer logic, the analog comparator, the analog input, the SPI interface, and other primary subsystems on the chip.

[Chapter 3](#) takes a closer look at the AVR microcontrollers used in Arduino boards, namely the ATmega168/328, the ATmega1280/2560, and the ATmega32U4 devices. It builds on the overview presented in [Chapter 2](#), and provides additional low-level details such as internal architecture, electrical characteristics, and chip pinouts.

[Chapter 4](#) covers the physical characteristics and interface functions of various Arduino boards. This includes the USB interface types, printed circuit board (PCB) dimensions, and board pinout diagrams.

What really makes the Arduino unique is its programming environment, and that is the subject of [Chapter 5](#). This chapter defines the Arduino sketch concept and how it utilizes the C and C++ languages to create sketches. It introduces the Arduino boot-

loader and the Arduino `main()` function. This chapter also describes how you can download the Arduino source code and see for yourself what it looks like under the hood.

[Chapter 6](#) describes the AVR-GCC toolchain and presents techniques for programming an Arduino board without using the Arduino IDE. It also covers makefiles and includes a brief overview of assembly language programming. The chapter wraps up with a look at the tools available to upload code into an AVR.

The focus of [Chapter 7](#) is on the standard libraries supplied with the Arduino IDE. The Arduino IDE is supplied with numerous libraries, and more are being added all the time. If you want to know if a library module exists for a particular sensor or for a specific operation, then this is a good starting point.

[Chapter 8](#) presents the various types of shields available for the Arduino. It covers many of the commonly available types, such as flash memory, prototyping, input/output, Ethernet, Bluetooth, ZigBee, servo control, stepper motor control, LED displays, and LCD displays. It also covers using multiple shields, and presents some hints and tips for getting the most from a shield.

[Chapter 9](#) describes some of the various add-on components available that can be used with an Arduino. These include sensors, relay modules, keypads, and other items that aren't specific to the Arduino, but work with it quite nicely. Electrical pin-out information and schematics are provided for many of the items discussed.

Sometimes there just isn't a readily available shield to do what needs to be done. [Chapter 10](#) describes the steps involved in creating a custom shield. It also describes how to use an AVR microcontroller without an Arduino-type circuit board but still take advantage of the Arduino IDE.

Chapters [11](#), [12](#), and [13](#) cover some projects that illustrate the capabilities of the AVR microcontrollers and the Arduino boards. They are intended to demonstrate how an Arduino can be applied in various situations, not as how-to guides for building a board or device. You can, however, build any of the items described yourself, if you feel so inclined, and they might serve as jumping-off points for your own projects. Each example project description includes theory of operation, schematics, a detailed parts list, PCB layouts (if required), and an overview of the software necessary to make it go.

Because the main emphasis of this book is on the Arduino hardware and related modules, sensors, and components, the software shown is intended only to highlight key points; my aim was not to present complete, ready-to-run examples. The full software listings for the examples and projects can be found on [GitHub](#).

[Chapter 11](#) describes a basic programmable signal generator, a handy thing to have around when working with electronic circuits. With this project you can generate

pulses at various duty cycles, output a sequence of pulses in response to a trigger input, generate sine waves, and also create programmable patterns of pulses.

[Chapter 12](#) covers the design and construction of a “smart” thermostat suitable for use with a home HVAC (heating, ventilation, and air conditioning) system. Instead of paying for something that is already built and sealed into a plastic case, you can build it yourself and program it to behave exactly the way you want it to. I’ll show you how to incorporate more than just a temperature sensor: features include multiple temperature and humidity sensors, and the use of your HVAC system’s fan to provide a comfortable environment without the cost of running the compressor or lighting up the heater.

In [Chapter 13](#) we will look at how to build an automatic model rocket launcher with a programmable sequencer and automatic system checks. Even if you don’t happen to have a model rocket handy, this project describes techniques that can be applied to many types of sequentially controlled processes, be it on a production line or a robotic material handling device in a laboratory.

[Appendix A](#) is an overview of the basic tools and accessories you may need if you want to go beyond prefabricated circuit boards and modules.

[Appendix B](#) is a compilation of the control registers for the ATmega168/328, the ATmega1280/2560, and the ATmega32U4 microcontrollers.

[Appendix C](#) is a summary listing of the Arduino and compatible products distributors and manufacturers listed in this book. It is by no means exhaustive, but hopefully it will be enough to get you started on your quest to find what you need.

[Appendix D](#) lists some recommended books that cover not just the Arduino, but also C and C++ programming and general electronics.

Finally, [Appendix E](#) is a summary of some of the readily available Arduino and AVR software development tools that are currently out there.

Endorsements

Other than references to the Arduino team and the folks at Arduino.cc, there aren’t any specific endorsements in this book—at least, not intentionally. I’ve made reference to many different component manufacturers, suppliers, and other authors, but I’ve tried to be evenhanded about it, and I don’t specifically prefer any one over another. My only criteria in selecting those I do mention are that I own one or more of their products and that I’ve successfully managed to use a shield, module, sensor, or Arduino main PCB (or clone PCB in some cases) from the supplier in something, even if just in a demonstration of some type. Any trademarks mentioned are the property of their respective owners; they appear here solely for reference. As for the photography, I tried to use my own components, tools, circuit boards, modules, and

other items as much as possible, and although an image may show a particular brand or model, that doesn't mean it's the only type available—it likely just happens to be the one that I own and use in my own shop. In some cases I've used images with permission from the vendor or creator, works in the public domain, or images with a liberal Creative Commons (CC) license, and these are noted and credited as appropriate. I created all the diagrams, schematics, and other nonphotographic artwork, and I am solely responsible for any errors or omissions in these figures.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Constant width

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.



This element signifies a tip or suggestion.



This element signifies a general note.



This element indicates a warning or caution.

Safari® Books Online



Safari Books Online is an on-demand digital library that delivers expert **content** in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of **product mixes** and pricing programs for **organizations**, **government agencies**, and **individuals**. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens **more**. For more information about Safari Books Online, please visit us [online](#).

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at <http://bit.ly/arduino-a-technical-reference>.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

For more information about our books, courses, conferences, and news, see our website at <http://www.oreilly.com>.

Find us on Facebook: <http://facebook.com/oreilly>

Follow us on Twitter: <http://twitter.com/oreillymedia>

Watch us on YouTube: <http://www.youtube.com/oreillymedia>

Acknowledgments

This book would not have been possible without the enduring patience and support of my family. Writing appears to be habit-forming, but they've been very encouraging and supportive, and will even bring me things to eat and occasionally check to make sure I'm still alive and kicking in my office. It doesn't get much better than that. I would particularly like to thank my daughter Seren for her photographic assistance and help in keeping my collection of various bits and pieces catalogued and organized.

I would also like to thank the editorial staff at O'Reilly for the opportunity to work with them once again. As always, they have been helpful, patient, and willing to put up with me. Special thanks goes to Brian Sawyer and Dawn Schanafelt for their excellent editorial support and guidance, and to Mike Westerfield for his insightful technical review of the material.

CHAPTER 1

The Arduino Family

This chapter provides a brief history of the Arduino, along with a terse genealogy of the various board types created since 2007. It doesn't cover boards produced before 2007, nor does it attempt to be comprehensive in its coverage of the various clones and derivatives that have been produced. The main focus here is on the differences between the various primary types of Arduino boards, with a specific focus on the types of processors used and the physical design of the boards. It also takes a quick look at the range of possible applications for Arduino circuit boards.

[Chapter 2](#) provides general information about the internal functions of the Atmel AVR processors, and [Chapter 3](#) covers the specific processors used in Arduino boards. With the exception of the Yún, [Chapter 4](#) describes the physical characteristics of different official Arduino boards introduced in this chapter.

A Brief History

In 2005, building upon the work of Hernando Barragán (creator of Wiring), Massimo Banzi and David Cuartielles created Arduino, an easy-to-use programmable device for interactive art design projects, at the Interaction Design Institute Ivrea in Ivrea, Italy. David Mellis developed the Arduino software, which was based on Wiring. Before long, Gianluca Martino and Tom Igoe joined the project, and the five are known as the original founders of Arduino. They wanted a device that was simple, easy to connect to various things (such as relays, motors, and sensors), and easy to program. It also needed to be inexpensive, as students and artists aren't known for having lots of spare cash. They selected the AVR family of 8-bit microcontroller (MCU or µC) devices from Atmel and designed a self-contained circuit board with easy-to-use connections, wrote bootloader firmware for the microcontroller, and packaged it all into a simple integrated development environment (IDE) that used programs called "sketches." The result was the Arduino.

Since then the Arduino has grown in several different directions, with some versions getting smaller than the original, and some getting larger. Each has a specific intended niche to fill. The common element among all of them is the Arduino runtime AVR-GCC library that is supplied with the Arduino development environment, and the on-board bootloader firmware that comes preloaded on the microcontroller of every Arduino board.

The Arduino family of boards use processors developed by the Atmel Corporation of San Jose, California. Most of the Arduino designs utilize the 8-bit AVR series of microcontrollers, with the Due being the primary exception with its ARM Cortex-M3 32-bit processor. We don't cover the Due in this book, since it is radically different from the AVR devices in many fundamental ways and really deserves a separate discussion devoted to it and similar microcontrollers based on the ARM Cortex-M3 design.

Although an Arduino board is, as the Arduino team states, just a basic Atmel AVR development board, it is the Arduino software environment that sets it apart. This is the common experience for all Arduino users, and the cornerstone of the Arduino concept. [Chapter 5](#) covers the Arduino IDE, the libraries supplied with the IDE, and the bootloader. [Chapter 6](#) describes the process of creating software for an AVR MCU without using the Arduino IDE.

Types of Arduino Devices

Over the years the designers at Arduino.cc have developed a number of board designs. The first widely distributed Arduino board, the Diecimila, was released in 2007, and since its initial release the Arduino family has evolved to take advantage of the various types of Atmel AVR MCU devices. The Due, released in 2012, is the first Arduino to utilize a 32-bit ARM Cortex-M3 processor, and it breaks from the rest of the family in terms of both processing power and board pinout configuration. Other boards, like the LilyPad and the Nano, also do not have the same pinout as the other members of the family, and are intended for a different range of applications—wearables in the case of the LilyPad; handheld devices for the Esplora; and compact size in the case of the Mini, Micro, and Nano.

With each passing year new types of Arduino boards appear, so what is listed here may be out of date by the time you're reading it. The newer versions have more advanced processors with more memory and enhanced input/output (I/O) features, but for the most part they use the same pinout arrangements and will work with existing add-on boards, called *shields*, and various add-on components such as sensors, relays, and actuators. [Table 1-1](#) lists the Arduino types that have appeared since 2005. The newer versions of the Arduino will also run most of the sketches created for older models, perhaps with a few minor tweaks and newer libraries, but sketches written for the latest versions may or may not work with older models.

Table 1-1 is not a buyer's guide. It is provided to give a sense of historical context to the Arduino. As you can see, the years 2007 and 2008 saw the introduction of the LilyPad; the small form-factor boards like the Nano, Mini, and Mini Pro; and the introduction of the Duemilanove as a natural evolutionary step based on the Diecimila. While there are no significant physical differences between the Diecimila and the Duemilanove, the Duemilanove incorporates some refinements in the power supply, most notably in its automatic switchover between USB power and an external DC (direct current) power supply. Later versions of the Duemilanove also utilize the ATmega328 MCU, which provides more memory for programs.

Table 1-1 doesn't include the Arduino Robot, which is a PCB with motors and wheels attached. One of the newest boards in the Arduino lineup is the Yún, an interesting beast that has both an ATmega32U4 microcontroller and a Linino module with an Atheros AR9331 MIPS-based processor capable of running a version of the Linux-based OpenWrt operating system. I won't get into the OpenWrt end of the Yún, but the Arduino side is basically just a standard Arduino (a Leonardo, to be specific). If you want to learn more about the Yún, I would suggest checking it out on the [Arduino website](#).

Table 1-1. Timeline of Arduino products

Board name	Year	Microcontroller	Board name	Year	Microcontroller
Diecimila	2007	ATmega168V	Mega 2560	2010	ATmega2560
LilyPad	2007	ATmega168V/ATmega328V	Uno	2010	ATmega328P
Nano	2008	ATmega328/ATmega168	Ethernet	2011	ATmega328
Mini	2008	ATmega168	Mega ADK	2011	ATmega2560
Mini Pro	2008	ATmega328	Leonardo	2012	ATmega32U4
Duemilanove	2008	ATmega168/ATmega328	Esplora	2012	ATmega32U4
Mega	2009	ATmega1280	Micro	2012	ATmega32U4
Fio	2010	ATmega328P	Yún	2013	ATmega32U4 + Linino

When more than one microcontroller type is shown in **Table 1-1**, it indicates that a particular version of an Arduino board was made initially with one microcontroller, and later with the other (usually more capable) device. For example, an older version of the Duemilanove will have an ATmega168, whereas newer models have the ATmega328. Functionally the ATmega168 and the ATmega328 are identical, but the ATmega328 has more internal memory.

The latest additions to the Arduino family, the Leonardo, Esplora, Micro, and Yún, all use the ATmega32U4. While this part is similar to an ATmega328 it also incorporates an integrated USB-to-serial interface component, which eliminates one of the integrated circuit (IC) parts found on boards like the Uno and Duemilanove.

The programming interface also behaves slightly differently with the boards that use the ATmega32U4, but for most people this should be largely transparent. [Chapter 2](#) describes the general functionality of AVR microcontrollers, [Chapter 3](#) contains descriptions of the specific AVR MCU types found in Arduino devices, and [Chapter 4](#) provides descriptions of the primary Arduino circuit boards and their pinout definitions.

Arduino Galleries

Tables 1-1 through 1-5 show some of the various types of Arduino boards, both past and present. It is not completely inclusive, since new types and updates to existing types occur periodically. The following images show the wide diversity in physical shapes and intended applications of the Arduino.

Physically, an Arduino is not a large circuit board. The baseline boards, which have the physical pin arrangement commonly used for add-on boards (called shields, described in [Chapter 8](#)), are about 2.1 by 2.7 inches (53.3 by 68.6 mm) in size. [Figure 1-1](#) shows a selection of Arduino boards with a ruler for scale, and [Figure 1-2](#) shows a Nano mounted on a solderless breadboard.

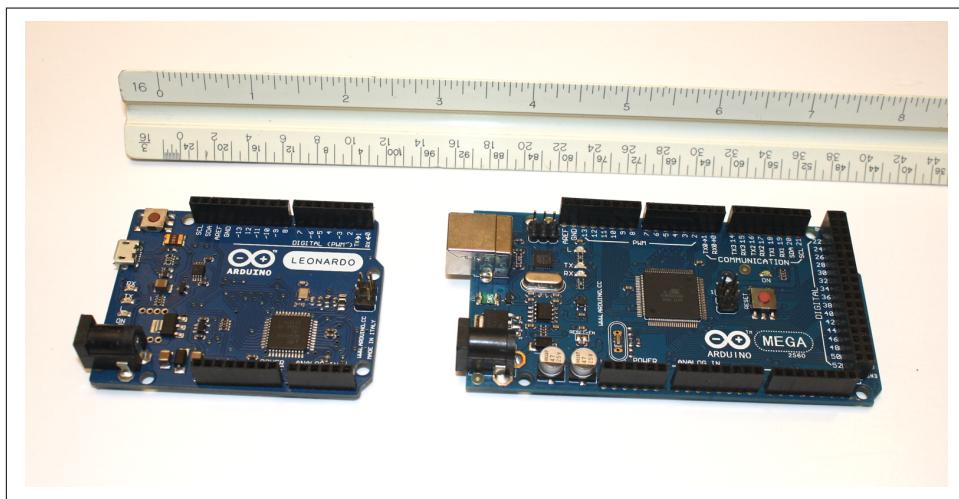


Figure 1-1. Relative sizes of Arduino boards

[Chapter 4](#) contains reference drawings with dimensions and pin definitions for most of the common Arduino boards. Note that while it is small, a board like the Nano has all of the same capabilities as a Duemilanove, except for the convenient pin sockets and regular (type B) USB connector. It is ideal for applications where it will not be disturbed once it is installed, and where small size is a requirement. Some applications that come to mind are autonomous environmental data collection devices

(automated solar-powered weather data stations or ocean data collection buoys, for example), timing and data collection for model rockets, security systems, and perhaps even a “smart” coffee maker.

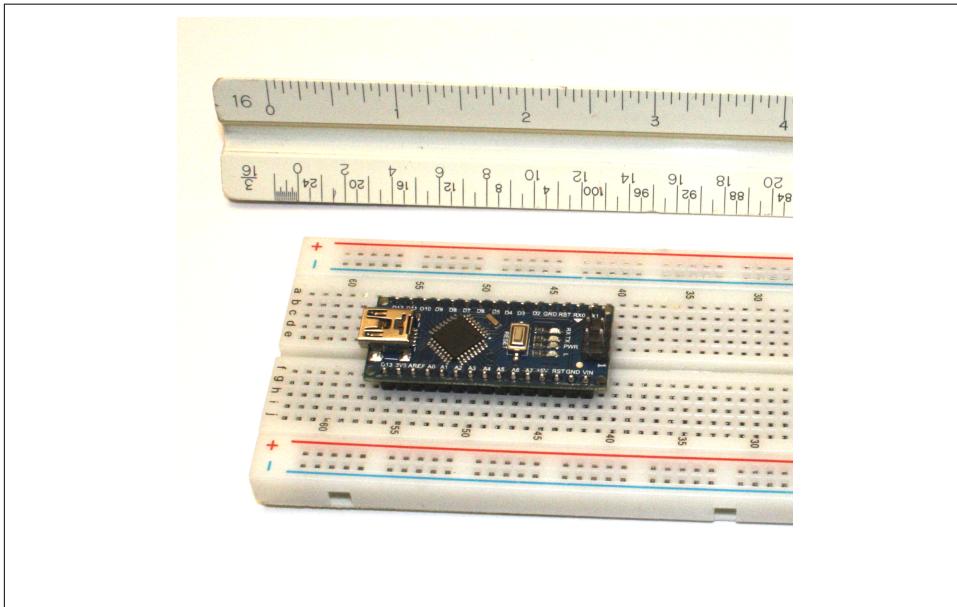
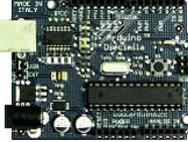
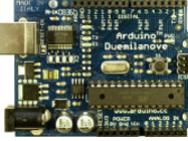


Figure 1-2. An Arduino Nano on a solderless breadboard

Table 1-2. Baseline layout of Arduino boards

Type	Year introduced
Diecimila	2007
	
Duemilanove	2008
	
Uno (R3 version)	2010
	

Type	Year introduced
	Ethernet 2011
	Leonardo 2012

Table 1-3. Mega layout of Arduino boards

Type	Year introduced
	Mega 2009
	Mega 2560 2009
	Mega ADK 2011

Table 1-4. Small form factor Arduino boards

Type	Year introduced
	Nano 2008
	Mini 2008
	Fio 2010
	Micro 2012

Table 1-5. Special form factor Arduino boards

Type	Year introduced
LilyPad	2007
Esplora	2012

Arduino-Compatible Devices

In addition to the various board types designed or sanctioned by Arduino.cc, there are many devices that are either hardware compatible or software compatible. What makes these devices Arduino compatible is that they incorporate the Arduino bootloader (or something that works like it), and they can be programmed with the Arduino IDE by selecting the appropriate compatible Arduino board type from the IDE's drop-down list.

Hardware-Compatible Devices

An Arduino hardware-compatible device is one where the various I/O pins on the board have been arranged to match one of the existing Arduino form factors. A hardware-compatible board can (usually) accept any of the shields and plug-in modules created for an official Arduino board. The reasons behind this are covered in “[The Arduino Naming Convention](#)” on page 9.

In most cases hardware-compatible boards look like any other Arduino board, except that the official Arduino logo and silkscreen graphics are missing. Other hardware-compatible products might not look anything like a typical Arduino board, but do provide the pin sockets in the correct arrangement for using a standard Arduino-type shield board. Some hardware-compatible products include additional connections, like the SainSmart version of the Uno with additional connectors for I/O functions. [Table 1-6](#) lists a few Arduino clones and compatible boards that are available. There are many more than what is shown here, but this should give some idea of what is available.

Table 1-6. Arduino hardware-compatible devices

Name	Type	Origin
 SainSmart Uno	Uno clone	China
 SainSmart Mega2560	Mega 2560 clone	China
 Brasuino	Similar to the Uno, with minor changes	Brazil
 Diavolino	An Arduino layout-compatible clone kit	USA

Note that Diavolino is a kit and requires assembly.

Software-Compatible Devices

There are many Arduino software-compatible boards available. These utilize the Arduino bootloader and development environment, but do not have a completely Arduino-compatible physical form factor. Software-compatible devices can be programmed with the Arduino development tools, but may use a different arrangement of I/O pins, or perhaps use some other types of connectors in place of the pin sockets found on stock Arduino boards. Custom circuits based on an AVR microcontroller and built into some larger device or system would fall into the software-compatible category if the Arduino bootloader is installed in the microcontroller.

The core of the Arduino is the processor and the preinstalled bootloader. Using that definition, one could have just a bare ATmega AVR IC with the Arduino firmware loaded into it. It could then be used with a solderless breadboard and the Arduino development environment. AVR MCU ICs with preloaded bootloader code are available for purchase from multiple sources, or you could do it yourself. [Chapter 5](#) describes the steps necessary to load an AVR MCU with the Arduino bootloader firmware.

It is interesting to note that some of the boards from Arduino, such as the Mini, Micro, Nano, LilyPad, and Esplora, are not hardware compatible in terms of using the “standard” I/O connector layout. They can’t be used directly with a conventional shield, but they are still Arduino boards, and they are supported by the Arduino IDE.

The Boarduino from Adafruit Industries is one example of an Arduino software-compatible device. This board is designed to mount on a standard solderless breadboard much like a full-size 40-pin IC. It is available in two styles: DC and USB. The DC version does not have an on-board USB chip, so an external USB adapter is needed to program it. Another example of a software-compatible board is the Dragonfly from Circuit Monkey, which uses standard Molex-type connectors instead of the pins and sockets used on a conventional Arduino. It is intended for high-vibration environments, such as unmanned aerial vehicles (UAVs) and robotics.

The Raspduino is designed to mount onto a Raspberry Pi board, and it is functionally equivalent to an Arduino Leonardo. This results in a combination that is roughly equivalent to the Yún, but not exactly the same. Each setup has its own strengths and weaknesses. **Table 1-7** lists a few Arduino software-compatible boards.

Table 1-7. Arduino software-compatible devices

Name	Description	Origin
	Boarduino DC Designed to fit on a solderless breadboard	USA
	Boarduino USB Designed to fit on a solderless breadboard	USA
	Dragonfly Uses Molex-type connectors for I/O	USA
	Raspduino Designed to fit on a Raspberry Pi board	Netherlands

This is just a small selection of the various boards that are available. Because the AVR microcontroller is easy to integrate into a design, it has found its way into numerous applications. With the Arduino bootloader firmware, programming a device is greatly simplified and the design possibilities are vast.

The Arduino Naming Convention

While the circuit design and software for the Arduino are open source, the Arduino team has reserved the use of the term “Arduino” for its own designs, and the Arduino

logo is trademarked. For this reason you will sometimes find things that behave and look like official Arduino devices, but which are not branded Arduino and have not been produced by the Arduino team. Some of them use “-duino” or “-ino” as part of the product name, such as Freeduino, Funduino, Diavolino, Youduino, and so on. Some, like the boards made by SainSmart, use just the model name (Uno and Mega2560, for example).



At the time of this writing, there was an ongoing dispute between the company created by the original founders (Arduino LLC) and a different company started by one of the original founders (Arduino SRL). As a result, Arduino LLC uses the trademark Arduino within the United States and Genuino elsewhere.

Occasionally someone will produce a board that claims to be an Arduino, but is in fact just a copy that uses the Arduino trademark without permission. The silkscreen mask used to put the logo and other information on an official Arduino is also copyrighted, and the Arduino folks don’t release the silkscreen with the PCB layout files. Massimo Banzi has [a section of his blog](#) devoted specifically to these unauthorized boards, and his examination of blatant and shameless copies is interesting, to say the least. Just search for the “hall of shame” tag.

The bottom line here is that you are welcome to copy the schematics, the bootloader code, and the Arduino IDE, and use these to create your own version of an Arduino. It is, after all, open source. Just don’t call it an Arduino or use the artwork from Arduino.cc without permission.

What Can You Do with an Arduino?

In addition to the ease of programming made possible by the Arduino IDE, the other big feature of the Arduino is the power and capability of the AVR microcontroller it is based on. With a handful of readily available add-on shields (described in [Chapter 8](#)) and a wide selection of low-cost sensor and actuator modules (these are described in detail in [Chapter 9](#)), there really isn’t a whole lot you can’t do with an Arduino provided that you keep a few basic constraints in mind.

The first constraint is memory. The AVR MCU just doesn’t have a whole lot of memory available for program storage and variables, and many of the AVR parts don’t have any way to add more. That being said, the ATmega32 and ATmega128 types can use external memory, but then the I/O functions for those pins are no longer readily available. Arduino boards were not designed to accommodate external memory, since one of the basic design assumptions was that the AVR chip itself would have the necessary I/O and that the user would be running a relatively short program. The Arduino was not intended to be a replacement for a full-on computer system with

gigabytes of RAM and a hard disk drive (HDD). There are inexpensive Intel-based single-board computers that fit that description, but they won't fit into an old mint tin, a section of PVC tubing strapped to a pole or a tree, a small robot, or the payload section of a model rocket. An Arduino will.

The second constraint is speed. The Arduino CPU clock rate is typically between 8 and 20 MHz (see [Chapter 4](#) for a detailed comparison of Arduino AVR device types). While this may sound slow, you should bear in mind two key facts: first, the AVR is a very efficient RISC (reduced instruction set computer) design, and second, things in the real world generally don't happen very quickly from a microcontroller's perspective. For example, how often does a so-called smart thermostat need to sample the temperature in a home or office? Once a second is probably overkill, and once every 5 or even 10 seconds will work just fine. How often does a robot need to emit an ultrasonic pulse to determine if there is an obstacle ahead? A pulse every 100 ms is probably more than enough (unless the robot is moving very, very fast). So, for an Arduino running at 16 MHz (like the Leonardo, for example), there will be on the order of 1,000,000 or more CPU clock ticks between sensor pulses, depending on whatever else the CPU is doing with the pulses. Given that an AVR can execute many instructions in one or two clock cycles, that's a lot of available CPU activity in between each pulse of the ultrasonic sensor.

The third main constraint is electrical power. Since the Arduino hardware is actually nothing more than a PCB for an AVR IC to sit on, there is no buffering between the microcontroller and the external world. You can perform a fast "charcoal conversion" of an AVR (in other words, overheat the IC and destroy it) if some care isn't taken to make sure that you aren't sourcing or sinking more current than the device can handle. Voltage is also something to consider, since some of the AVR types have 3.3V I/O, whereas others are 5V tolerant. Connecting 5V transistor-transistor logic (TTL) to a 3.3V device usually results in unhappy hardware, and the potential for some smoke.

With the preceding constraints in mind, here are just a few possible applications for an Arduino:

- Real-world monitoring
 - Automated weather station
 - Lightning detector
 - Sun tracking for solar panels
 - Background radiation monitor
 - Automatic wildlife detector
 - Home or business security system
- Small-scale control
 - Small robots

- Model rockets
- Model aircraft
- Quadrotor UAVs
- Simple CNC for small machine tools
- Small-scale automation
 - Automated greenhouse
 - Automated aquarium
 - Laboratory sample shuttle robot
 - Precision thermal chamber
 - Automated electronic test system
- Performance art
 - Dynamic lighting control
 - Dynamic sound control
 - Kinematic structures
 - Audience-responsive artwork

In Chapters 11, 12, and 13 we will look at applications such as a smart thermostat, a programmable signal generator, and an automated rocket launch control system to help fulfill your suborbital yearnings. These are just the tip of the iceberg. The possibilities are vast, and are limited only by your imagination. So long as you don't try to make an Arduino do the job of a full-on computer system, you can integrate one into all sorts of interesting applications—which is exactly what the folks at Arduino.cc want you to do with it.

For More Information

The boards listed in this chapter are just a small selection of what is available, and there is much more to the story of the Arduino. Entering “Arduino” into Google’s search bar will produce thousands of references to explore.

The official Arduino website can be found at <http://www.arduino.cc>.

Massimo Banzi’s blog is located at <http://www.massimobanzi.com>.

Also, check the appendixes for more website links and book recommendations.