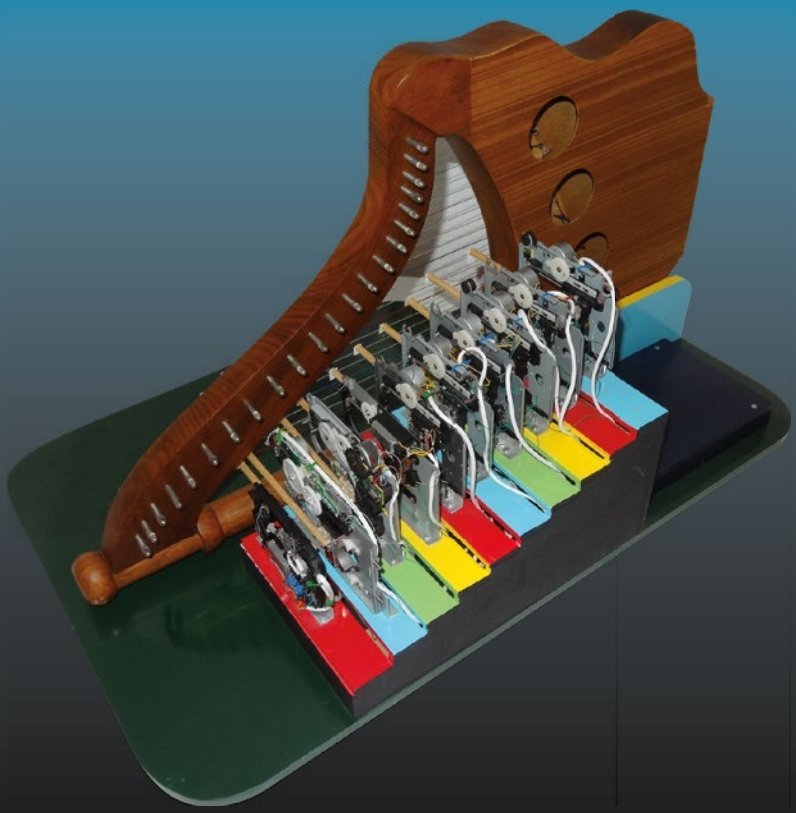
TECHNOLOGY IN ACTION™



Arduino Music and Audio Projects

Mike Cook

Foreword by Massimo Banzi and David Cuartielles, Founders of the Arduino Project

Apress®

Arduino Music and Audio Projects

Mike Cook

Foreword by Massimo Banzi and David Cuartielles,
Founders of the Arduino Project

Apress®

Arduino Music and Audio Projects

ISBN-13 (pbk): 978-1-4842-1720-7

ISBN-13 (electronic): 978-1-4842-1721-4

Managing Director: Welmoed Spahr
Lead Editor: Michelle Lowman
Editorial Board: Steve Anglin, Pramila Balan, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Michelle Lowman, James Markham, Susan McDermott, Matthew Moodie, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing
Coordinating Editor: Mark Powers
Copy Editor: Kezia Endsley
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

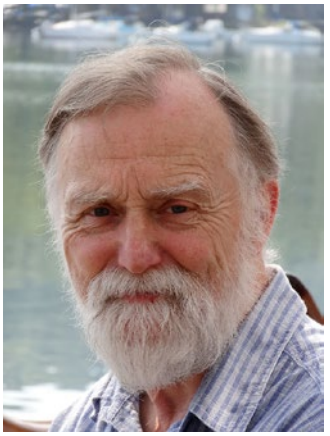
Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com/9781484217207. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

This book is dedicated to my lifelong friend Arthur Broughton. When you are at school you might expect to make friends that last a lifetime but you seldom expect that friend to be the father of a friend. He introduced me to the delights of good beer, good music and a wonderful sense of humor. He is what you would call an old fashioned gentleman. Now in his nineties he remains active of mind and very much like he was when I first met him.

Contents at a Glance

Contents

About the Author



Mike Cook has been making electronic things since he was at school in the 60s. He would have been a chemist except his school had no chemistry lab so he turned his attention to physics. He left school at the age of 16 to get a job in electronics. Day release part time study earned him the qualifications required to take a BSc (Hons) in Physical Electronics. Subsequently he did post graduate researched into a novel form of digital sound compression in the mid 70s. This was before the advent of microcomputers, when everything had to be made using logic gates. Then an academic career followed reaching the level of Senior Lecturer (a post known as Professor in the USA) in Physics, at Manchester Metropolitan University. During his time at the University he wrote more than three hundred computing and electronics articles in the pages of UK computer magazines starting in the 1980s for 20 years. Mainly for The Micro User and Acorn Computing, these were magazines devoted to the computers produced by Acorn Ltd, the inventers of the RISC processor.

Leaving the University after 21 years, when the Physics department closed down, he got a series of proper jobs where he designed digital TV set top boxes and access control systems. Now retired and freelancing, he spends his days surrounded by wires, exhibiting at Maker Fairs, and patrolling the Arduino forum as Grumpy Mike.

He is the co-author of three books about the Raspberry Pi, all published by Wiley: *Raspberry Pi for Dummies, First and Second editions* (with Sean McManus); *Raspberry Pi Projects* (with Andrew Robison); and *Raspberry Pi Projects for Dummies* (with Jonathan Evans & Brock Craft). He also has a monthly column in *The MagPi*, an online and print magazine published by the Raspberry Pi foundation. Which is not bad for a dyslexic or, as it was known when he was at school, thick.

Acknowledgments

I would like to thank my editor Mark Powers for his help and encouragement and also Michelle Lowman for believing in, and commissioning this book. I would also like to thank Kezia Endsley for her accurate and sympathetic copy editing, along with all the other professionals at Apress who had a part in the production of this book.

I would also like to thank the team who created the amazing Arduino project, Massimo Banzi, David Cuartielles, Tom Igoe and David Mellis without whose inspiration the modern maker movement would not be what it is today.

Foreword

We first met "Grumpy Mike" Cook at the Arduino forum, where we spent long hours giving support to people from all over the world. We created special categories for our users depending on the amount of answers they would write to the questions popping up in the forum. For awhile, you would be listed as "God" if you had answered over 500 posts in the forum and someone, already in 2010, defined the "Grumpy Mike" level, since he had over 9,000 posts answered.

We began adding new levels using names of scientists and technologists, and Grumpy Mike quickly reached the highest level—Brattain—the one reserved for those having answered over 15,000 posts.

Besides his amazing capability to help people, Grumpy Mike builds his own projects. We happened to meet him in person for the first time at Maker Faire Rome 2013, where he displayed his RFID music sequencer. As researchers in Interactive Interfaces, we have to say that it is pretty complex to handle big antenna arrays with multiple RFID readers, but Grumpy Mike solved it without problems.

Not only is he good technically, and at solving complex questions in the forum...he happens not to be grumpy at all! Grumpy Mike is one of the least grumpy people we've ever met in the geek world.

We hope you will enjoy reading his book as much as I enjoy looking through his projects.

—David Cuartielles and Massimo Banzi
Co-Founders, The Arduino Project

Introduction

The Arduino series of controller boards has revolutionized the way that inventors and creators can realize their imaginings. Originally a low cost platform delivering the same computing power as a early 1980 hobby computer with a form factor and price an insignificant fraction of its predecessors. It has grown, over the last ten years, into a range of controller boards with varying form factors and raw computing power. Unhindered by the burden of an operating system every ounce of computing power can be directed towards your own project. It has almost reached the stage that if you can imagine it then the odds are you can do it with one of the Arduinos on offer...

The Arduino has built up a community of dedicated enthusiasts who congregate around the company's official forum. Moderated entirely by unpaid volunteers, it aims to educate and inspire newcomers to the platform. It has developed a collective ethos about behavior and conduct on its pages with an organic consensus. Its overriding concern is that no question is considered too simple or stupid and yet no legal project is off limits. It aims to be beginner friendly and many beginners stop on to help others once they have developed. There is also a section of professional engineers that are ready to dispense wisdom from a long career in the subject. It is amazing what a collection of specialties the collective membership can muster.

One lesser considered aspect of applying Arduinos to projects is when it comes to audio, and it is this that this book aims to correct. Here, in one place, is a grounding in the theory and practice of most aspects of the subject. This book does not pretend to be definitive in all aspects of audio because it would take many volumes to do that, but it will provide a through grounding into most aspects of what you would want to do in a project. The book is split into three parts. Part one looks at high level audio control mainly concerned with the MIDI system and the associated message systems of Open Sound Control (OSC). The second part looks at the direct generation of audio signals from the Arduino. The final third part considers signal processing, that is reading in real time audio data, processing it and outputting it again. Thought the book not only the theory is explained but project examples using this theory are given. These projects range from a quick ten minute demonstration to full blown three month long projects. No matter what the length of the project they can all be freely adapted to put your own spin on them allowing you a stepping stone to creating something totally unique to you.

PART I

MIDI and OSC

The first part of the book looks at the MIDI system universally used for passing data between musical systems. MIDI is not concerned with the sounds themselves; rather it is a system of messages that tell other things to generate the sound. Rather like the difference between sheet music and a tape recorder. OSC (Open Sound Control) is a similar idea brought up to date for networked systems.

The data gathered from a keyboard, in a form that is capable of expressing all the subtitles of a performance, to computer generated notes, these can be expressed as MIDI data. On the receive side sound generators can turn this data into sounds of quality that are as deep as your pocket. As well as music, other things can be controlled by MIDI like lights or motors. This part of the book explores in depth the wide possibilities that are on offer with MIDI and other forms of messages.

CHAPTER 1

Basic Arduino

This chapter covers:

- Arduino models and capabilities—what to look for
- Understanding and reading a schematic
- Some basic prototype construction techniques
- Adding extra parts to an Arduino

About this Book

This book is about using the Arduino development platform for audio applications. It is not intended for absolute beginners but for those who have a little experience with the platform, those who have set up the system, have gone through some of the built-in examples, and are wondering what to do next. You should know the basics of the C language, like what a function is and what structures you can put in them (`for` loops and `if` statements). I am not expecting you to be a programming wizard. I know that different people will be coming to this book with different degrees of knowledge and so this first chapter is a sort of foundation level, catching-up chapter to ensure that we all start from the same level. You might already know the majority of what is here in this first chapter, but if you struggle with large parts of it then it is best to do more foundation work before diving into the rest of the book.

This book also contains projects—some are simple and can be completed in an hour or two; others are much longer, cover a whole chapter, and can take months to complete. When projects get to this level then rather than just trying to replicate what I did, it would be great if you put your own input into it by putting your unique twist on the project. There are suggestions to help you do this at the end of the big projects. So let's start by looking at the Arduino itself.

The Arduino

The Arduino project has a long and prestigious linage. Its roots can be traced back to the influential John Maeda who worked as a supervisor for Casey Rease. I first came across both these people's work in the classic book, *Creative Code,* by Thames and Hudson, 2004, and it didn't take a genius to spot their talent. Casey and Ben Fry worked on a project designed to make coding accessible to a non-technical artistic audience, as by then, computing has moved away from the home coder that was so much part of the 80s micro-computer revolution. The result was a programming language called *Processing*, perhaps one of the worst names for a computer language, despite quite steep competition. Then Casey and Massimo Banzi supervised a student named Hernando Barragan to develop a version of Processing that did the same sort of thing for hardware that Processing did for software; it was called *Wiring*. The last step was in 2005, when Massimo and David Cuartielles took the basic code from these two open source projects and forged it into their own Arduino project, which itself

is open source both in software and hardware. It is a testament to the power of the open source concept, it shows how previous work can be built on and extended without the normal commercial constraints and secrecy. And also without the mega bucks development budget that starting from scratch on a project like this would entail.

The brief was simple, so the system had to be simple and accessible both in terms of hardware and software. The fact that all these years later, you are reading this book is testament to how well they achieved that goal. The continued success and evolution of the project speaks volumes about their dedication and insight. They started making the boards in a small factory close to Turin in northwest Italy, today there are manufacturing facilities in Europe, the Far East and the USA, selling under the brand of Arduino in the UAS and Genuino in the rest of the world. A schism in 2015 with the original manufacturer has resulted in a so far unresolved legal disputes over the Arduino name outside the USA.

Arduino Architecture

The number of models or types of Arduino continue to grow by the year and I assume you have at least one. So to enable you to evaluate the existing models and see where the new models fit into picture, I want to look at the architecture of an Arduino and describe how various models enhance one, or more, particular aspects of it.

The design philosophy has been to bring out a basic design concept and then add variations to it, building up families of Arduinos some with specialist functions. However, the current basic vanilla Arduino is the Uno, whose name was chosen not the least so that it could be pronounced by non-Italians. Earlier basic models went under the name of Duemilanove, Arduino NG (Nuova Generazione), and Diecimila. These are very close variants of the same basic board, but much harder to pronounce, for me at least. Figure 1-1 shows the basic anatomy of an Arduino. It consists of five major blocks:

- The processor
- Communications
- User I/O pins
- Power supply control
- Peripherals

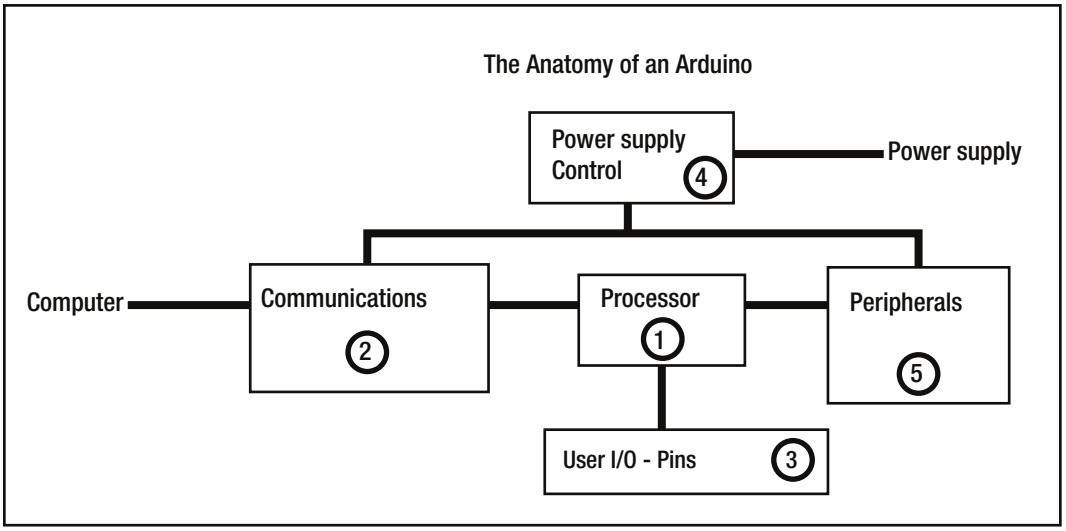


Figure 1-1. *The basic blocks of an Arduino*

Arduino models and variants play with, or alter, one or more of these blocks. So to understand the models, you have to understand, or at least appreciate, these basic blocks. Each block is numbered and discussed in the following sections.

The Processor Block

The heart of the Arduino is the processor, and in many ways it defines the capabilities of the whole board. It is based around processor families and the normal philosophy is to have as much of the processor's resource available to the user as possible. When a manufacturer makes a processor, they normally make a number of variants that have a basic core functionality but have a different amount of the different types of memory. For example, the basic vanilla Arduino always had an ATmega x8 processor; that isn't the part number but the generic base part number where x is substituted for a number representing the variant. The first Arduino used the ATmega 8 chip but soon moved onto the ATmega 168 and then onto the ATmega 328 used today. The main difference between these is the amount of memory they had for storing the program, known as Flash memory, and the amount of memory for storing data, known as SRAM (Static Random Access Memory). A third, less important, sort of memory is the EEPROM (Electrically Erasable Programmable Read Only Memory), and it provides a small provision for non-volatile storage of data. Non-volatile means that it persists even after the power has been removed.

The processor is normally in a DIL (Dual In Line) package, which means it is a plug and socket arrangement. This is great for user replacement when the inevitable accidents happen. However, manufacturing chip shortages have forced some variants to use surface mount processors, whereas the later models use processors that are only available in surface mount packages. This is true of the other processors used in Arduino boards—the ATmega 2560 used in the Arduino Mega boards is essentially the same core processor but with more internal peripherals like extra timers and four serial ports along with many more I/O (input/output) pins. The 32uX series of processors support USB, thus allowing them to look to the computer like other USB devices such as keyboards, mice, or MIDI devices. The Leonardo model uses the 32u4 processor with Arduino Micro being essentially the same but with a much smaller from factor. The other processor in the Arduino stable is the SAM3X8E ARM Cortex-M3 and is in the Arduino Due and the Arduino Zero. This is a huge step up in processor technology and is at least two orders of magnitude greater than the other processors used.

There is a new class of Arduino beginning to emerge—one that uses two or more processors. An example of this is the Arduino Yun. The proposed TRE has a traditional Atmel processor but also a much more powerful one running Linux, with the two processors communication over a "bridge".

The Communications Block

The communications block shown in Figure 1-1 is the other main differentiator in the Arduino zoo. This is usually a USB/Serial converter of some sort, but it can be other things. The very first Arduino used a simple serial interface but computers do not fit those any more, so some sort of conversion from USB is needed. For many models, this block was provided by a dedicated chip called the FTDI chip. Later this has been replaced with another programmable processor using the ATmega8U2. Some models do not have this block at all and rely on you providing that function separately. This is for boards you are building permanently into projects and therefore don't require communications with a computer in the final configuration.

The User I/O Pins

The user I/O pins are often considered very important especially by beginners. However, any basic Arduino is capable of being expanded to an almost unlimited number of pins. It is worth mentioning here that the use of the word "pin" to mean a single input/output line is unique to the Arduino and is sometimes confused with the physical pin on a chip. Traditionally processor I/O has been grouped into ports consisting of eight

bits on each port. What the Arduino does is abstract those bits into individual pins. Mapping those pins from the port is controlled by the software, which takes into account the model you have chosen. In that way many programs can be run as-is on different models, making the system appear a lot more uniform than they actually are.

■ **Note** The use of the word "pin" to mean a single input/output line is unique to the Arduino and is sometimes confused with the physical pin on a chip.

The Power Supply Control Block

Perhaps the least glamorous is the power supply control. On the first Arduinos this was simply a movable link that controlled whether the power came from the USB socket or an external input power jack. People were always forgetting this and thinking their entire system had broken. On all modern Arduinos, there is a circuit that automatically detects if there is power being supplied through the external power jack and, if so, switches to it. Designs have changed slightly, but the function remains the same.

The Onboard Peripherals Block

There are two sorts of onboard peripherals—those inside the processor and those mounted on the Arduino board itself. At this stage, you might not understand all of them and their importance for audio work. When you need to use them in this book, I will fully explain them.

Peripherals Inside the Processor

The processor-based peripherals are things like the number of internal hardware counter/timers and number of serial ports, the pin interrupt capabilities (more about these in Chapter 3), and the A/D (Analogue to Digital) converter. They are best assessed by reading the data sheet of the processor involved and often result in changes to the pin functions.

For example, the hardware counter/timers are registers that work autonomously; that is, they do not need any program intervention. They can be set up to produce a fixed frequency PWM (Pulse Width Modulation) signal and each counter/timer can control two PWM outputs. The basic ATmega 328 processor has three counter/timers so any board using this processor can produce six PWM signals. On the other hand, the Arduino Mega has an ATmega 2560 processor with many more timers with different modes and offers 15 PWM signals.

Peripherals Mounted on the Board

The peripherals mounted on the board are easier to understand and most of the time they define the Arduino model. For example, the Arduino Ethernet has an Ethernet controller built-in, and the Arduino WiFi has a WiFi module built-in. No prizes for guessing what an Arduino Blue Tooth has built-in. These are built into the board as opposed to the other option of an add-on board, or "shield," as it is called. These are plug-in boards that add extra chunks of functionality and can be added to many different basic Arduino boards. They do not belong in this section and will be discussed later in this chapter. Other boards have more mundane peripherals built-in. For example, the Arduino Esplora, shown in Figure 1-2, has a microphone, slider, joystick, push buttons, speaker, light sensor, and LEDs all built on the board. These are arranged to look a bit like a games controller and although the microphone input would appear to be useful for sound, it is not because it is configured as an envelope detector and so only records the peak amplitude of sound.

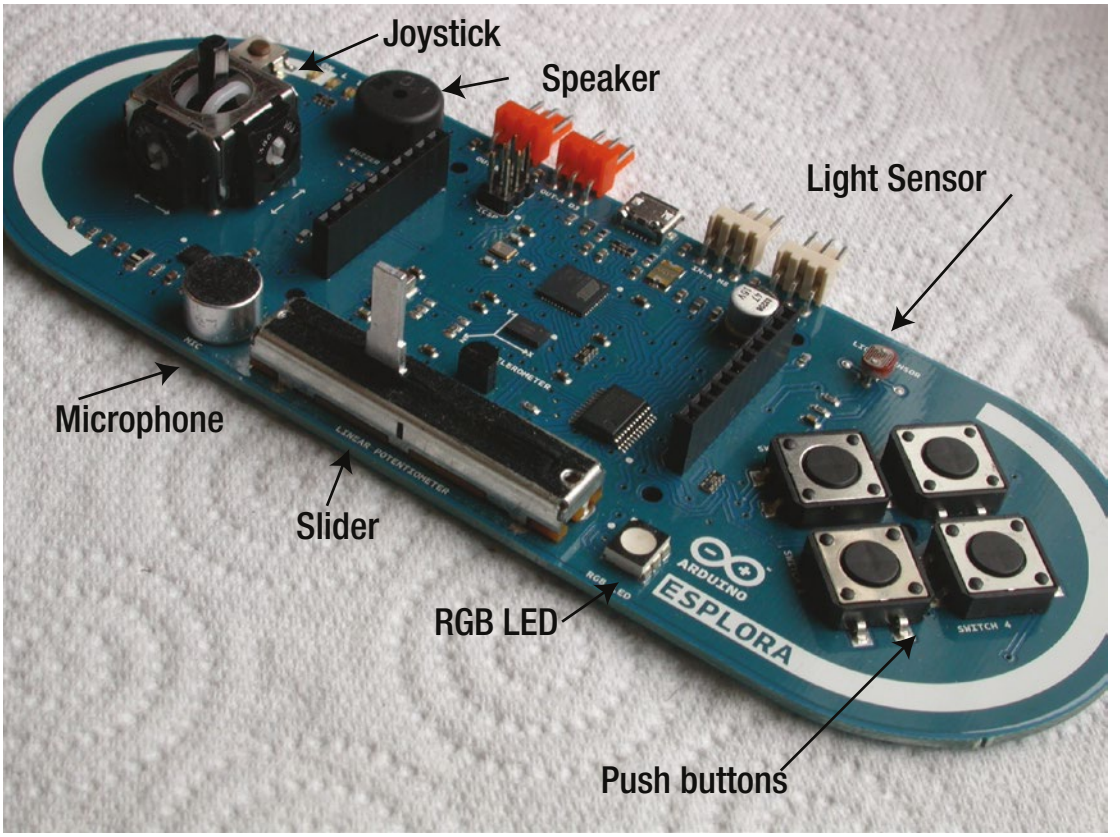


Figure 1-2. *Arduino Esplora*

Arduino Ripoffs, Clones, and Compatibles

The phenomenal success of the Arduino has spawned many copies. Being an open source project, all schematics and PCB (Printed Circuit Board) layout files are freely available—so what is the problem? Well, some people try to deceive you into thinking you are buying the "real thing," whereas others are quite up-front as to what you are buying. It is important to differentiate between the ripoffs, the clones, and the compatibles.

Ripoffs

Although it is perfectly acceptable to make and sell your own version of an Arduino, making it look exactly like the official ones, down to the names and logos, is illegal, and is known as "passing off". This allows the customs officials of your country to confiscate such a board, in theory. This does not happen very often but there are a few cases of this actually happening. Often this form of ripoffs use inferior components like low-quality PCBs, LEDs, and connectors, but the processors are all from the same source and so are likely to work just as well. However, what is more important than the quality of any ripoff model is the fact that the perpetrators of these boards are not adding anything to the project as a whole, but are just acting as parasites on the system. Moneys raised by the Arduino company go into educational projects, research for

the next generation of models, and funding of the Arduino Forum, an incredible collection of some very knowledgeable individuals with a beginner-friendly ethos. Also, working conditions and wages in the places where most boards are made do not cause controversy like some Far East factories might.

Clones

Clones are another matter. They are clearly marked and labeled so that you know they are not official boards. They often bring other things to the party as well. For example, some have features not found on other Arduinos like power over Internet, built-in RGB LED matrix and many powering options. Others might have a physically different layout, or be sold as an unpopulated PCB. There are some with ruggedized protected input/output pins and some with different sorts of connectors to access the input/output pins. There are many clones. Since their only value lies in the lower cost to the end user and this is to the determent of income to the whole Arduino project, I do not encourage you to buy them.

Arduino Certified Boards

The Arduino Galileo is the first of the Arduino Certified boards. It is made by Intel with the cooperation of the Arduino project, so it can be said to be an official Arduino. Basically, this is a Pentium class processor with an Arduino style interface. This extends to both the physical hardware connections and the way you program it using an Arduino IDE look-alike. This is a single processor and will run a version of Linux as well, with all the joys and sorrows that entails.

Compatibles

Some boardsX use the “compatible” term to try to boost their own profiles. These normally consist of having the I/O on the same type of connectors in the same order as an Arduino. Some of these boards are produced by major semiconductor manufacturers. It is a tribute to the power of the Arduino Brand that they have sought to leverage their products in this way.

Some compatibles push the boundaries of what is possible in an Arduino environment. One example of this is the Teensey from PRJC, the latest model 3.2 being based on the MK20DX256VLH7 32-bit ARM Cortex-M4,. The physical footprint certainly lives up to its name, being 1.5” by 0.8” with earlier models using different processors being even smaller. However, this compatible is important in the context of this book because it offers the easiest way of implementing a system that looks like a USB MIDI device to a computer connecting to it. It does this in an easy-to-use way and it integrates nicely into the Arduino IDE (Integrated Development Environment or “the software”) by adding just a few menu options. As you’ll see in Chapter 3, it is possible to do this with some models of official Arduinos, but there are drawbacks.

Roll Your Own

When you make your own project and you want it to be permanent, you don’t have to sacrifice an Arduino board to it. The major functions can easily be replicated and built on strip board. Then you can simply transfer just the processor from the Arduino and place it in the circuit, and then get just another processor to replace the one in your Arduino. The only thing you have to watch is that the replacement processor has built into it some code called a “boot loader”. Many people sell processors like this, and if you get one without a boot loader, you can always program one in yourself if you have a hardware programmer. The great thing about the Arduino project is that you can make a hardware programmer using an Arduino itself, so it can program in the boot loader yourself quite easily. The alternative, and it is one that I favor, is that you can have a programing header built in to your project and use an Arduino or some USB to serial board to program and continue to develop your project. In effect, you’re turning your project into an Arduino compatible with your own on-board peripherals.

Arduino for Audio

From the point of view of audio, the more powerful the processor the better. However, there is still a lot you can do with the less powerful processors. We will be exploring both ends of the spectrum in this book. The shortcomings in terms of peripherals can, in many cases, be accommodated for by adding additional input/output pins using external chips. Even the lack of memory of the Uno may be compensated for by adding extra memory chips. This memory, however, is not part of the general memory; that is, it can't be used to store variables or arrays. However, it can be used to store data in a byte-by-byte manner, in the same way as the built-in non-volatile EEPROM memory, as you'll see later Chapter 13. Figure 1-3 shows the Arduino Due.

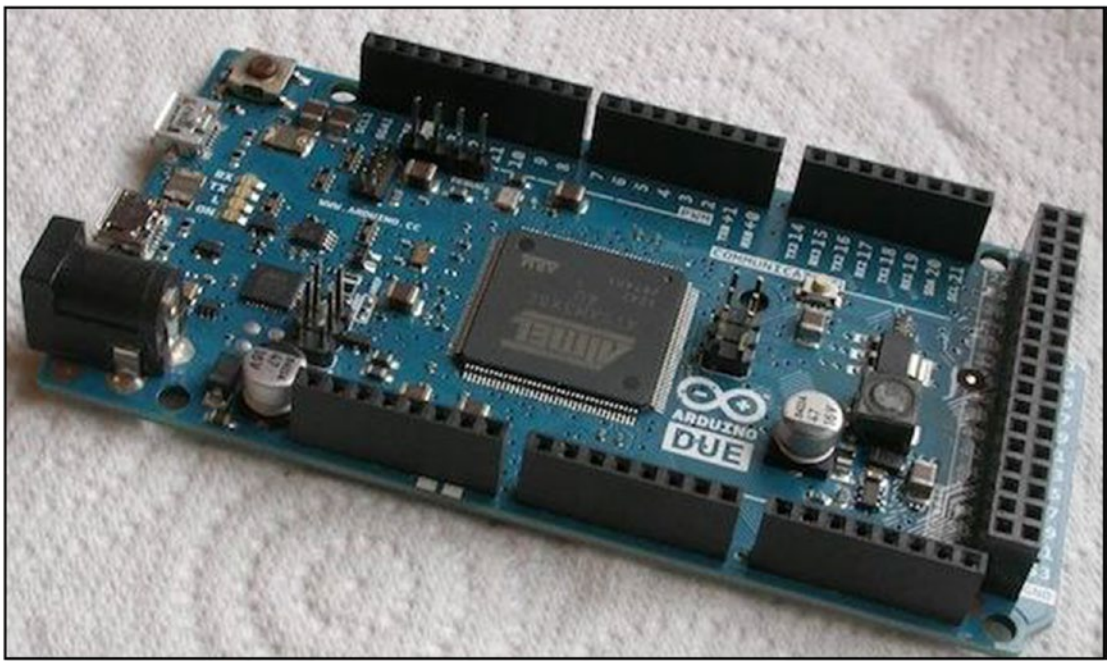


Figure 1-3. *Arduino Due*

The Arduino Due

Currently the best Arduino to use for audio is the Due. At 96K of SRAM, it a decent amount of memory, although when it comes to audio you can hardly have too much memory. The A/D converter is capable of 12-bit resolution (4096 steps), as opposed to the normal 10-bit resolution (1024 steps) of the other Arduino boards. Unlike the other boards it also has two built-in D/A (digital to analogue converters) so that makes it ideal for generating sound using the techniques in the last two parts of this book. It also has lots of I/O, making it suitable for the sound controllers covered in the first part of the book.

The only downside to the Due is that fact that it works on a 3v3 (that is the way professionals refer to 3.3 volts) and the output capacity of the pins with regard to current is much lower than the other Arduinos. The smaller working voltage is a trend in the electronics industry and even lower voltages are being used to allow the cramming of more and more components onto a chip (called component density). You see, the lower the voltage the less heat the switching circuits generate and so the more circuits you can cram into any given space. Other voltages used are 2v3 and 1v8, but we have not seen these in Arduinos yet. So in short, while the Due is the best, it is a bit more delicate and can require extra components when you're interfacing other devices to it. The Arduino Zero is a recently introduced upgrade for this board.

The Arduino Uno

For the beginner, I recommend the Arduino Uno to start with, because it is a lot more forgiving of mistakes and can be expanded in terms of input/output. It lacks true grunt in its processing power, but it is still capable of generating some quite extraordinary sounds and is more than adequate when it comes to being used as a controller. It can even be made so that the computer sees it as a true MIDI device, as you'll see in Chapter 3. However, the downside to this is that when you make it perform like this, it becomes much more awkward to program. Using the Teensy can overcome this when you want a true MIDI device. Using the Teensy is optional, not being an "official" Arduino. As of November 2015 the 1.6.6 version of the IDE allows this feature to be used with the Arduino Leonardo and the Arduino Micro. This is done by using a library called MIDIUSB available from https://github.com/arduino-libraries/MIDIUSB. It makes the Arduino look like a USB HID MIDI device and so behaves just like a MIDI device connected through a MIDI interface. Some of the code examples on the book's web site include examples of its use. Note: when reprogramming such an Arduino, hold down the reset button until the "Downloading" message is seen, and then release it.

Figure 1-4 shows the comparative size of the Uno and the Teensy.

Figure 1-4. *Arduino Uno and the Teensy 3.0 from PJRC*

Schematics

If you want to build anything non-trivial in electronics, you need to learn how to read, and to a lesser important extent, draw a schematic. I used to think there were no rules for drawing schematics, but given some of the abominations you see online now I think there are definite rules. Despite what some beginners might think, the point of a schematic is to make a simple, clear, and uncluttered representation of an

electronic circuit. Many years ago I visited Russia with my family and I bought an electronics construction set for my son. One of those types with a block containing a component with the symbol on the top Although we spoke and read no Russian, we could still use the set fully with the instruction book because the language of schematic diagrams is universal.

Many beginners think you need some sort of special software to draw schematics and indeed there are many packages out there that cater to this. However, a word of caution—most are designed as a front end for a printed circuit layout package and have little flexibility over things like the positioning of pins on an integrated circuit. This results in schematics that are a lot messier and harder to read than they should be. Packages that allow this flexibility are often expensive and very difficult to use. Another thing is that the process of symbol is sometimes tricky and you can spend hours looking for the symbol you need, only to find it does not exist. Some people then use symbols for other components, which makes the schematic absolutely useless. My advice is to ignore these packages and just use a generalized 2D drawing package to draw schematics.

What a Schematic Is and Isn't

A schematic shows which pins, or connections, of which components are connected. What it should not show is how these components and wires are physically arranged. The schematic should show only the topological connections, not any physical ones. This is because this normally makes the interconnections look a lot messier and hard to follow. The translation between the schematic and physical layout is a separate step and not one that needs to necessarily be documented. This is mainly because any such documentation tends to look like a mess and is impossible to follow, so there is little point in creating it.

If you make something from a physical layout and it does not work you really have nowhere to go. If, however, you use a schematic then you can follow the signal flow on the page and pick points on the circuit to check. Of course, the error might be in translation from schematic to physical and if so you will not see the signal you expect and you can check again just that one step in the wiring. You can't do this if all you have is a physical layout. In fact many people, when faced with trying to find a fault using only a physical layout, often resort to dismantling the whole thing and wiring it up again. This is a great waste of time and unfortunately there is a good chance of making the same mistake again.

Of course there are always exceptions to abstracting the physical component to a symbol or plain box. Personally I find having a modicum of physicality applied to components with odd configured connections like some plugs and sockets. MIDI connectors are a case in point; the numbering of the connectors does not follow any logical pattern I can see and this is perhaps the number one reason why beginner's MIDI projects fail. The schematics for the projects in Chapters 3 to 7 will be drawn that way.

Symbols

The key to reading a schematic is to know the symbols used to represent the components; however these symbols are not, as you might expect, universal but normally a mish-mash of several standards. This is made a bit more chaotic by the fact that new components are being invented all the time and official symbol standards take time to catch up. You might think that given this, you would not stand a chance of understanding a schematic, but it is not the case. Most of the time you can work out what a symbol stands for. Also next to the symbol will be a part number, which uniquely identifies the component.

Part Symbols

Typing a part number directly into Google is almost always successful in locating information about it and places to buy it. While the physical appearances of components can vary vastly, the symbol will remain the same. For example, a capacitor can be something the size smaller than a grain of rice and sometimes larger than a can of beer, yet the symbols are the same because they essentially do the same job. Having said that,