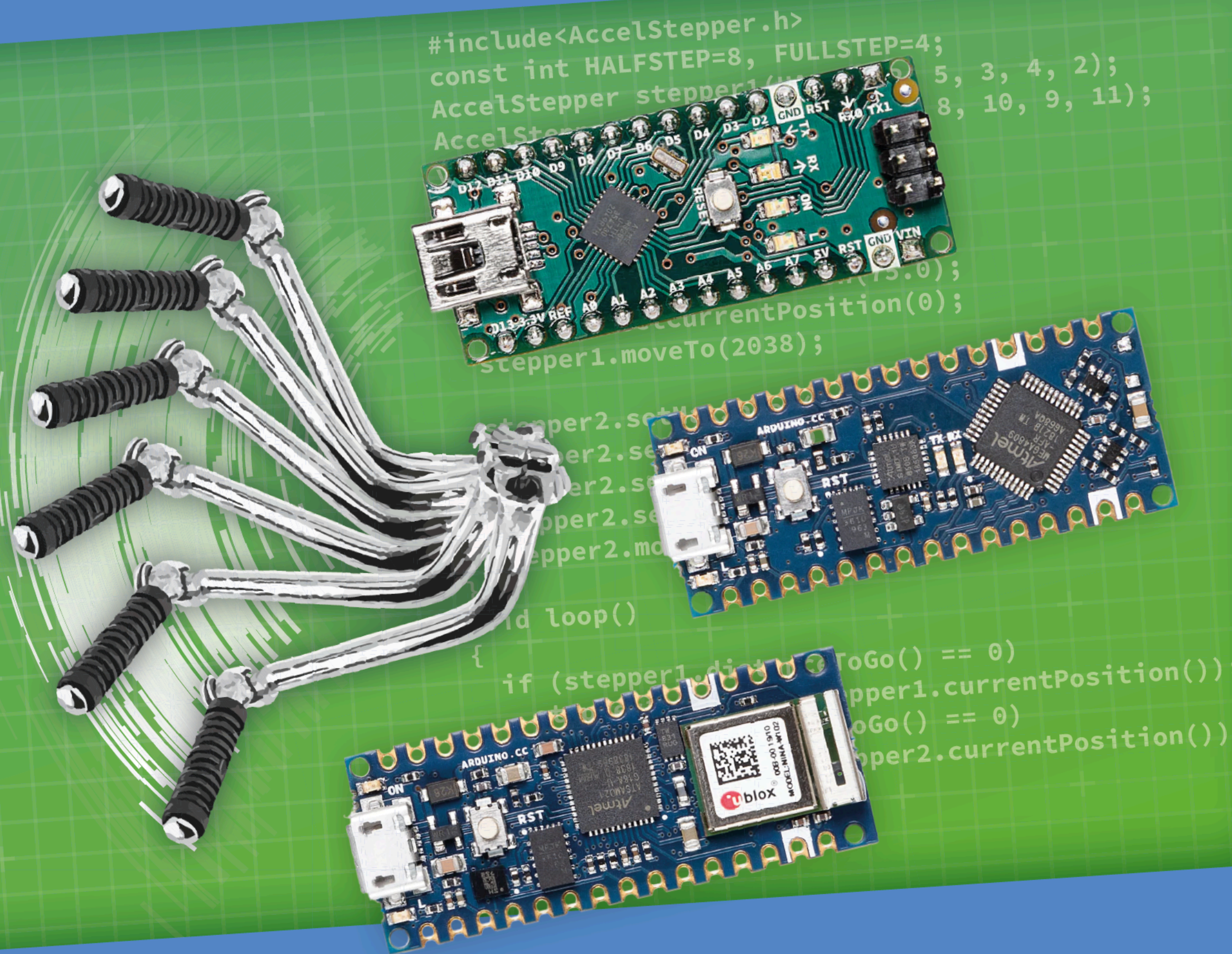


Kickstart to Arduino Nano

Get Cracking with the Arduino Nano V3, Nano Every, and Nano 33 IoT



Ashwin Pajankar

Kickstart to Arduino Nano

Get Cracking with the Arduino Nano V3, Nano Every, and Nano 33 IoT



Ashwin Pajankar

● This is an Elektor Publication. Elektor is the media brand of Elektor International Media B.V.
PO Box 11, NL-6114-ZG Susteren, The Netherlands
Phone: +31 46 4389444

● All rights reserved. No part of this book may be reproduced in any material form, including photocopying, or storing in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication, without the written permission of the copyright holder except in accordance with the provisions of the Copyright Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licencing Agency Ltd., 90 Tottenham Court Road, London, England W1P 9HE. Applications for the copyright holder's permission to reproduce any part of the publication should be addressed to the publishers.

● **Declaration**

The Author and Publisher have used their best efforts in ensuring the correctness of the information contained in this book. They do not assume, and hereby disclaim, any liability to any party for any loss or damage caused by errors or omissions in this book, whether such errors or omissions result from negligence, accident, or any other cause.

All the programs given in the book are Copyright of the Author and Elektor International Media. These programs may only be used for educational purposes. Written permission from the Author or Elektor must be obtained before any of these programs can be used for commercial purposes.

● British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

● **ISBN 978-3-89576-509-4** Print

ISBN 978-3-89576-510-0 eBook

● © Copyright 2022: Elektor International Media B.V.

Editor: Jan Buiting

Prepress Production: D-Vision, Julian van den Berg

Elektor is part of EIM, the world's leading source of essential technical information and electronics products for pro engineers, electronics designers, and the companies seeking to engage them. Each day, our international team develops and delivers high-quality content - via a variety of media channels (including magazines, video, digital media, and social media) in several languages - relating to electronics design and DIY electronics. www.elektormagazine.com

Contents

Acknowledgements	9
Dedication	9
Preface	10
Chapter 1 • Introduction to the Arduino Platform and the Arduino Nano	11
The Microcontroller	11
Popular MCU Families	11
Arduino	12
The Arduino "Classic" Family	12
The Arduino "MKR" Family	12
The Arduino "Nano" Family	13
The Arduino "Pro" Family	13
Arduino is for Everyone	13
The Arduino Ecosystem	13
Arduino Software	14
Official Arduino Boards	14
Arduino Clones and Derivatives	14
Arduino Counterfeits	14
The Arduino Nano and Arduino Nano Every	14
Arduino Nano and Arduino Nano Every Pinouts	16
Powering the Nano and Nano Every MCU boards	19
The Arduino IDE	20
Installing the stable version of Arduino IDE	26
Installation of Arduino IDE on Raspberry Pi OS	30
Working with the Boards Manager	32
Working with the Arduino Nano Every	35
Working with the Arduino Nano	39
Summary	41
Chapter 2 • Playing with Electronics	42
Basics of Programming with Arduino IDE	42
Blink in the built-in LED	43

Working with the basic electronic components	45
Breadboards and power supplies.	46
Jumper cables	50
Light Emitting Diodes	51
Resistors	52
Pushbuttons	53
Improving the LED blink sketch with Functions	54
Building your first circuit on a breadboard	56
Circuits using Nano	58
Working with multiple LEDs	59
Adding a pushbutton to the circuit.	63
Working with RGB LEDs	65
Using Arduino Nano boards with expansion shields	67
Summary	70
Chapter 3 • Assorted Buses and the Analog Input	71
Parallel and Serial Data Transfer	71
Arduino Serial.	73
SPI and I ² C	75
Analog Input	75
Plotting multiple variables	80
Summary	81
Chapter 4 • Pulse Width Modulation and Driving Unipolar Stepper Motors with Digital I/O.	82
The Concept of Pulse Width Modulation	82
PWM with Arduino Nano.	84
Working with a Servo Motor	89
Working with the 28BYJ-48 Unipolar Stepper Motor and the ULN2003A Motor Driver.	92
Using a Custom Library for Stepper Motors	98
Working with Multiple Stepper Motors	101
Summary	102
Chapter 5 • Plotting Geometric Art on an External Display	103
The Ilitek 9225 Driver IC and the Display.	103

Programming the Display	104
Summary	143
Chapter 6 • Working with a Buzzer and a Sensor	144
Working with a Buzzer	144
Working with a Joystick	149
Working with DS18B20 Temperature Sensor.	151
Summary	153
Chapter 7 • Working with the Arduino Nano 33 IoT	154
Introduction to the Nano 33 IoT board.	154
Getting Started	157
Working with WiFININA library	159
A Telnet-based Group Chat Server	172
Pinging a Remote Server	174
A simple Web Client.	176
Working with a Real-Time Clock	177
Using the DS18B20 Temperature Sensor Jointly with the RTC.	179
Visualizing the Temperature graph with ThingSpeak	185
Programming the Built-in IMU.	190
Summary	191
Conclusion	191
Index	192

Acknowledgements

I wish to thank my friend Anuradha for encouraging me to author this book. I also want to express my heartfelt gratitude towards Ferdinand TeWalvaart, Jan Buiting, Alina Neacsu, and Shenja Panik from the Elektor team for guiding me through each phase of the publication process. This is my second book with Elektor International; Media and it is a great experience to perform with their publishing team. Finally, I want to show appreciation to everyone directly and indirectly associated with this project.

Dedication

This book is dedicated to the memory of Prof. Govindarajulu Regeti (July 9, 1945 — March 18, 2021)



Popularly known to everyone as RGR, Professor Govindarajulu obtained his B. Tech in Electrical and Electronic Engineering from JNTU Kakinada. He also earned his M. Tech and Ph. D. from IIT Kanpur. Prof. Govindarajulu was an early faculty member of IIIT Hyderabad and played a significant role in making IIIT Hyderabad the top-class institution it is today. He was by far the most loved and cheered for faculty member of the institute. He was full of energy to teach and full of old-fashioned charm. There is no doubt he was concerned with every student as an individual, taking care to know about as well as guide them. He has taught, guided, and mentored many batches of students at IIIT Hyderabad, including the author of this book.

Preface

I have been working with the Arduino since 2016 and I believe it to be an excellent platform not only to learn electronics but also to deploy real-life production systems. Using Arduino boards, I have produced and deployed many real-world applications such as IoT enabled electrical infrastructures at home and workplaces, web-enabled weather monitoring systems, and a robot for welcoming guests in the reception area of my former employer.

I always wished to write a step-by-step and detailed book on one of the most prominent family of microcontroller boards by Arduino, the "Nano". In this book, I have covered three members of this family: Arduino Nano V3, Arduino Nano Every, and Arduino Nano 33 IoT, in happy unison with the Arduino IDE 1.x and Arduino IDE 2 RC5 for the purpose of demonstrating the C and C++ code. I have written the book in a step-by-step and detailed way to explain concepts, build required circuits, and finally write the code. The book covers more than 60 examples to address a variety of major concepts related to the world of electronics and the Arduino Ecosystem in particular.

This book required me to find and digest loads of documentation, attend online tutorials, write code, and post on online forums when I was stuck with some problem. This result accounts for hundreds of manhours of work I spent to finish the project. I have referred many online sources for code and images. I also mentioned those sources with all the licenses whenever and wherever I borrowed and modified the source codes and images.

I finished writing the draft of the book when India (my current resident country) and the world is gradually recovering from the COVID-19 pandemic. This project and the guidance provided by the Elektor team gave me a great sense of purpose and motivation to continue exploring the world of knowledge on Arduino. I hope all the readers will like and enjoy this book as much as I enjoyed writing it. Happy learning and exploring!

Reader Notice. An archive file (.zip) comprising the software examples and Fritzing-style circuit diagrams discussed in the book may be downloaded free of charge from the book's product and resources page on www.elektor.com (search for: book title and author).

Chapter 1 • Introduction to the Arduino Platform and the Arduino Nano

Welcome to the very first chapter. In case you have not read the Table of Contents and the Preface, I strongly recommend that you do so before commencing with this chapter. Let's begin the journey to the amazing world of the Arduino Platform and its Ecosystem. This is an introductory chapter, so you will be learning a lot of concepts needed to build sufficient background to begin with Arduino. This chapter addresses the following topics:

- the microcontroller
- Arduino
- Arduino is for Everyone
- the Arduino Ecosystem
- the Arduino Nano and Arduino Nano Every
- powering Nano and Nano Every microcontroller boards
- the Arduino IDE
- working with the Boards Manager
- working with the Arduino Nano Every
- working with the Arduino Nano

After reading this chapter, you are all set to explore the Arduino Platform and Ecosystem.

The Microcontroller

A microcontroller is usually a tiny, complete computer system on a single integrated circuit (abbreviated as IC). An MCU (MicroController Unit) can have one or more processors, memory, and programmable Input/Output (IO) within a single IC package. In most cases, there are two types of memory included on the chip. The first one is Programmable memory, usually EEPROM (Electrically Erasable Programmable Read-Only Memory), OTP (One Time Programmable) ROM, ferroelectric RAM, or NOR flash. Also, MCUs also have a small amount of RAM (Random Access Memory) where programs are loaded and executed. In short, an MCU is a microprocessor with Program Memory, RAM, and Programmable IO on a single chip. The main difference between microprocessors and MCUs is in their applications. Microprocessors are general-purpose processors used in desktop computing and general-purpose computing. MCUs are usually used in embedded systems and tasked to performing some sort of physical action. For example, MCUs can be employed in printers, vehicle dashboards, or engine controls in cars and aircraft.

Popular MCU Families

Based on their architectures, MCUs can be divided into various families. Here's a list of some of the most common families:

- PIC (Peripheral Interface Controller)
- ARM (Advanced RISC Machines)
- AVR (Alf and Vegard's RISC Processor)
- MSP (Mixed Signal Processor)
- Intel 8051 and derivatives

Arduino

Arduino (<https://www.arduino.cc>) is an open-source MCU platform, meaning it has open-source and free software and open-source hardware. It usually combines an MCU on a single PCB with other features. Arduino is a popular family of MCU based MCU-based boards. Earlier, the Arduino group had also produced a series of Linux-based computers. However, by now they seem to have discontinued them and are solely focusing on MCUs. The Arduino family has a lot of members, and they can be categorized into three subfamilies as follows:

The Arduino "Classic" Family

This family has the most popular, the oldest, and the most successful boards, including (April 2022):

- Arduino UNO Rev3
- Arduino Mega2560 Rev3
- Arduino Leonardo
- Arduino UNO Mini Limited Edition
- Arduino Due
- Arduino Micro
- Arduino Zero
- Arduino UNO Wi-Fi Rev2

You can find them at the URL:

<https://www.arduino.cc/en/hardware#classic-family>

The Arduino "MKR" Family

MKR is a family of Arduino MCU boards that combine an MCU with advanced communications such as Wi-Fi, LoRa, Bluetooth, and Sigfox. They are based on the MCUs that are members of Microchip's SAM D21 Family. These are the members of MKR family:

- Arduino Nano 33 IoT
- Arduino Nano RP2040 Connect
- Arduino Nano BLE Sense
- Arduino Nano 33 BLE
- Arduino Nano Every
- Arduino Nano

You can read more about the MKR family at

<https://www.arduino.cc/en/hardware#mkr-family>

You can also read more about the SAM D21 family at

https://content.arduino.cc/assets/mkr-microchip_samd21_family_full_datasheet-ds40001882d.pdf

The Arduino "Nano" Family

The "Nano" family has a very small physical footprint, and its members range from the basic Arduino Nano Every to advanced boards such as Nano RP2040 Connect.

- Arduino Nano 33 IoT
- Arduino Nano RP2040 Connect
- Arduino Nano BLE Sense
- Arduino Nano 33 BLE
- Arduino Nano Every
- Arduino Nano

We will discuss the **Arduino Nano** and **Arduino Nano Every** boards in detail throughout this book.

You can read more about the Nano family at the URL:

<https://www.arduino.cc/en/hardware#mkr-family>

The Arduino "Pro" Family

This family provides a complete IoT platform and industry-ready solutions. The members include (but are not limited to):

- Portenta H7
- Portenta X8
- Portenta Machine Control

You can read more about the Pro family at the URL

<https://www.arduino.cc/pro/hardware>

Arduino is for Everyone

Anyone can use an Arduino thanks to the following features:

- inexpensive
- cross-platform
- simple, clear programming environment
- open-source and extendable software
- open-source and extendable hardware

You will explore these features in detail in this book in the upcoming chapters.

The Arduino Ecosystem

As I have mentioned in the earlier section, "Arduino" is actually a family of open-source and free software and open-source hardware. Consequently, both software and hardware are open-source and can be extended. There is a mature community of makers and professionals using these tools, as well as many open-source libraries available for various third-party

hardware components that can be interfaced with Arduino. Even you can develop your own library once you are comfortable with the coding of the Arduino platform. Let's dive deeper into the ecosystem of Arduino and explore all the components.

Arduino Software

The Arduino organization has published a free and open-source software and IDE known as the **Arduino IDE**, a desktop application for Linux, Windows, and macOS. It comes with many libraries and example programs for learning the programming "on" the Arduino platform. It supports C and C++ syntax tailor-made for the Arduino platform. All the members of Arduino family can be programmed with this IDE. It is freely available for download and use. Also, it is open-source. Many people have created their own libraries and made them available for everyone.

Official Arduino Boards

All the boards produced, marketed, and sold under the brand **Arduino** are official boards. They have all the Arduino markings. You can procure them from the Arduino eStore located at <https://store.arduino.cc>. You can find the list of the appointed global distributors at <https://store-usa.arduino.cc/pages/distributors>.

Arduino Clones and Derivatives

Since the hardware is open-source, all the schematics for creating the PCBs (Printed Circuit Boards) are available online for free at the Arduino pages for the respective board. Consequently, you can create your own boards and sell them. The boards that emulate the functionality of the original and official Arduino boards are known as **Clones**. They are not marketed under the Arduino brand, their makers promoting them under their own brand(s). Similarly, the Arduino design-based boards that have additional functionality and are promoted under brand names other than Arduino are known as **Derivatives**. The line that distinguishes the clones from derivatives is vague at best.

Arduino Counterfeits

These are based on the official boards and bear the Arduino logo, except they are not manufactured, marketed, or sold by Arduino. These products are known as fakes or counterfeits. You can guesstimate their status from the quality of components and build. Do not buy and encourage counterfeits. They are illegal and unethical products.

The Arduino Nano and Arduino Nano Every

As I have mentioned in an earlier section, **Arduino Nano** is a family of boards within the larger ecosystem. The Nano members are marked by a tiny physical footprint. This book explores the two members of this family, the **Arduino Nano** and the **Arduino Nano Every**, in detail. So, let's get started.

The Arduino Nano is the elementary and oldest member of the Arduino Nano family. The current revision is **Arduino Nano 3.x**. The Arduino Nano Every is a pin-to-pin compatible upgrade for Arduino Nano with a faster processor and more memory. You can use both indiscriminately in your projects. Let's compare the specifications of both boards.

		Arduino Nano	Arduino Nano Every
Microcontroller		ATmega328	ATmega4809
USB connector		Mini-B USB	Micro USB
Pins	Built-in LED pins	13	13
	Digital I/O pins	14	14
	Analog input pins	8	8
	PWM pins	6	5
Serial Communications	UART	RX/TX	RX/TX
	I2C	A4 (SDA), A5 (SCL)	A4 (SDA), A5 (SCL)
	SPI	D11 (COPI), D12 (CIPO), D13 (SCK). Use any GPIO for Chip Select (CS).	D11 (COPI), D12 (CIPO), D13 (SCK). Use any GPIO for Chip Select (CS).
Power	I/O Voltage	5 V	5 V
	Input Voltage	7 V to 12 V	7 V to 18 V
	DC Current per I/O Pin	20 mA	15 mA
Clock Speed		16 Megahertz	16 Megahertz
Memory	SRAM	2 KB	6 KB
	Flash	32 KB	48 KB
	EEPROM	1 KB	256 bytes
Physical Dimensions	Weight	5 grams	5 grams
	Width	18 mm	18 mm
	Length	45 mm	45 mm

You can check the official pages at arduino.cc for more information on Nano, see

<https://store-usa.arduino.cc/products/arduino-nano>

<https://docs.arduino.cc/hardware/nano>

You can download ATmega328 Datasheet from the following URL:

<http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf>

You can check the official pages at arduino.cc for more information on the Nano Every, here:

<https://store-usa.arduino.cc/products/arduino-nano-every>

<https://docs.arduino.cc/hardware/nano-every>

You can download the ATmega4809 Datasheet from the following URL:

https://content.arduino.cc/assets/Nano-Every_processor-48-pin-Data-Sheet-megaAVR-0-series-DS40002016B.pdf

After checking the above URLs, you must have realized that the Arduino Nano Every is better than the Arduino Nano. It is even cheaper, so, for all your projects I would recommend using an Arduino Nano Every.

Arduino Nano and Arduino Nano Every Pinouts

Let's study the pinouts of Arduino Nano and Arduino Nano Every. The diagrams shown in Figures 1-1, 1-2, and 1-3 are reproduced from

https://content.arduino.cc/assets/Pinout-NANO_latest.pdf

and

https://content.arduino.cc/assets/Pinout-NANOEvery_latest.pdf.

They are shared under creative commons license which can be found at <https://creativecommons.org/licenses/by-sa/4.0/deed.en>. I have modified them for using here in this chapter.

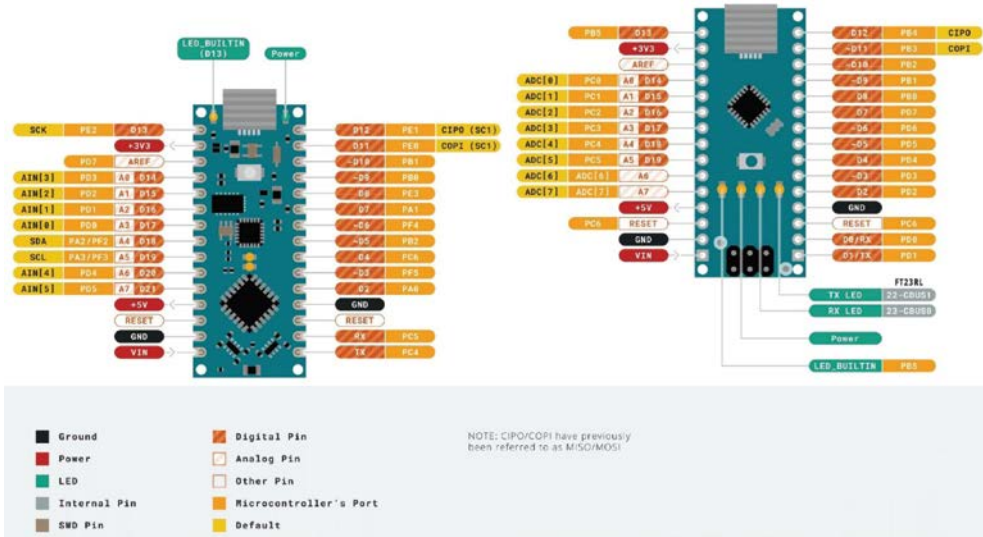


Figure 1-1: Pinouts of Arduino Nano Every (left) and Arduino Nano (right).

In Figure 1-1, at the bottom, the legend explains the meaning of the colors. You need to focus on Digital Pins, Analog Pins, Ground, Power, and LED pins in this diagram.

In Figure 1-2, please focus on the Serial Communications pins allocated to I²C (SDA and SCL), SPI (SS, MOSI/COPI, MISO/CIPO, SCK), and UART (RX and TX),

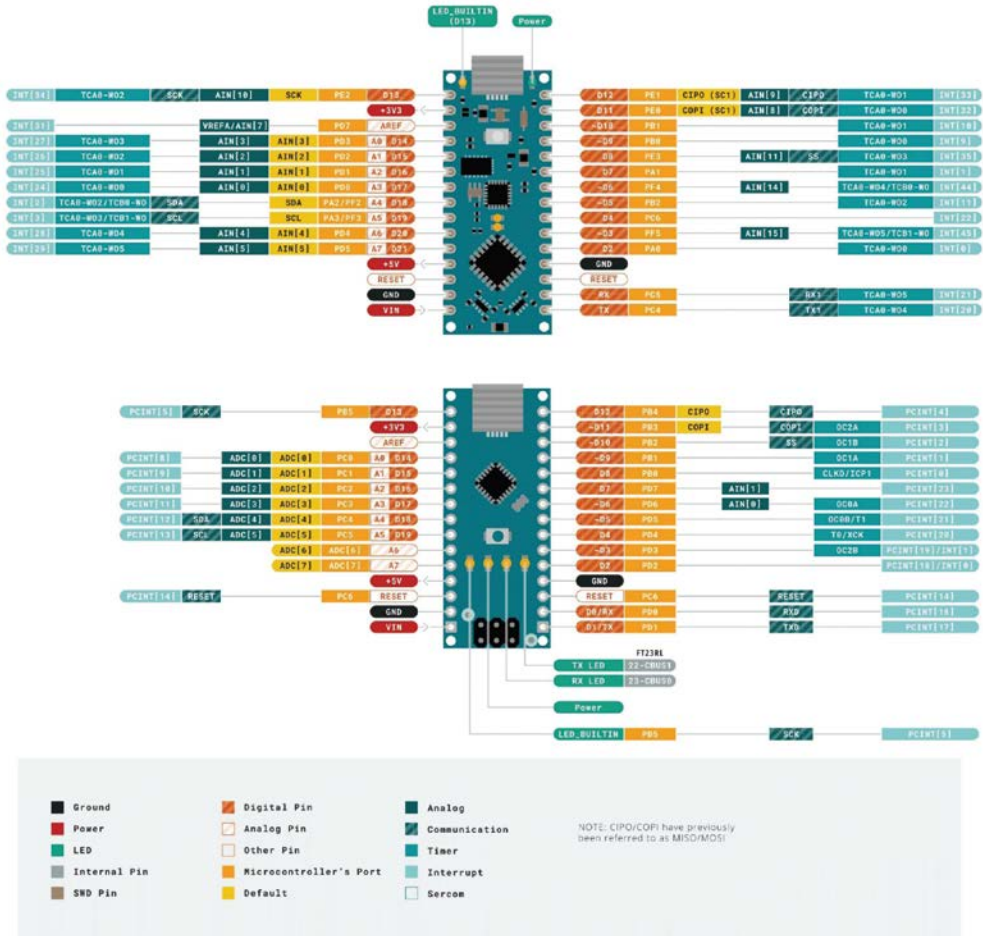


Figure 1-2: Serial communications and interrupt pin description.

In Figure 1-2, the top section shows the Nano Every pin functions, and the middle section, those of the Arduino Nano.

You can study the meaning of additional pins from Figure 1-3.,

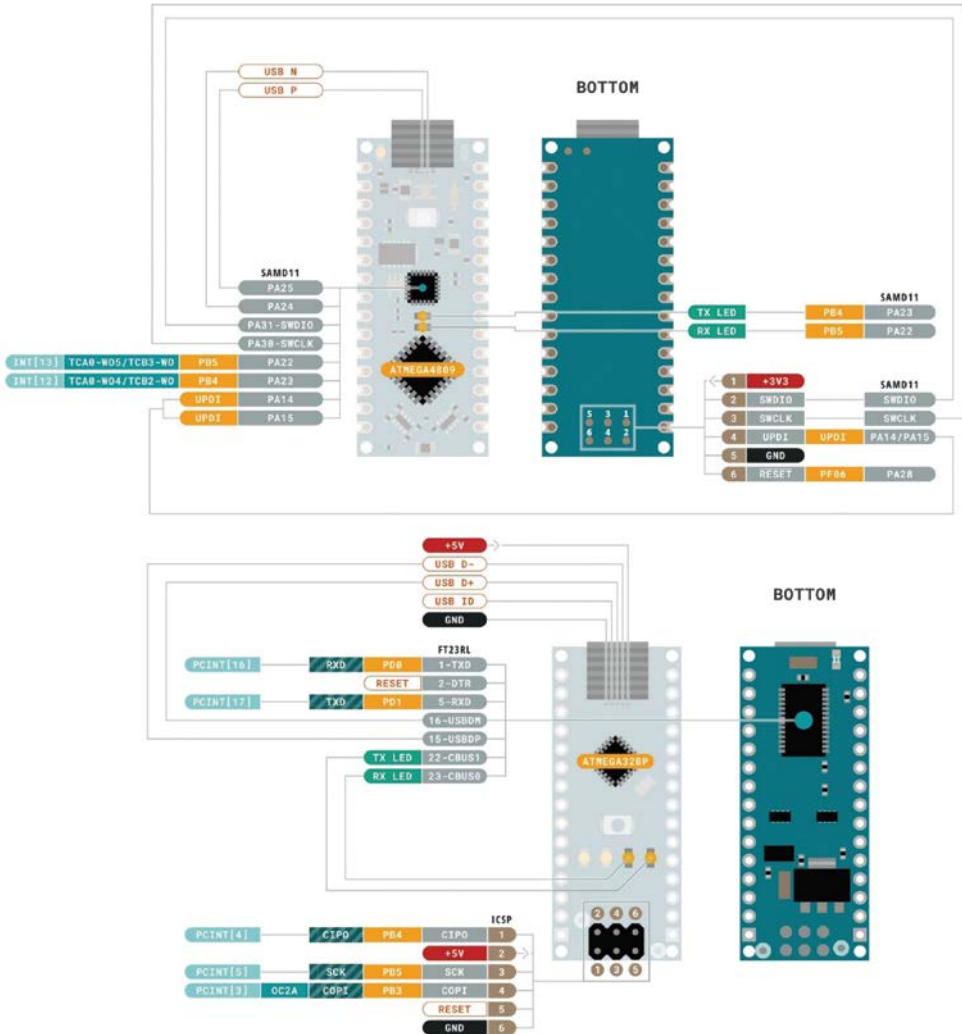


Figure 1-3: Bottom views of the Arduino Nano Every and Arduino Nano boards.

Let's fathom the meanings of the pins shown and described in the diagrams. The pin descriptions of the Nano and Nano Every are almost identical with a few exceptions. Let's see that in detail.

Digital Pins – The pins from D0 through D13 are 14 digital input output pins in all. They operate at 5 volts logic swing Each pin can provide or receive a maximum current of 40 mA. Each pin also has an internal pull-up resistor with a value of 20-50 k-ohms which is disconnected by default.

A few digital pins have additional functions as follows:

UART – These are D0 (**RX**) and D1 (**TX**), used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the pins of an onboard FTDI USB-to-TTL Serial converter chip.

PWM – In the Nano, pins D3, D5, D6, D9, D10, and D11 provide an 8-bit PWM output. In the Nano Every, there are only five pins allocated to PWM: D3, D5, D6, D9, and D10.

SPI – SPI communication is provided by D10 (**SS**), D11 (**MOSI/COPI**), D12 (**MISO/CIPO**), and D13 (**SCK**) on both boards.

On both boards, the on-board LED is connected to D13.

The Nano and Nano Every each have 8 analog inputs (A0 through A7). Each pin provides 10 bits of resolution that enables $10^8 = 1024$ different values for input. The default range for the operational voltage for these pins is 0 V to 5 V. You can change the upper range from 5 V to your desired value by programming.

In the Nano, pins A0 to A5 can also be configured as digital pins D14 through D19.

In the Nano Every, pins A0 to A7 can also be configured as digital pins D14 through D21.

I²C – This functionality is provided by the pins labelled A4/D18 (**SDA**) and A5/D19 (**SCL**). They support I²C using the **Two Wire Interface (TWI)** and use the Wire library.

These are all the important pins on both MCU boards. There are more pins and more meanings to the pins. However, I will not be covering most of those concepts, as they are really out of the scope of this book.

Powering the Nano and Nano Every MCU boards

You can power a Nano with built-in USB with a Mini B type cable and a Nano Every with a USB Micro B type cable as shown in Figure 1-4.

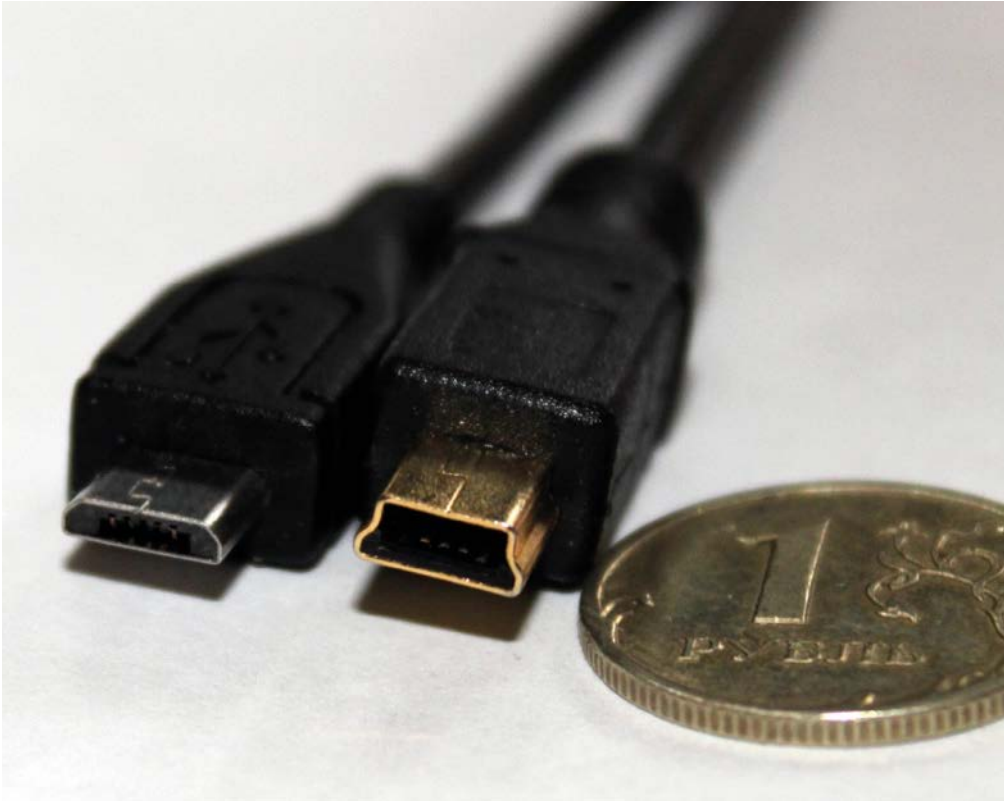


Figure 1-4: USB Micro B and USB Mini B cables. Image provided by [Andrej A. Antonov](https://creativecommons.org/licenses/by-sa/3.0/deed.en) under the license <https://creativecommons.org/licenses/by-sa/3.0/deed.en>

You can power these boards by plugging one end of the USB cable to the MCU board and the other end to a 5-V battery pack, power supply, or the USB port of a computer. You can also power the boards through their VIN pins, but I do not recommend it yet to the beginning readers. Powering through the USB is the best and hassle-free way to power your Arduino.

The Arduino IDE

Arduino IDE is an open-source and free **Integrated Development Environment** for developing the code for Arduino and other MCU boards. You can download the latest version from <https://www.arduino.cc/en/software>. At the time of authoring this book, the latest version is 1.8.19. However, I am going to try the Release Candidate for version 2.0. This is going to be the new major release of the IDE with a lot of improvements. You can download it by clicking the **MSI Installer** option for Windows as shown in Figure 1-5.

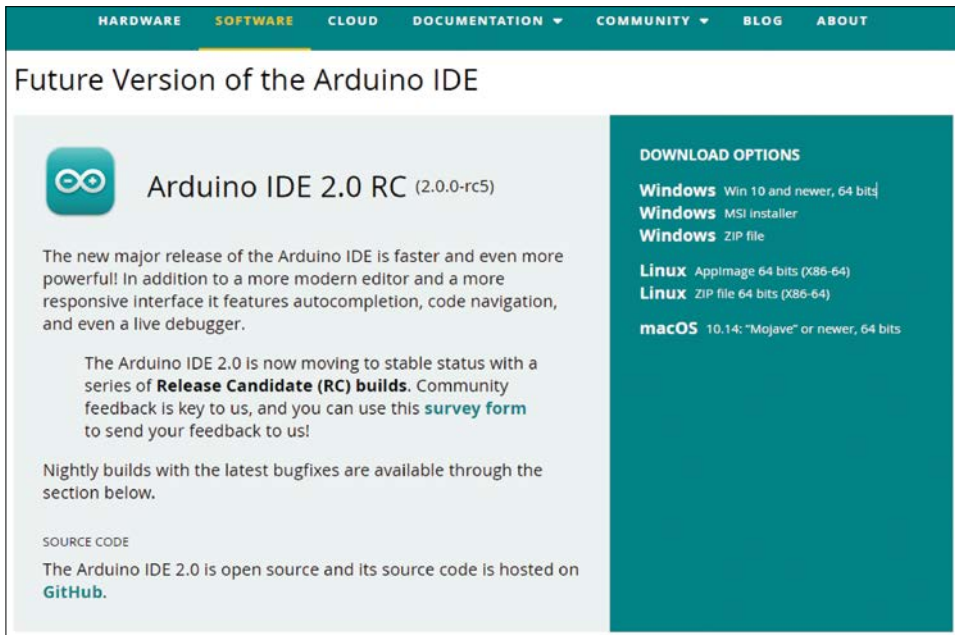


Figure 1-5: Arduino IDE 2.0 Release Candidate (RC).

Options for macOS and Linux distributions are also available. Once you click an appropriate option, the appropriate installation package for your platform will be downloaded. I have a 64-bit Windows computer and consequently downloaded the file **arduino-ide_2.0.0-rc5_Windows_64bit.msi**. **rc5** i.e., **Release Candidate 5**. You can see the file at the bottom bar of your **Chrome browser**, as well as under the **Downloads** option in the browser. You can find the physical file in the **Downloads** directory of your user in the OS. Launch (run) the file and it will show a window like in Figure 1-6.

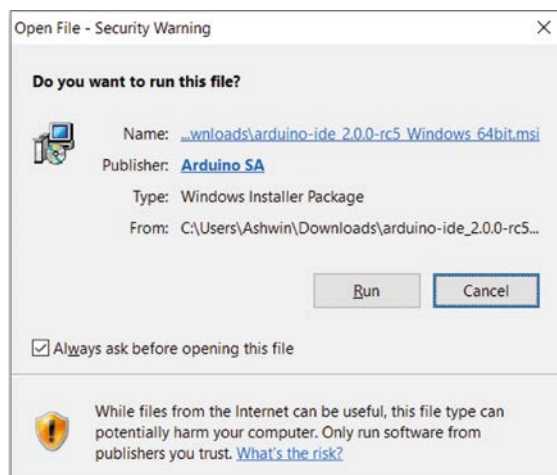


Figure 1-6: Installation window.

Click on the button **Run**. It will configure the IDE and show a progress bar as shown in the Figure 1-7.

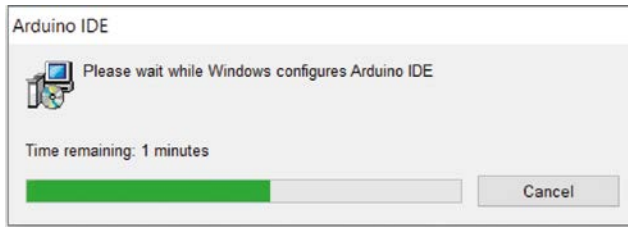


Figure 1-7: Installation progress bar.

Wait for the installation to finish. Once done, this progress bar disappears, and you can check for the IDE using the **Windows Search** feature. Enter the word **Arduino** and it will show the option for it. Click on it. It will show the Arduino IDE splash screen. When you run the IDE for the very first time, the **Windows Firewall** shows you options as in Figure 1-8.

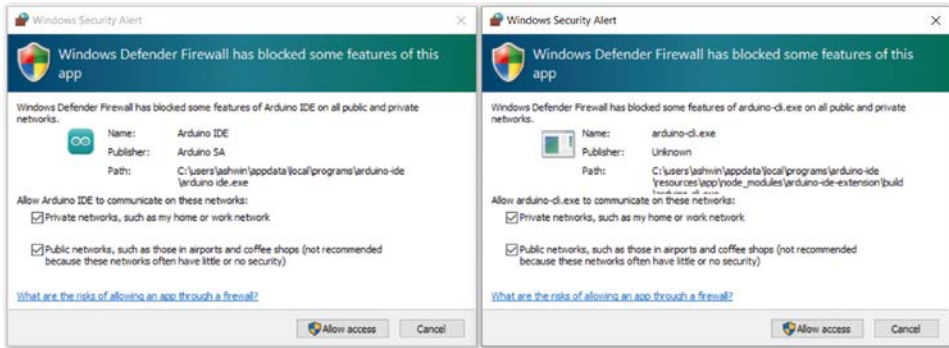


Figure 1-8: Windows security alert.

Check both the boxes in both windows and click on the button **Allow access**. The splash screen for the IDE is as shown in Figure 1-9,



Figure 1-9: Arduino IDE splash screen.

After some time, the splash screen disappears and the IDE is launched. The default window is shown in Figure 1-10.

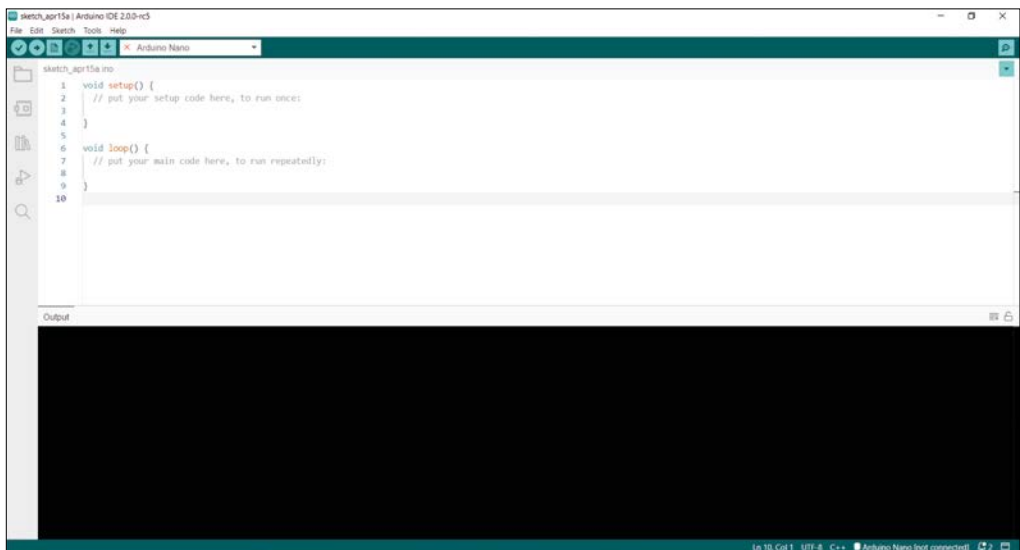


Figure 1-10: Arduino IDE Window.

Before continuing, let's change the preferences. Go to the **File menu** in the menu bar at the top and choose the option Preferences. It opens a window (Figure 1-11).

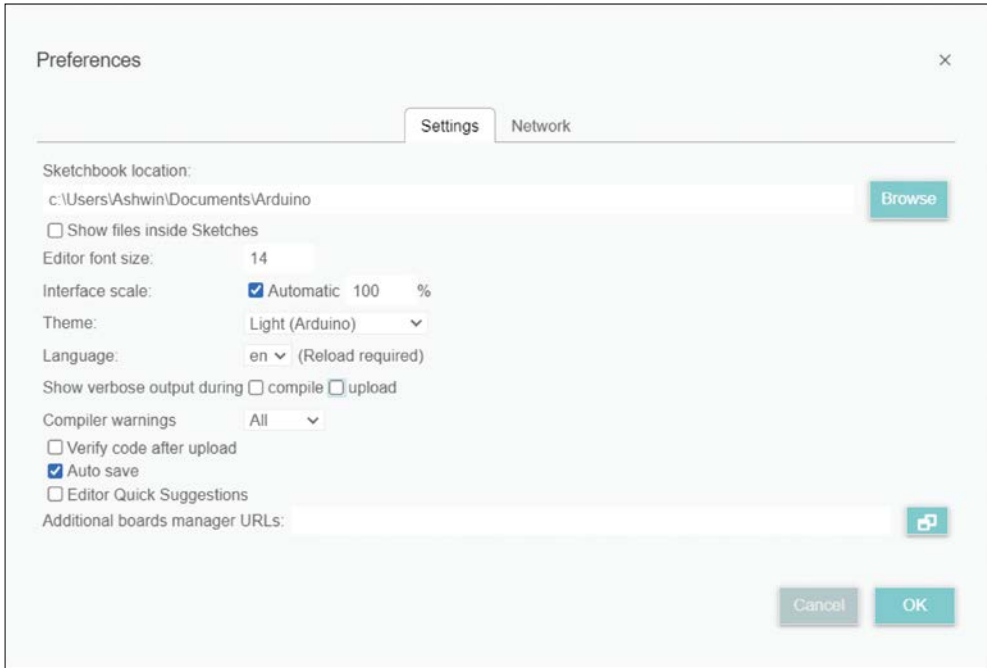


Figure 1-11: Arduino IDE default preferences.

Tick all the checkboxes here and change the path under the textbox labeled **Sketchbook location**. You can use the button **Browse** to choose the location instead of typing it.

The next set of changes are my personal preferences for reducing eye strain. I increased the size of the font used and changed the theme to **Dark** mode. The preferences window looks like the one shown in Figure 1-12 after the changes.

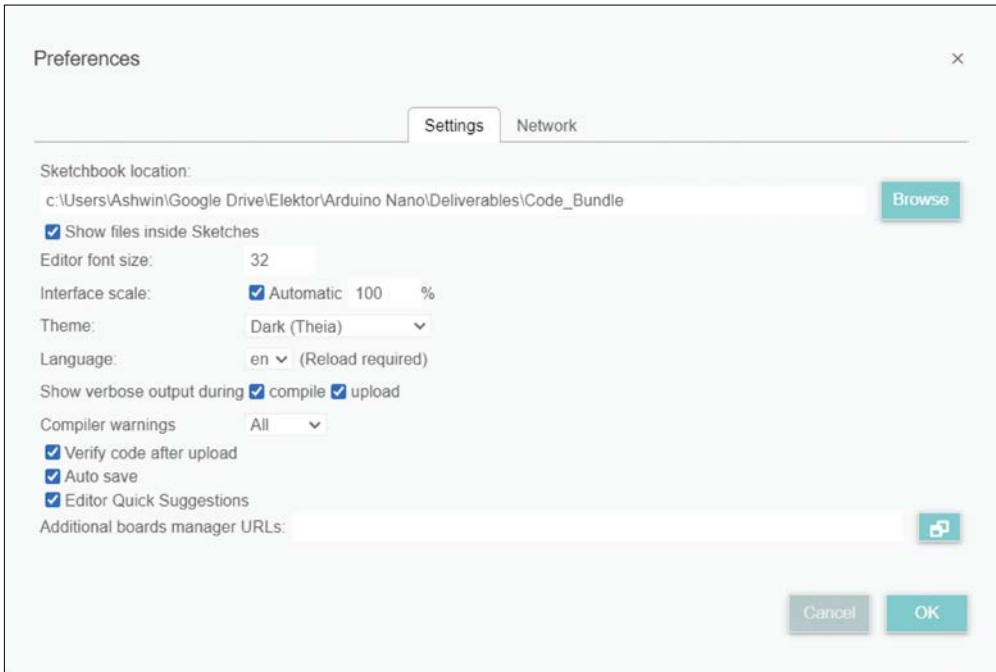


Figure 1-12: Arduino IDE preferences – changed.

This is how you can change preferences for the IDE. After changing the preferences to those shown in Figure 1-12, the IDE window looks as shown in Figure 1-13.

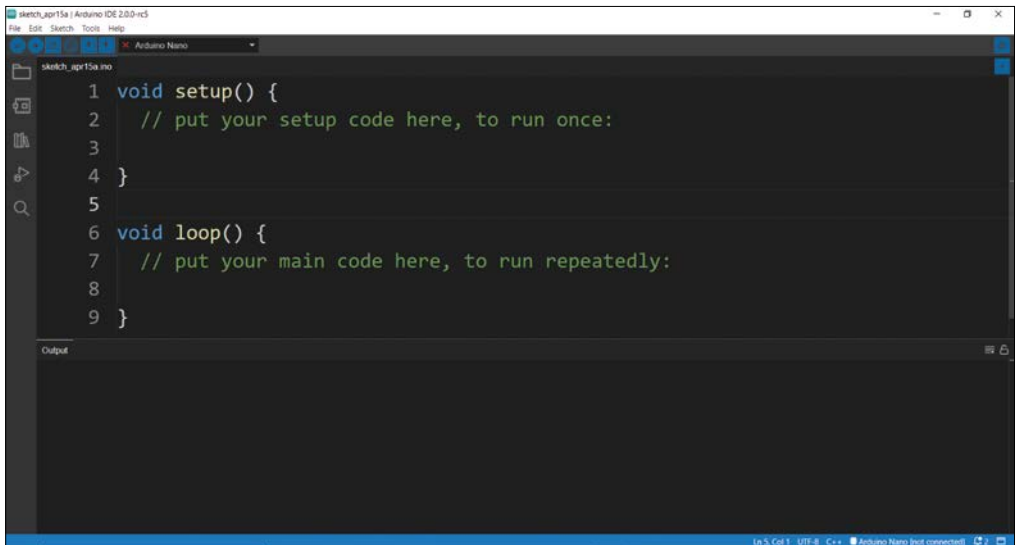


Figure 1-13: Arduino IDE window after changing the preferences.

By default, the file is named with a naming convention which combines current date and a character in alphabetical order. For example, the current file has been assigned a name by default and it is **sketch_apr15a**. If you create a new file using the option under the **File** menu, then the new file is assigned the name **sketch_apr15b** (Figure 1-14).

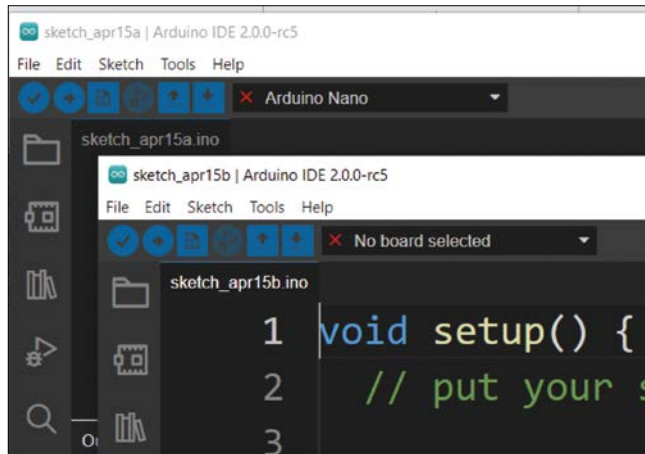


Figure 1-14: Arduino default filenames.

Note that the name assigned to this sketch is just temporary and not yet saved to your disk. You can save it by pressing **CTRL + S** on the keyboard or from the **Save** option under the **File** menu. You can give it another name if you wish. The option **Save** and **Save As**, when clicked, opens the location which you entered in the preferences by default. You can also save at any other valid location by browsing the filesystem.

Installing the stable version of Arduino IDE

You can also install the stable version of the IDE, currently v. 1.8.19. Download it from the software portal (<https://www.arduino.cc/en/software>). Figure 1-15 shows a screenshot of the download options.



Figure 1-15: Arduino IDE Stable version.

Again, be sure to download the correct version based on your computer's architecture. In my case, it is **arduino-1.8.19-windows.exe**. Launch it and it shows a window like in Figure 1-16.

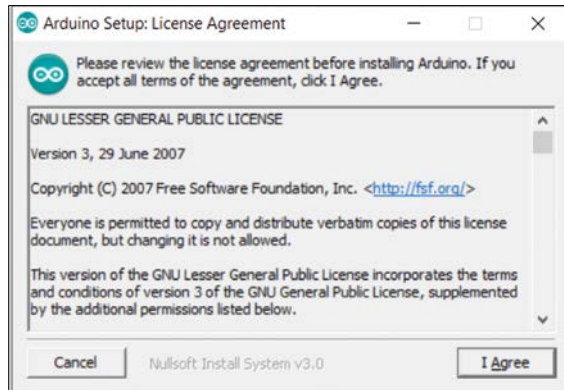


Figure 1-16: Arduino IDE Stable version installation options.

Click on the button **I Agree**. It will show options (Figure 1-17). Check all the checkboxes.

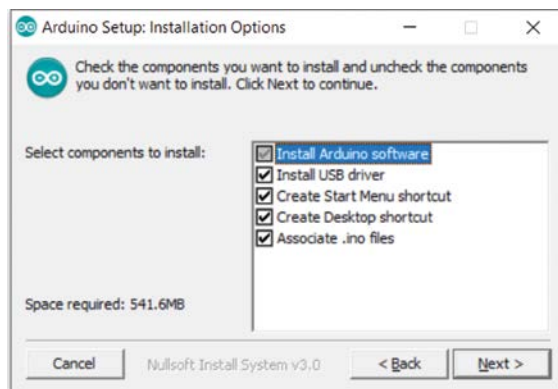


Figure 1-17: Arduino IDE installation options.

Click on the **Next** button. It shows the window where you can select a directory for installation (Figure 1-18).

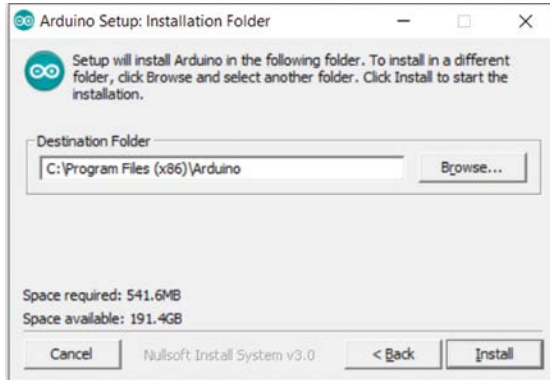


Figure 1-18: Arduino IDE installation directory.

Finally, click on the button **Install** and your PC starts copying the files to the disk. Figure 1-19 is a screenshot of the installation in progress.

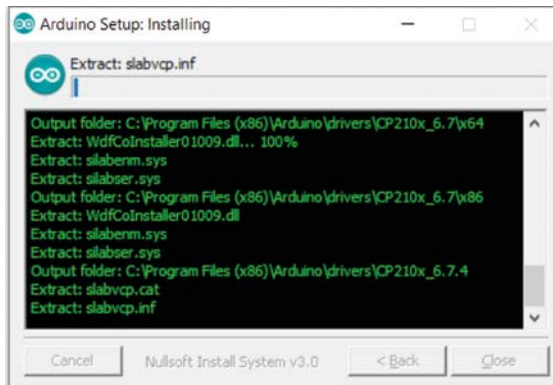


Figure 1-19: Arduino IDE installation underway.

The process creates an icon on the desktop. Also, in the Windows Menu, you can see two options now, Arduino (stable version) and Arduino IDE (corresponding to 2.0 Release candidate), as shown in Figure 1-20.

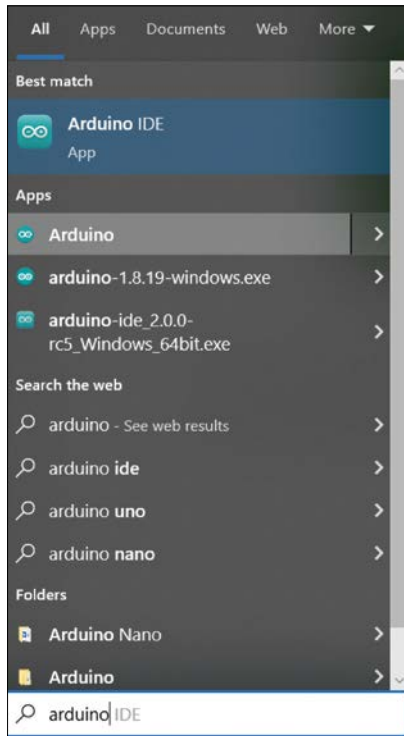


Figure 1-20: Arduino IDEs under Windows Menu.

The current stable version you just installed does not have the "dark" theme that you selected in the Release Candidate version (2.0 RC5). Open the stable version you just installed, and it shows the splash screen Figure 1-21.

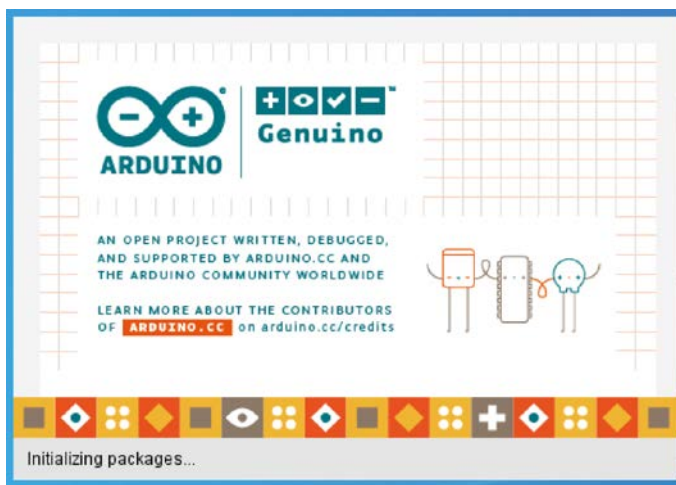


Figure 1-21: Arduino IDE 1.8.19 splash screen.

After that, it shows the following window (Figure 1-22),

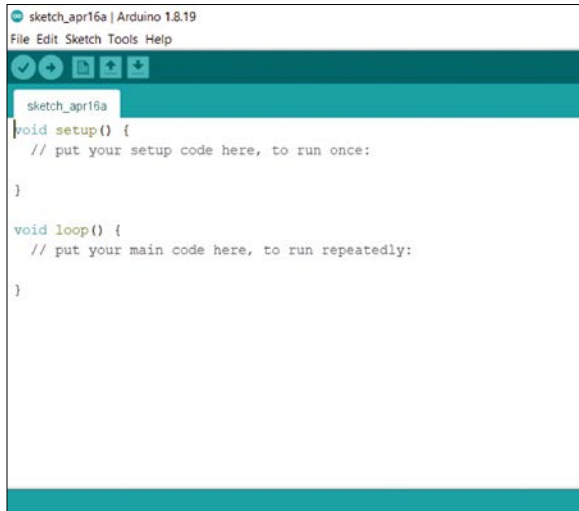


Figure 1-22: Arduino IDE 1.8.19 editor window.

I will be mostly using Arduino IDE 2.0 RC5 to explain all the code demonstrations. However, if you are planning to use IDE version 1.8.19 and if there is anything specific to it, then I will explain it in detail with screenshots. Go ahead and change the preferences to your liking for IDE 1.8.19.

NOTE: After the book is published, both versions of the Arduino IDE are subject to updating after some time. However, the instructions mentioned in the book cover both versions and will also apply to the future updates of both versions unless there is some major change in the user interface (which is very unlikely).

Installation of Arduino IDE on Raspberry Pi OS

You can install the IDE on the Raspberry Pi OS with the following command,

```
sudo apt install arduino -y
```

After installation, you can launch the IDE from the Raspberry Pi OS Menu and under the Programming section as shown in Figure 1-23.

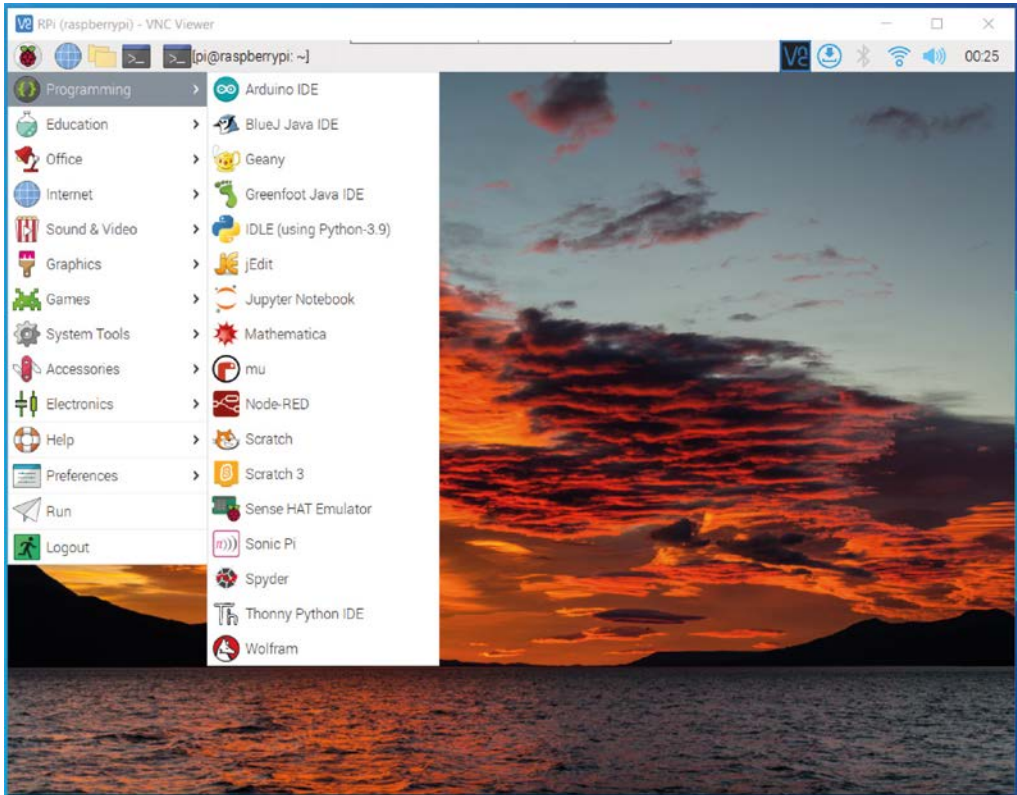


Figure 1-23: Arduino IDE in Raspberry Pi OS.

The entire interface is the same as in version 1.8.19 running on a Windows OS.

Figure 1-24, provided by Cedar101 under Public Domain Creative Commons License at <https://creativecommons.org/publicdomain/zero/1.0/deed.en>), is a screenshot of an instance of Arduino IDE 1.8.5 running on a macOS.



Figure 1-24: Arduino IDE running on a macOS.

Working with the Boards Manager

Arduino IDE comes with a program to add a particular board to the IDE. The Arduino Nano board comes pre-installed and you do not have to install anything for it. However, you have to install the Arduino Nano Every Board. You can open the Boards Manager in IDE 2.0 RC5 from the shortcut vertical menu bar located at the left-hand side of the Code Editor as shown in Figure 1-15.

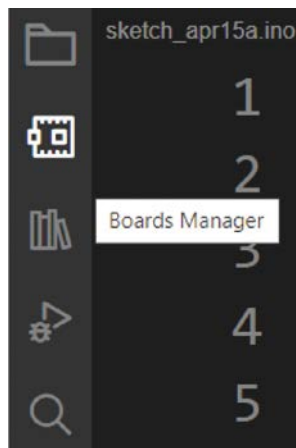


Figure 1-25: Boards Manager in the shortcut menu bar within IDE 2.0 RC5.

Clicking it will open the Boards Manager to the right of this shortcut menu bar as shown in the Figure 1-26.

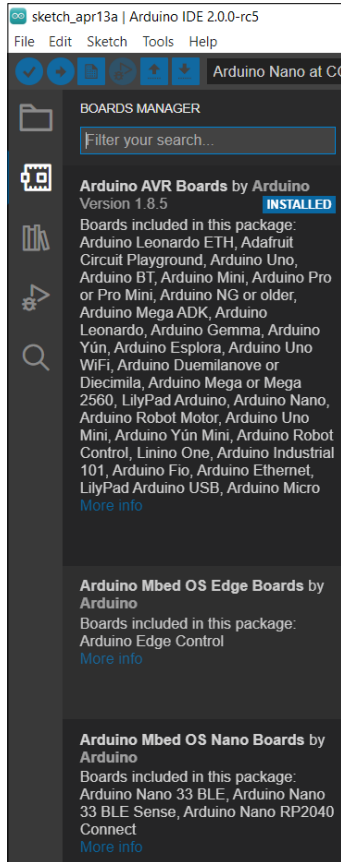


Figure 1-26: Boards Manager alongside the vertical shortcut menu bar within IDE 2.0 RC5.

You can also open it from the main menu bar under the menu **Tools -> Board -> Boards Manager** as shown in Figure 1-27.

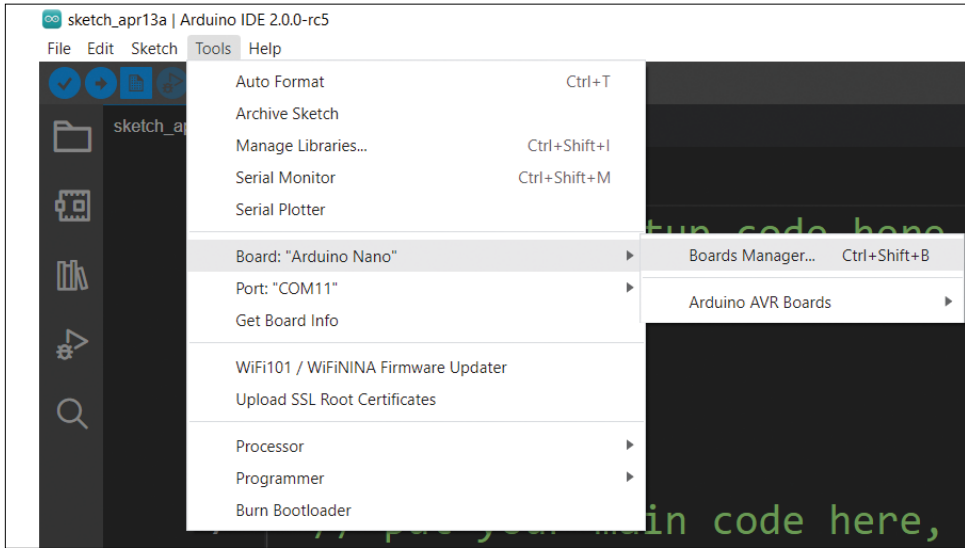


Figure 1-27: Boards Manager option in the menu.

In IDE version 1.8.19, the interface window for the Boards Manager is a bit different as you can see from Figure 1-28.

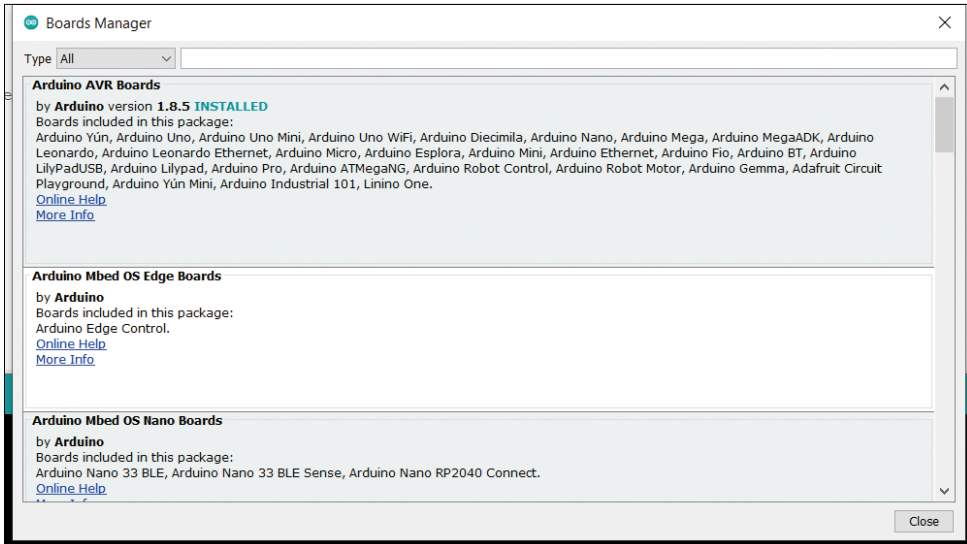


Figure 1-28: Boards Manager running in IDE version 1.8.19.

In both interfaces (Figures 1-26 and 1-28), you can see the option for searching a board. In the search bar, type **megaAVR** and press the Enter key (Figure 1-29).

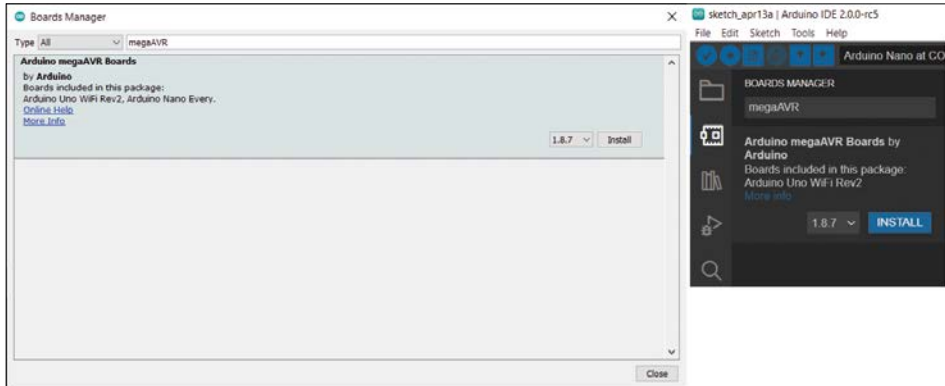


Figure 1-29: Installing the megaAVR on IDE version 1.8.19 and IDE 2.0 RC5.

Doing so will show all the options for installation. Install **Arduino megaAVR Boards**. The system will prompt you to install drivers as shown in the Figure 1-30.

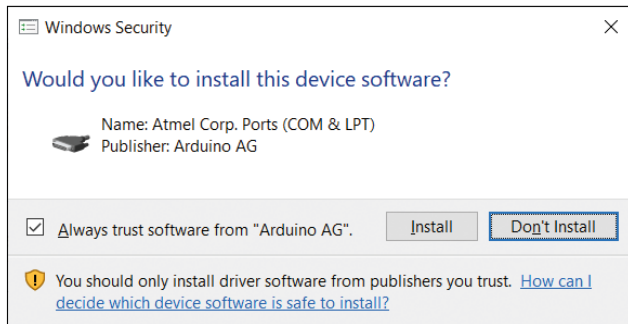


Figure 1-30: Installing drivers.

Working with the Arduino Nano Every

Once installation is done, connect the Arduino Nano Every board to the computer with a cable. In the IDE (both versions), go to **Tools** in the main menu and select the board from the menu as shown in Figure 1-31.

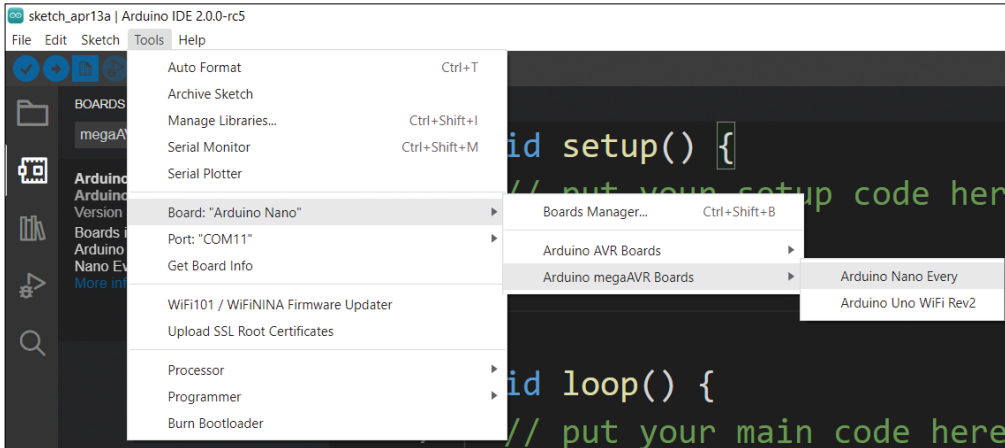


Figure 1-31: Selecting the Arduino Nano Every Board.

When you connect any Arduino-supported board to a computer, it appears as a device on one of the COM ports. Again, from the Tools menu, choose the COM port where the Arduino Nano Every is to be connected as shown in Figure 1-32.

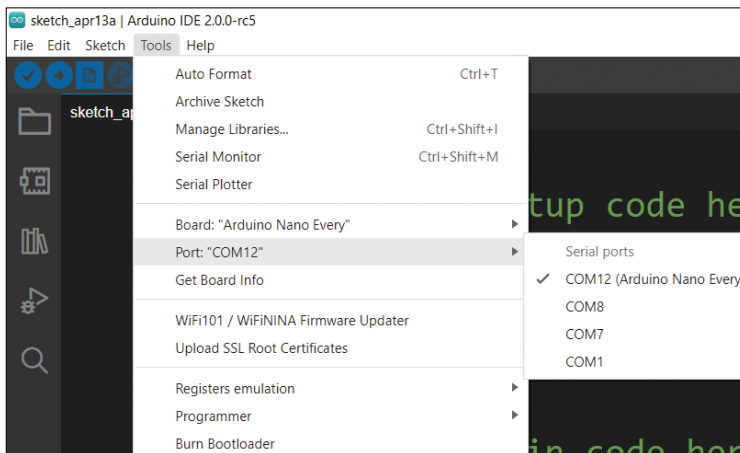


Figure 1-32: Choosing the correct COM port

In case you have difficulty identifying the COM port, check the **Device Manager** application on Windows. Figure 1-33 shows a screenshot just before an Arduino Nano board got attached.

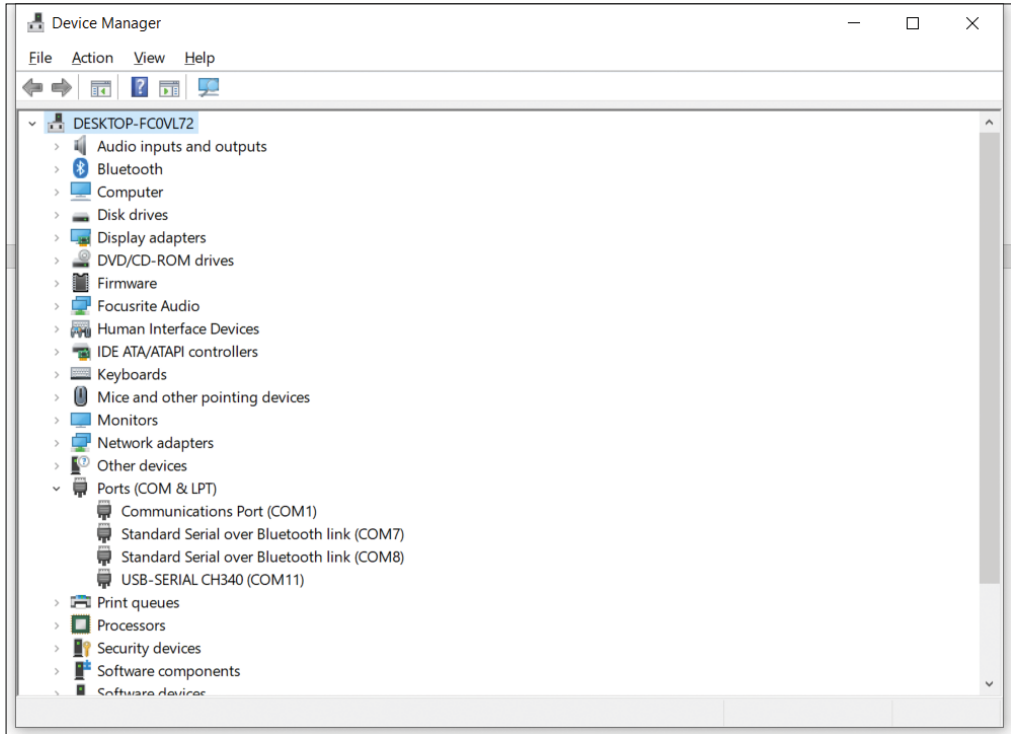


Figure 1-33: Finding the correct COM port.

Within the Raspberry Pi OS, the port appears as in Figure 1-33.,

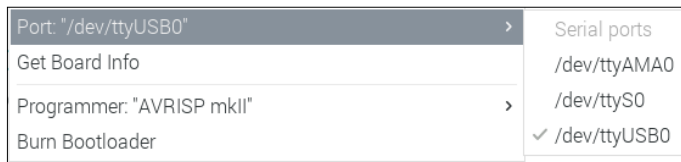


Figure 1-34: Port finding in the Raspberry Pi OS.

In IDE version 1.8.19, you also have to choose an additional option as shown in Figure 1-35.

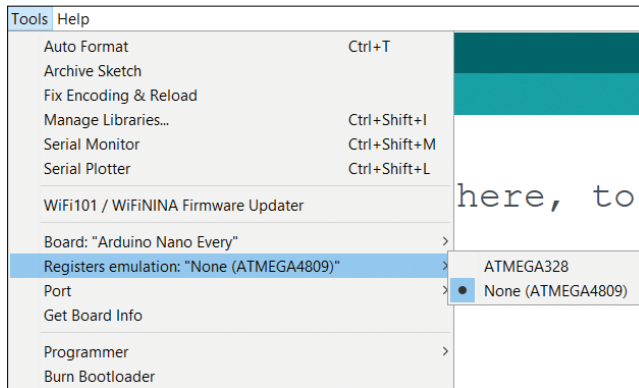


Figure 1-35: Registers emulation selected as "None (ATMEGA4809)".

Now, save the default program (also known as "sketch") to the location of your choice. Name it as you like. The system creates a folder with the given name automatically and saves the program file with the same name as the folder (user-given name). The program file has the **.ino** extension by default.

You can even check the board information by selection that option from the **Tools** menu as shown in the Figure 1-36.

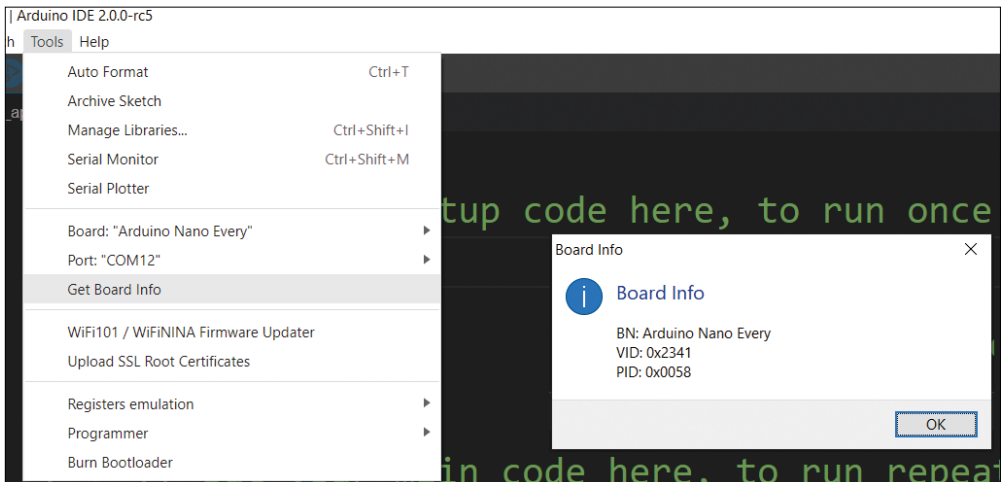


Figure 1-36: Board information.

Now, let's compile/verify and upload the code. Do not worry about the contents of the sketch/program. We will study them in detail in the next chapter. Check the following (Figure 1-36) image to see the shortcuts in both the version of IDEs,



Figure 1-36: Shortcuts in IDEs.

The first row corresponds to the options in IDE 2.0 RC5, and the second row to IDE version 1.8.19. The check mark is the button for **Compile/Verify**. The right pointing arrow is the button for **Upload**. The page symbol is the button for **New Sketch**. The Up arrow is the button for **Open** and the down arrow for **Save**. The **Debug** button is disabled for the Nano and Nano Every so you will not be using it. Click the check mark to compile/verify the sketch. Once verified, it shows the following message in the output window:

Sketch uses 802 bytes (1%) of program storage space. Maximum is 49152 bytes.

Global variables use 22 bytes (0%) of dynamic memory, leaving 6122 bytes for local variables. Maximum is 6144 bytes.

Now, click the **Upload** button. Once the sketch is uploaded, it shows the following message in the end,

```
avrdude: 1 byte of fuse8 written
avrdude: verifying fuse8 memory against 0x00:
avrdude: load data fuse8 data from input file 0x00:
avrdude: input file 0x00 contains 1 byte
avrdude: reading on-chip fuse8 data:
```

```
Reading | #####
##### | 100% 0.00s
```

```
avrdude: verifying ...
avrdude: 1 byte of fuse8 verified
```

```
avrdude done. Thank you.
```

Working with the Arduino Nano

In the previous section, you learned how to verify a program and upload it to Nano Every. In this section, you will learn how to upload a program to the Arduino Nano. Many people use Arduino Nano clones and derivatives that use the "CH340" chip for USB-to-Serial communication. The Original Nano will be detected without hassle. However, if your Nano clone is not detected by your computer, you will have to download and install drivers for the CH340 to your PC, Linux, or Mac. You can do it from <https://sparks.gogo.co.nz/ch340.html>. Keep your Arduino Nano board connected while installing the drivers, run the setup file and then click the button **Install**. See Figure 1-37.

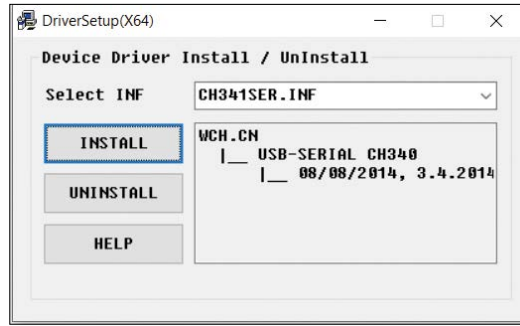


Figure 1-37: CH340 Driver installation.

Once done, it shows a window with the message: Success. If your computer is still not able to detect the Nano board, disconnect and connect it again. At this point, the computer should be able to detect the board without any issue. The process of verification and uploading the sketch/program is identical for the Arduino Nano. However, you have to choose the board within IDE 2.0 RC5 as shown in Figure 1-38.

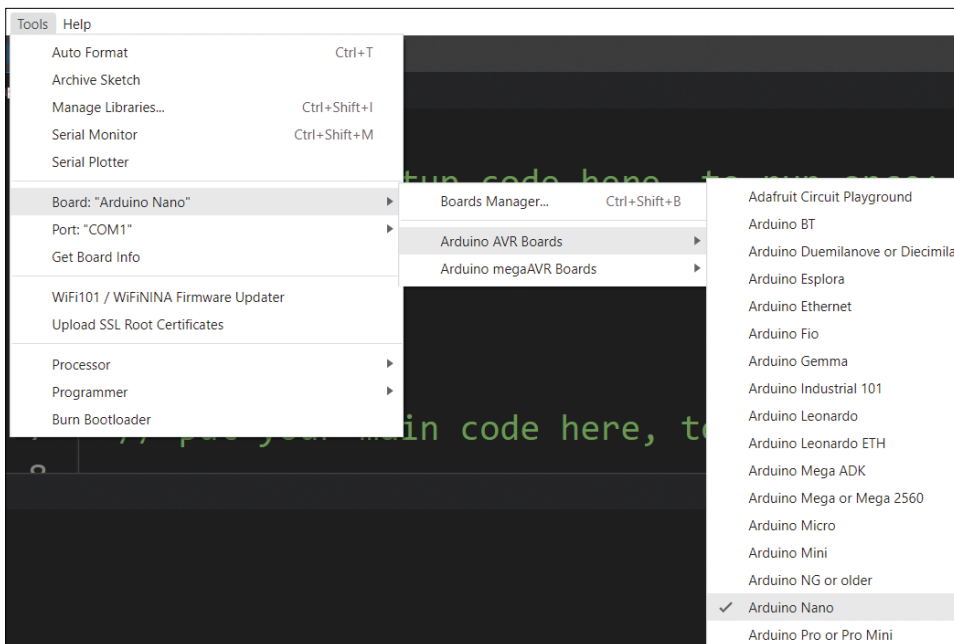


Figure 1-38: Choosing the Arduino Nano board within IDE 2.0 RC5.

The process of selecting the board within IDE version 1.8.19 is exactly the same. However, additional options are offered. You have to select the bootloader for your board as shown in Figure 1-39.

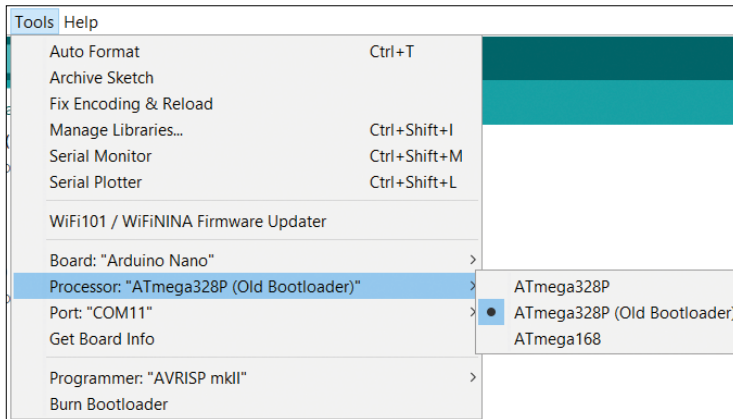


Figure 1-39: Choosing the bootloader within version 1.8.19.

You have to try the bootloaders from the list in order to find the one that matches your board. The rest of the process of compiling and uploading a sketch to a board is the same as you have learned for the Nano Every.

Summary

In this chapter, you got yourself acquainted with the Arduino ecosystem, the Arduino Nano board, and the Arduino Nano Every board. Moreover, you learned the basics of working with stable and upcoming versions of Arduino IDE. You learned how to install a new board. You learned how to verify/compile a sketch and upload it to a board.

As an exercise, explore the various options in the menu within both versions of the IDE. Check for the keyboard shortcuts for **Verify** and **Upload**.

In the next chapter, you will learn to play with simple electronic components and write programs for them. It will be a detailed and hands-on chapter on electronics and programming.

Chapter 2 • Playing with Electronics

In the previous chapter, you learned the basics of the **Arduino Ecosystem**, a couple of boards in the **Nano Family**, and the basics of the current stable as well as next Release Candidate versions of the **Arduino IDE**. In this chapter, you will learn the basics of writing programs/sketches with Arduino IDE and working with electronics. Here's a list of topics you will learn about in this chapter:

- basics of programming with Arduino IDE
- working with the basic electronic components
- improving the "LED blink" sketch with functions
- building your first circuit on a breadboard
- working with RGB LEDs
- using Arduino Nano boards with expansion shields

After absorbing this chapter, you should be comfortable with the basics of Arduino programming and a few electronic components.

Basics of Programming with Arduino IDE

In the previous chapter, you uploaded the default basic program to an Arduino board. In this section, you will learn to understand the meaning of the code. If you recall, you saved the program before uploading. Even if you did not save the program (or sketch, you can use these words indiscriminately, which I will be doing throughout the book), you can always create a new sketch and by default it will have the basic code. The basic code looks like this:

```
prog00.ino
void setup() {
  // put your setup code here, to run once:

}

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

}
```

I have copied and pasted the code as it is, without any modifications. This is the bare minimum code required to compile (or verify, again, these two words can be used interchangeably) the sketch. As you can see, there are two blocks of code, *setup()* and *loop()*. To define your basic program, both blocks are a must for a sketch. Now, try to remove the *setup()* block and compile the sketch, and you will find the following error message in the **Output** section,

```
undefined reference to 'setup'
```


Similarly, if you remove the `loop()` section and try to compile, the compilation will be unsuccessful again with the following message,

```
undefined reference to 'loop'
```

This is the bare minimum of code required for any compilation —it will always be there in any sketch anyone writes for Arduino.

When any sketch is uploaded to a board, it is stored to the board's Flash memory. Even after disconnecting the board from the computer (or any other power source), the code will be retained by the Flash memory, and it won't be erased until you upload a new sketch. Every time the board is powered on, the `setup()` section is executed once. After that, the `loop()` section continues to run as long as the board is powered. If the board is disconnected powered off, it stops running the code (obviously!). When you connect the board to a power source again, the entire process is repeated.

Blink in the built-in LED

All the Arduino boards come with a built-in LED. On the board, it can be identified as "L". Let's write some code to blink it. It is considered the **Hello World!** of the world of electronics programming. Also, I will be using the Arduino Nano Every board running IDE 2.0 R5. However, you can use the Arduino Nano and any version of the IDE for writing the sketch. Let's write a simple sketch (refer: prog00.ino) to turn on the on-board LED.

```
prog00.ino
/*
This Program is written by Ashwin Pajankar
for Elektor on 20-APR-2022
*/

// the setup function runs once
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs forever
void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
}
```

Let's examine it line by line. The code that begins with `/*` and ends with `*/` represents a multi-line comment. Programmers usually use comments to provide additional information about the code. Similarly, the lines that begin with `//` are single-line comments. I have mentioned in the previous chapter that most of the Arduino boards have an on-board LED.

It is connected to digital pin 13 in the Arduino Nano family. The function `pinMode()` initializes a digital pin as an *INPUT* or *OUTPUT*. In this chapter, you will learn their usage. You have to write the pin number and *INPUT* or *OUTPUT* as arguments. The built-in constant `LED_BUILTIN` refers to the on-board LED. So, you do not have to change the code when you upload it to the different boards. `LED_BUILTIN` will always point to the on-board LED. The function `digitalWrite()` sends either *HIGH* or *LOW* to the specified digital pin which means 1 or 0, respectively.

Note: You can check the details of built-in constants at this url:
<https://www.arduino.cc/en/reference/constants>.

Now, select the board and port from the **Tools** menu, then upload the code. You will see a small LED near the micro-USB port lighting up. It is orange-colored which distinguishes it from the power indicator LED which lights up green.

As you have learned, the `loop()` part runs forever, and thus as long as the board receives power, this on-board LED (along with the power indicator LED) will always light unless you upload some other program to the board.

Now, disconnect the Arduino board from the computer and power it with a power bank as shown in Figure 2-1 (Image by Santeri Viinamäki under the <https://creativecommons.org/licenses/by-sa/4.0/deed.en> license).



Figure 2-1: A power bank is a good power source for your Arduino experiments.

Once you power the board again, the LED will light again. This is because the system is running the same program you uploaded earlier. It was uploaded to its flash memory and will not be erased if you turn the power off — it will be rewritten only when you upload new code to it.

Congrats, you just wrote and uploaded your own first custom sketch to the board. Let's create a new file and name it `prog01.ino`. This should be the contents of the file:

```
prog01.ino
int blink_duration = 1000;

void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(blink_duration);
  digitalWrite(LED_BUILTIN, LOW);
  delay(blink_duration);
}
```

From here on, I will use comments in the code as I am explaining the relevant functions. It is considered good practice to add comments to the code (along with separate and detailed documentation) to aid readers and fellow programmers in understanding what your code is supposed to do. You will also benefit from your own comments after a few months as you revisit your projects. You certainly do not wish to stare at an alien piece of code!

The function `delay()` pauses the execution of the sketch for the given amount of time stated in milliseconds. You have also defined an integer variable in the beginning. This variable can be accessed in any block of code in the same sketch, as it is a **global variable**. However, the variables defined in the `loop()` and the `setup()` sections can only be accessed within the respective sections. They are known as **local variables**. Now, upload the code and you will see the LED connected to the digital pin 13 blinking.

Note: You can read more about the function `delay()` at this url:

<https://www.arduino.cc/reference/en/language/functions/time/delay/>

and about the function `digitalWrite()` at this url:

<https://www.arduino.cc/en/Reference.digitalWrite>

Working with the basic electronic components

Let's start working with electronic components such as breadboards, LEDs, resistors, and jumper cables.

Breadboards and power supplies

Figure 2-2 (Giacomo Alessandrini, <https://creativecommons.org/licenses/by-sa/4.0/deed.en> license) is an image of a breadboard.

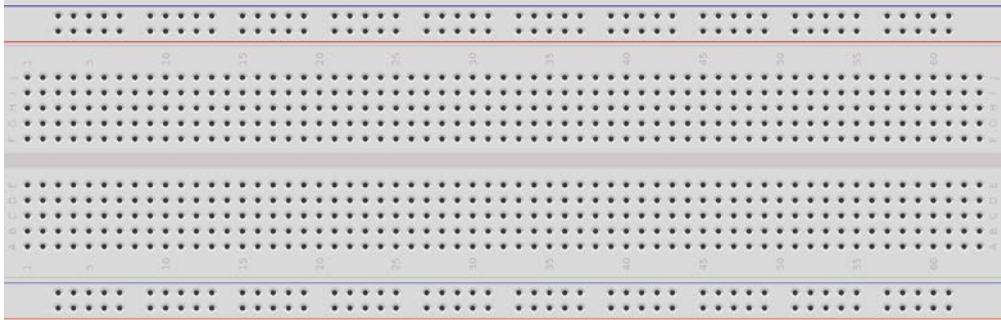


Figure 2-2: A breadboard, also called prototyping board or plugboard.

Shown here is a type **MB 102 830-points breadboard**. All the holes in the breadboard provide electrical contact. You can see four rows of horizontal lines, two marked in blue and two in red. Every "point" in those lines is connected to all the points in the same line. So, if you connect an electrical component to the point on the far left in a line, you get the same electrical signal at the far-right end of the same line. These lines are known as power-supply lines or the power bus lines. They will be used to distribute the supply voltage or "power" across the board. The red line will have the positive supply voltage, and the blue one carries the Ground or 0-V potential.

The middle part is divided into two sections. Each section is comprised of rows and columns. Now, unlike the power bus lines, the points in a single row are not connected. Here, they are connected column-wise. Two sections are separate, and they do not have any electrical contact. In both sections, there are columns comprising of five contact points in each column. If you look carefully, the rows and columns are marked with letters (A, B, C, D, E, F, G, H, I, and J) and numbers (1 through 63), respectively. This means that the points in column 1 are divided into two groups. The first group is A1, B1, C1, D1, and E1. The second group is F1, G1, H1, I1, and J1. All the points in a column group are electrically connected to each other.

Figure 2-3, image by Cz-David under <https://creativecommons.org/licenses/by-sa/3.0/deed.en> license) shows a **Dual in Package (DIP) Integrated Circuit (IC)** mounted on a breadboard.

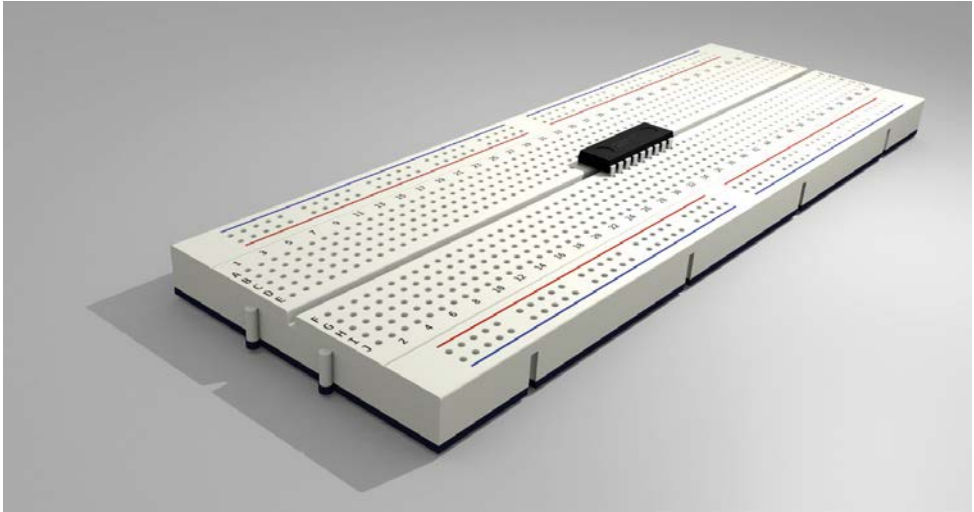


Figure 2-3: A breadboard with a lone DIP integrated circuit plugged in.

This image makes the purpose of the gap between two sections obvious. You will also mount your Arduino Nano boards on the breadboard in similar fashion. You can see how the points are internally connected in a mini breadboard pictured in Figure 2-4.

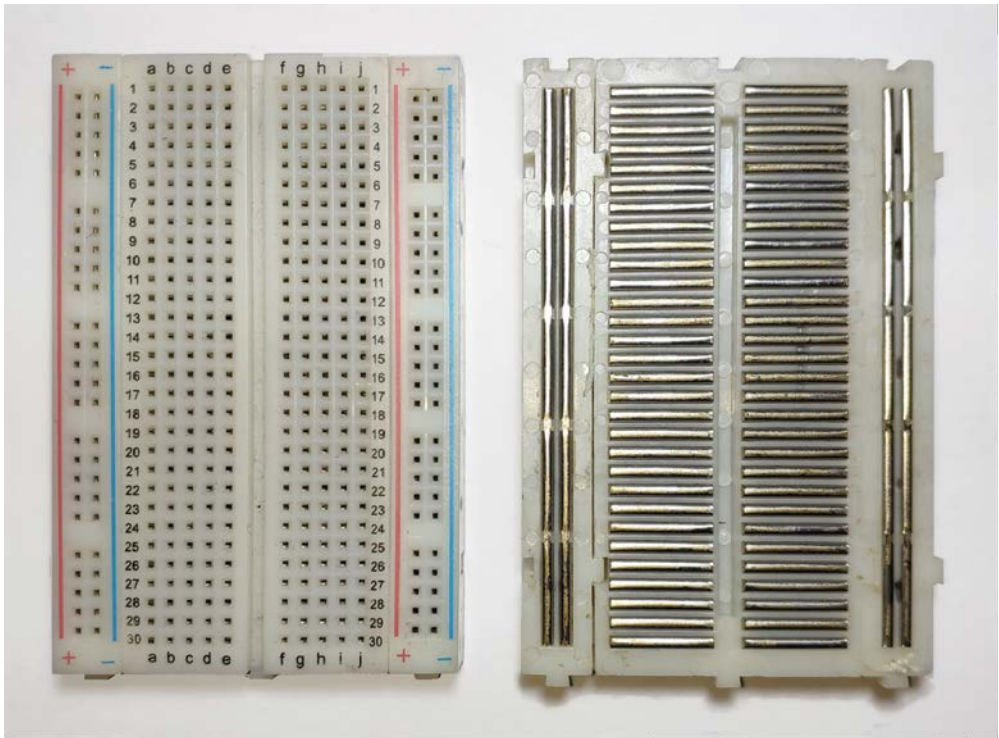


Figure 2-4: A half breadboard and its internal connections at the back side.

There is another type of similar-looking breadboard which is known as the type "GL 12" breadboard having 840 points. I will be demonstrating almost everything with a type B 102 830-point breadboard and smaller breadboards — an example is shown in Figure 2-5.

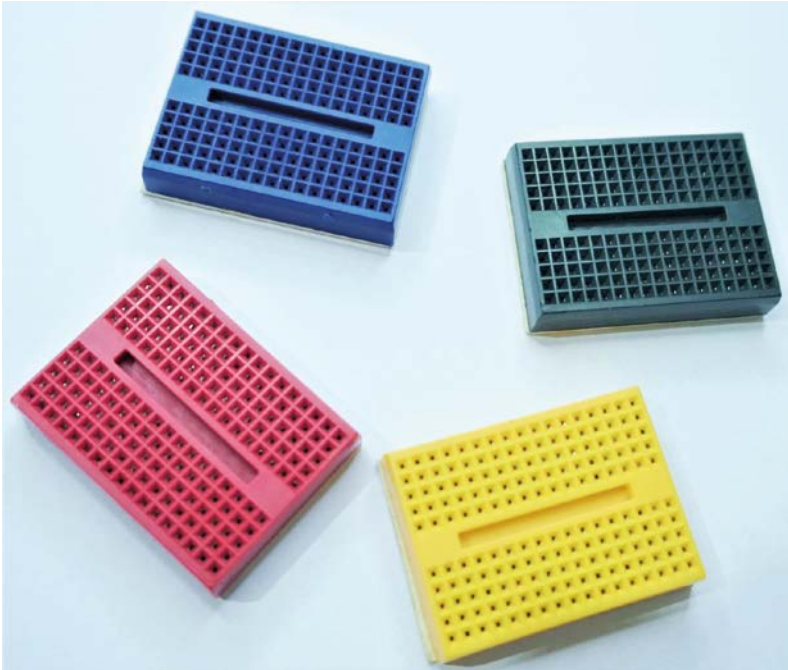


Figure 2-5: Assorted small breadboards.

You can safely use an MB 102-style power supply with an MB 102 830-point or a mini breadboard shown in Figure 2-4. Figure 2-6 shows a power supply mounted on a breadboard.

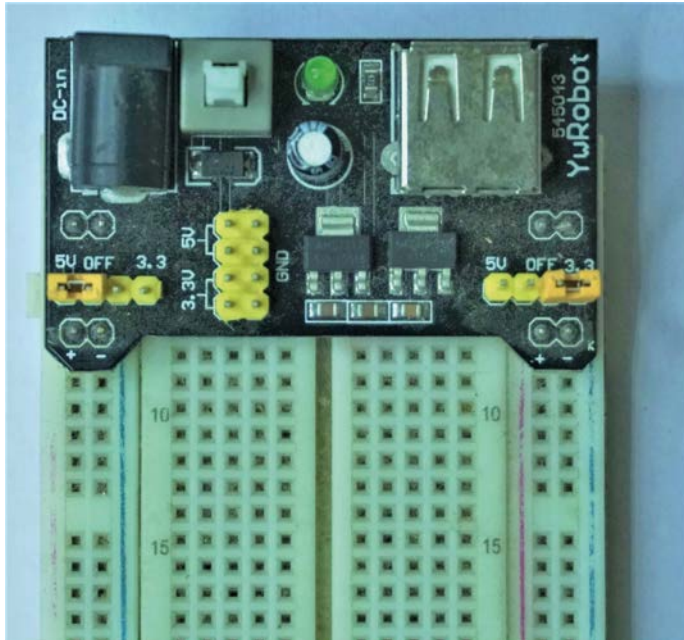


Figure 2-6: Breadboard with power supply installed at the edge.

The supply module fits perfectly on the compatible breadboard. It also has a jumper plug to choose between 3.3 V (also represented by 3V3) and 5 V. These are the most frequently used voltage levels. The power supply proper can be powered with a barrel jack with 2.1 mm center-pin positive plug adapter as shown in Figure 2-7.



Figure 2-7: AC to DC power adapter with 2.1 mm barrel jack connector.

You can also connect a 9 V block battery to a connector and connect it to a male 2.1 mm barrel jack as shown in Figure 2-8.



Figure 2-8: Portable power supply.

This arrangement can be used as a field or portable power supply for your Arduino projects.

Jumper cables

In order to connect the contact points of breadboard with each other and various electronic components, you can use jumper cables. Figure 2-9 pictures a collection of male-to-male jumper cables (image by oomlout under <https://creativecommons.org/licenses/by-sa/2.0/deed.en> license).

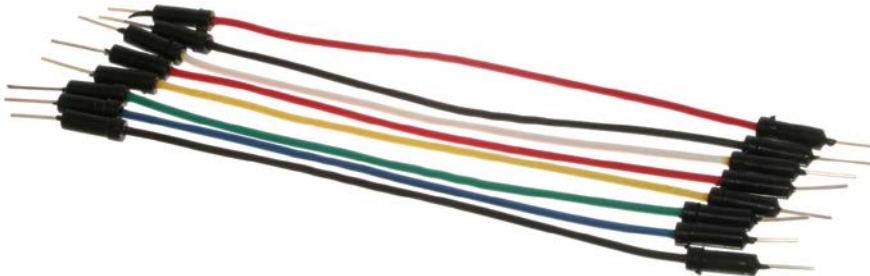


Figure 2-9: Male-to-male jumper cables for use on breadboards.

Figure 2-9 shows a female-to-female jumper cable (image by oomlout under <https://creativecommons.org/licenses/by-sa/2.0/deed.en> license),



Figure 2-10: Female-to-female jumper cable for use on breadboards.

The third type is the male-to-female jumper cable. You will be using all of these types of jumper cable throughout the demonstrations in this book.

Light Emitting Diodes

Diodes are electrical components that allow current to flow only in one direction. They have "anode" and "cathode" wire ends. If you connect the anode to a positive voltage and the cathode to negative or Ground, current will flow through the diode. However, if you connect the ends the other way round, the diode will block the current. Diodes emitting light due to current flow are known as **Light Emitting Diodes** (LEDs). Figure 2-11 (image by Gussisaurio under <https://creativecommons.org/licenses/by-sa/3.0/deed.en> license) shows blue LEDs in action.

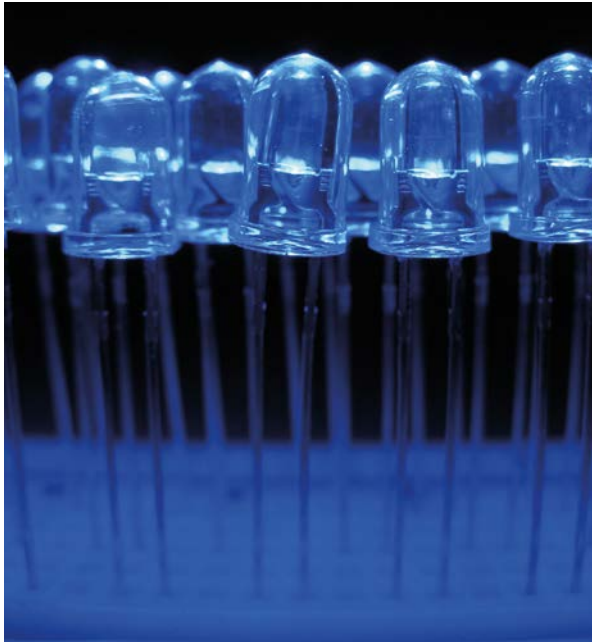


Figure 2-11: Blue LEDs.

The inventors of the blue LED won the Nobel Prize (<https://www.nobelprize.org/prizes/physics/2014/press-release/>). LEDs come in different configurations and with different colors.

Resistors

Resistors are passive electrical elements with two wire or solder ends, and they offer electrical resistance. They are used to divide voltages and limit the current in the circuits. Figure 2-12 (image by Afrank99 under <https://creativecommons.org/licenses/by-sa/2.5/deed.en> license) shows a few resistors.



Figure 2-12: Resistors.

You can calculate the value of the resistance offered by the resistor by reading the marked color codes. This page:

<https://www.digikey.in/en/resources/conversion-calculators/conversion-calculator-resistor-color-code>

offers a ready-made tool for this task. In the project descriptions, I will always mention the values of the resistors used in construction of the relevant circuit.

Pushbuttons

Pushbuttons are momentary switches. Applying pressure with your finger closes the contacts enabling current to flow. The switches used in our projects are usually rectangular in shape. Two opposite sides have two contact points each. When you apply pressure, the internal switch lever is pushed downwards and all the contact points on the same side are connected to each other. These pushbuttons are manufactured in such a way that they can easily be used on breadboards. Figure 2-13 shows the collection of pushbuttons I have in my drawer.

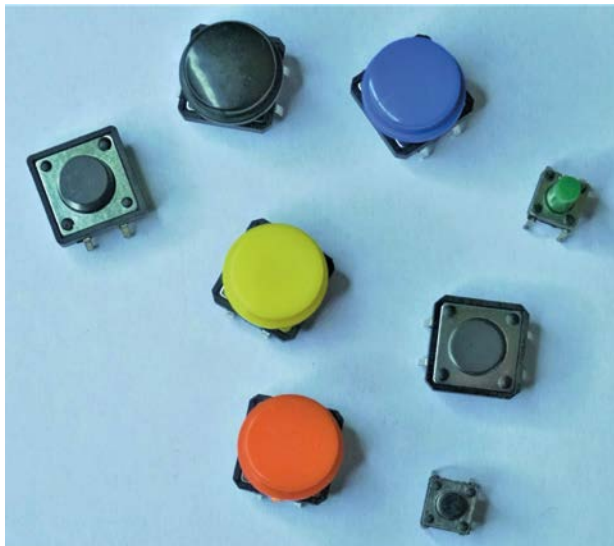


Figure 2-13: Assorted pushbuttons.

Figure 2-14 shows an assortment of pushbuttons mounted on a breadboard.

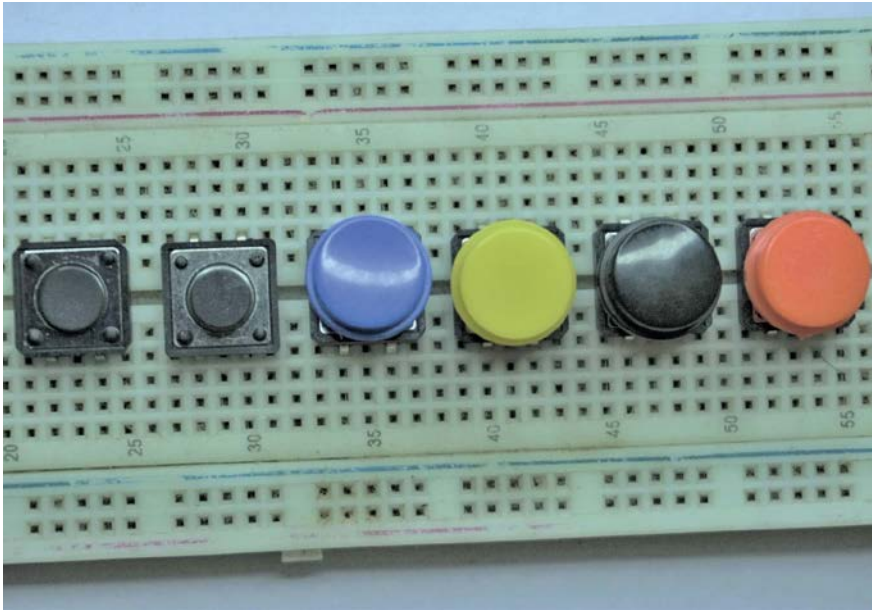


Figure 2-14: Pushbuttons mounted on a breadboard.

You will be working with a lot of electrical components and you will learn about them whenever appropriate throughout this book. Everything presented so far should be enough to get started with the basics of building a circuit.

Improving the LED blink sketch with Functions

You have written a program to flash the on-board LED. You can use the same example for learning a more complex concept. You can modify the program with the help of user-defined Functions. A function is a block of code that can be named, reused, and cladded from any other block of code. Almost all the high-level programming languages have facilities for creating functions. Depending on the programming language, they are called by various names such as routines and subroutines.

Arduino comes with a large set of functions. They are known as built-in functions. Many third-party developers create collections of user-defined custom functions for various purposes such as interfacing with hardware. These code chunks are known as libraries.

You can also define your own custom function. Let's modify the previous code example (**prog01.ino**) and create a user-defined function for the blink functionality (**prog02.ino**):

```
prog02.ino
int blink_duration = 1000;

void blink(int pin, int duration)
{
    digitalWrite(pin, HIGH);
```

```
    delay(duration);
    digitalWrite(pin, LOW);
    delay(duration);
}

void setup()
{
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
    blink(LED_BUILTIN, blink_duration);
}
```

In **prog02.ino**, you have defined a separate function to implement the blinking of an LED. You have to state the number of the digital pin to which an LED is connected to (here, a built-in constant is stated), and the duration of the blink as an argument. You can also add default arguments to this function definition. If you do not state any arguments while calling the function, the system will assume the default arguments. Refer to sketch **prog03.ino**:

```
prog03.ino
int blink_duration = 1000;

void blink(int pin=LED_BUILTIN, int duration=1000)
{
    digitalWrite(pin, HIGH);
    delay(duration);
    digitalWrite(pin, LOW);
    delay(duration);
}

void setup()
{
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
    blink();
}
```

You will continue using the skeleton of this sketch and employ similar coding standards throughout this book.

Building your first circuit on a breadboard

Let's create a simple circuit with an LED. Let me describe in words how to create a circuit for this one:

1. Mount the power supply on the breadboard and power it with a power adapter.
2. Connect the smaller leg (cathode) of an LED to the GND (Ground) pin and the longer leg (anode) to an end of a resistor with a value of 470 ohms. Connect the other end of the resistor to the positive voltage.
3. Switch on the power supply and the LED will light up.

Congrats! You have built your first digital circuit; it's shown in Figure 2-15.

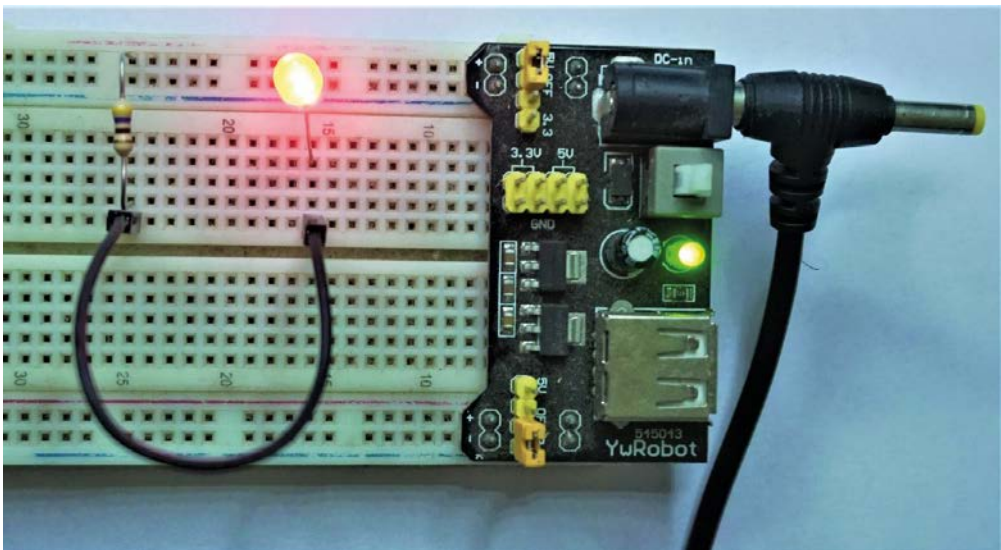


Figure 2-15: Simple LED circuit built on a breadboard with a plug-on power supply module installed.

Now, after reading the description and observing the image, you must have understood that it is difficult to show circuits using only words and photographs. That's why I am going to use a tool known as Fritzing (read more at <https://fritzing.org/>) to show the circuit diagrams. If you wish, you can also purchase this tool costing 8 euros. However, it is worth every single cent you pay for it. In the code bundle released for this book, I have included the Fritzing circuit files for your reference if you wish to work with them. Figure 2-16 is the Fritzing-style "circuit diagram".

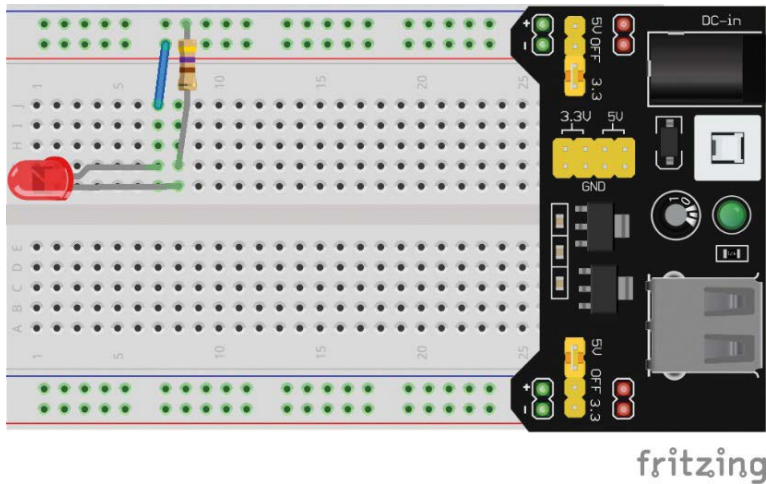


Figure 2-16: Simple LED on a breadboard.

You can find the Fritzing parts for the breadboard and Arduino Nano Every board at the following URLs:

1. <http://omnigatherum.ca/wp/?p=262>
2. <https://docs.arduino.cc/hardware/nano-every>

From here on, I will mostly use Fritzing images to represent circuits. I will also add the URLs of the Fritzing files for the parts that do not come with Fritzing.

Let's make this circuit a little bit more sophisticated. Let's add a pushbutton. Figure 2-17 shows a (Fritzing) circuit with a pushbutton

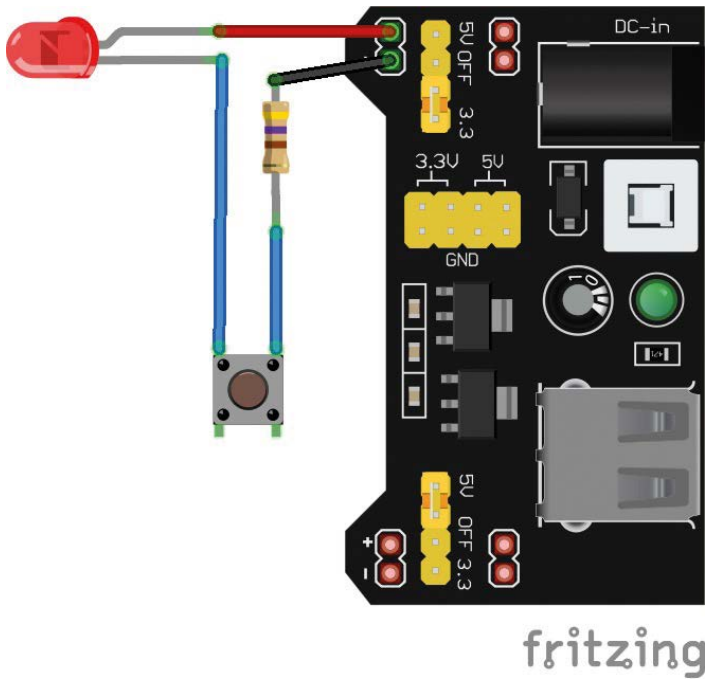


Figure 2-17: Circuit of a simple LED with a switch.

Circuits using Nano

Let's proceed by building a circuit using the Arduino Nano. Refer to Figure 2-18.

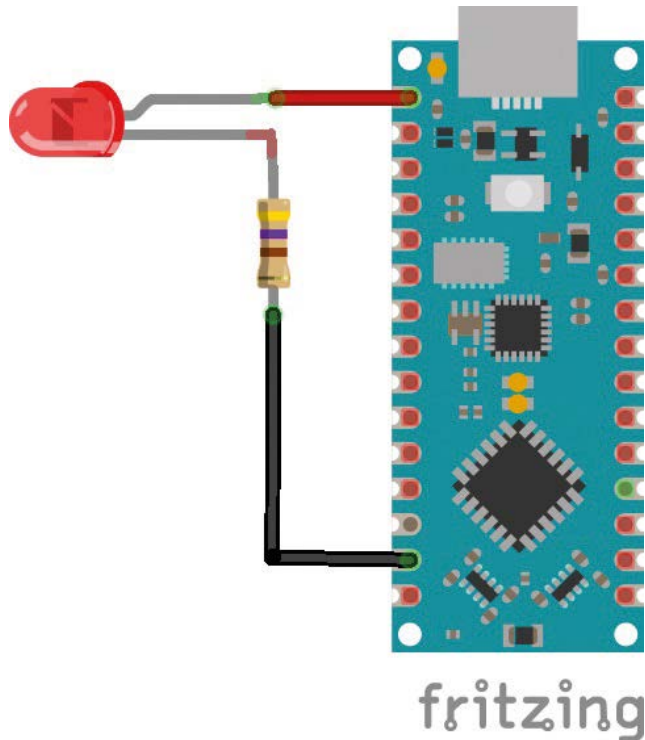


Figure 2-18: External LED hooked up to digital pin 13.

Let's connect an external LED to the digital pin 13. Connect the anode of the LED to digital pin 13 and then connect the cathode of the LED to the GND pin through a 470 ohm resistor. After this, upload the sketch **prog03.ino** to the Nano board. The LED will start blinking.

NOTE: I am using the **Arduino Nano Every** for most of the demonstrations in this book. Feel free to use the **Arduino Nano** instead. Just remember to select the correct board from the **Tools** menu.

Working with multiple LEDs

Let's connect multiple LEDs to the Nano board. Use pins D2 to D9 to connect eight LEDs as shown in Figure 2-19.

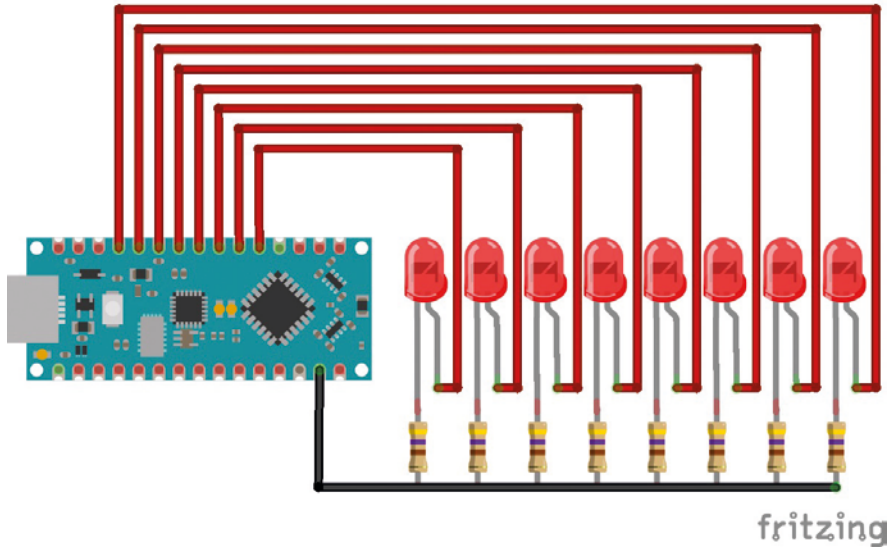


Figure 2-19: Multiple external LEDs connected to pins D2 through D9.

All the resistors are 470 ohm types You can turn on all the LEDs under software control with the aid of **prog04.ino**:

```
prog04.ino
void setup()
{
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
}

void loop()
{
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(9, HIGH);
}
```

Alternatively, you can make them blink (flash) with the following code:

```
prog05.ino
void setup()
{
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
}

void loop()
{
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(9, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  delay(1000);
}
```

Isn't this program too long? You can reduce the length of the program by using arrays and loops as follows:

```
prog06.ino
int pins[8] = {2, 3, 4, 5, 6, 7, 8, 9};

void setup()
{
  for ( int i=0; i<=7; i++ )
```

```
    pinMode(pins[i], OUTPUT);
}

void loop()
{

for ( int i=0; i<=7; i++ )
    digitalWrite(pins[i], HIGH);
delay(1000);

for ( int i=0; i<=7; i++ )
    digitalWrite(pins[i], LOW);
delay(1000);
}
```

This is a very compact way of writing programs. You are storing all the pin numbers in an array and then accessing them with loops to get the desired results.

You can modify the earlier sketch to make the LEDs blink in succession. This is known as the LED chaser effect. The sketch is as follows.

```
prog07.ino
int blink_duration = 50;

void blink(int pin=LED_BUILTIN, int duration=1000)
{
    digitalWrite(pin, HIGH);
    delay(duration);
    digitalWrite(pin, LOW);
    delay(duration);
}

int pins[8] = {2, 3, 4, 5, 6, 7, 8, 9};

void setup()
{
    for ( int i=0; i<=7; i++ )
        pinMode(pins[i], OUTPUT);
}

void loop()
{
    for ( int i=0; i<=7; i++ )
        blink(pins[i], blink_duration);
}
```

Upload the sketch to see the compelling chaser effect.

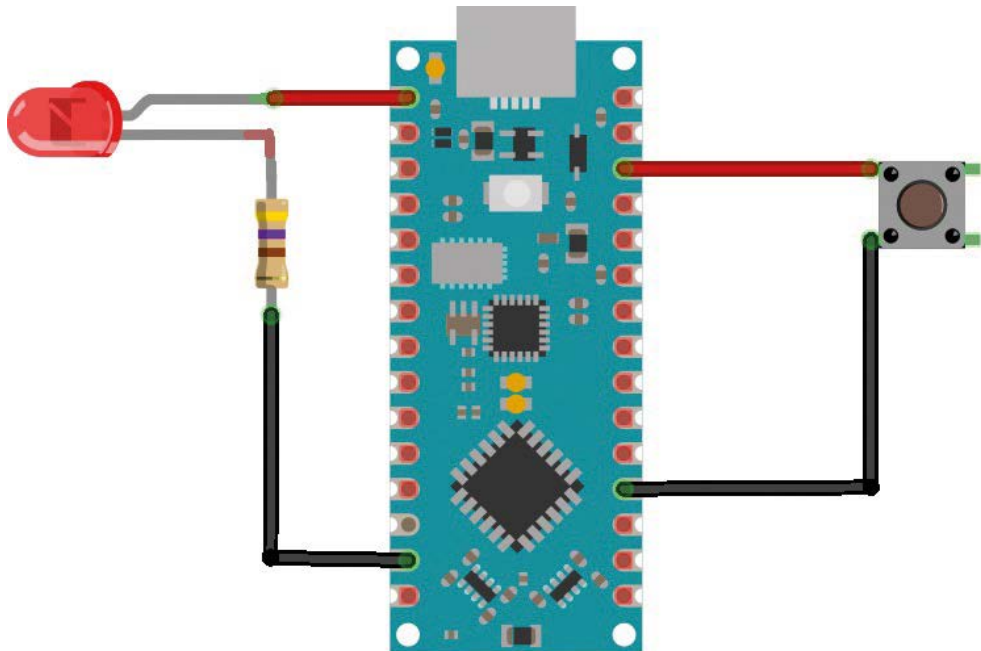
You can also implement a 4-bit binary counter by using only the first four LEDs connected to pins D2, D3, D4, and D5:

```
prog08.py
int pins[8] = {2, 3, 4, 5};
int i = 0;
void setup()
{
  for ( int i=0; i<=3; i++ )
    pinMode(pins[i], OUTPUT);
}
void loop()
{
  digitalWrite(pins[3], LOW);
  digitalWrite(pins[2], LOW);
  digitalWrite(pins[1], LOW);
  digitalWrite(pins[0], LOW);
  i++;

  if((i % 2) > 0) { digitalWrite(pins[0], HIGH); } else { digitalWrite(pins[0],
LOW); }
  if((i % 4) > 1) { digitalWrite(pins[1], HIGH); } else { digitalWrite(pins[1],
LOW); }
  if((i % 8) > 3) { digitalWrite(pins[2], HIGH); } else { digitalWrite(pins[2],
LOW); }
  if((i % 16) > 7) { digitalWrite(pins[3], HIGH); } else { digitalWrite(pins[3],
LOW); }
  delay(1000);
}
```

Adding a pushbutton to the circuit

Let's modify the single external LED circuit (where the LED is connected to digital pin 13) and add a pushbutton to that. See Figure 2-20.



fritzing

Figure 2-20: Adding a pushbutton.

Till now, you have learned how to use the digital pins as outputs. It is possible to use a digital pin as an input, though. You have to connect a pushbutton to it, as shown in Figure 2-10, where one end of the pushbutton is connected to D10, and another is connected to the GND pin. Have a look at the following sketch.

```
prog09.ino
void setup()
{
  pinMode(10, INPUT_PULLUP);
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  int sensorVal = digitalRead(10);
  if (sensorVal == HIGH)
    digitalWrite(LED_BUILTIN, LOW);
  else
    digitalWrite(LED_BUILTIN, HIGH);
}
```

Every digital pin of the Arduino has a built-in pull-up resistor. When you initialize a digital pin with the mode **INPUT_PULLUP**, it produces a LOW signal when pressed. When the button is in the normal state, it is HIGH. The function **digitalRead()** reads the value of the digital pin passed as the argument. This sketch makes the LED light when the pushbutton is pressed.

Working with RGB LEDs

So far, you worked with LEDs of a single color. Now, you will learn the basics of RGB LEDs. These are essentially three different LEDs in a single package. An RGB LED has four pins. One pin is a common anode (to be connected to a positive supply level) or a common cathode (to be connected to Ground). The rest of the pins are for red, green, and blue LEDs, and they can be connected to a digital output pin. In the case of the **common-cathode** RGB LED, you need to send a HIGH signal to a LED pin for the LED to come on. With the **common-anode** RGB LED, you need to send a LOW signal to a LED pin for the LED to light. Figure 2-21 shows the circuit for common-cathode LED. All resistors have a value of 470 ohms.

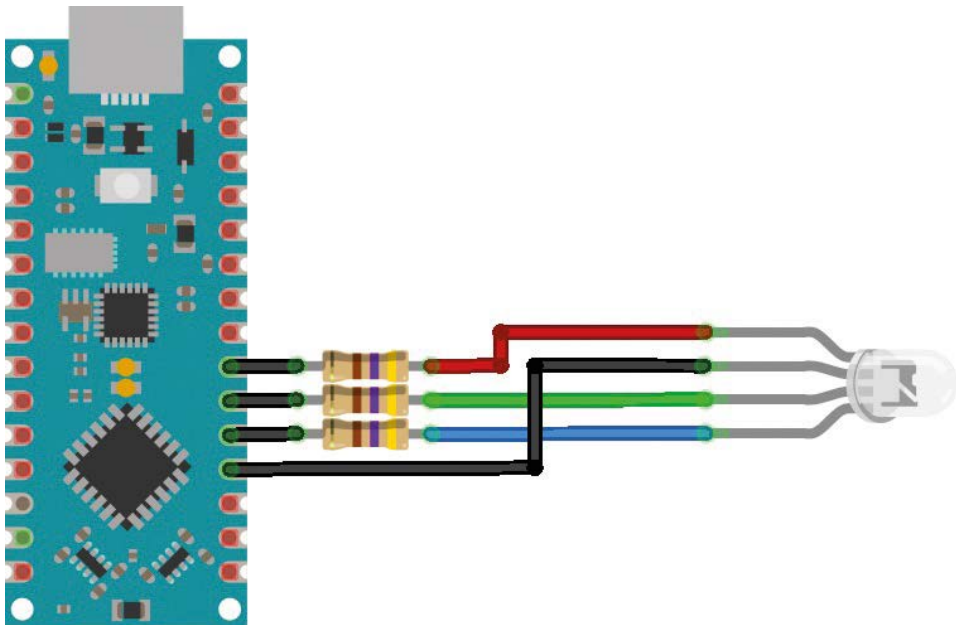


Figure 2-21: Common-cathode RGB LED control circuit.

The sketch (**prog10.ino**) uses a binary counter to show all the combinations of colors possible by means of digital output. It shows 7 colors and an Off state (i.e., 8 combinations using three digital output pins).

prog10.ino

```
int duration = 200;

void setup()
{
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}

void loop()
{
  digitalWrite(2, HIGH); digitalWrite(3, HIGH); digitalWrite(4, HIGH);
  delay(duration);
  digitalWrite(2, HIGH); digitalWrite(3, HIGH); digitalWrite(4, LOW);
  delay(duration);
  digitalWrite(2, HIGH); digitalWrite(3, LOW); digitalWrite(4, HIGH);
  delay(duration);
  digitalWrite(2, HIGH); digitalWrite(3, LOW); digitalWrite(4, LOW);
  delay(duration);
  digitalWrite(2, LOW); digitalWrite(3, HIGH); digitalWrite(4, HIGH);
  delay(duration);
  digitalWrite(2, LOW); digitalWrite(3, HIGH); digitalWrite(4, LOW);
  delay(duration);
  digitalWrite(2, LOW); digitalWrite(3, LOW); digitalWrite(4, HIGH);
  delay(duration);
  digitalWrite(2, LOW); digitalWrite(3, LOW); digitalWrite(4, LOW);
  delay(duration);
}
```

If necessary, you can replace the common-cathode RGB LED with a common-anode RGB LED. For that you have to adapt the circuit as shown in Figure 2-22.

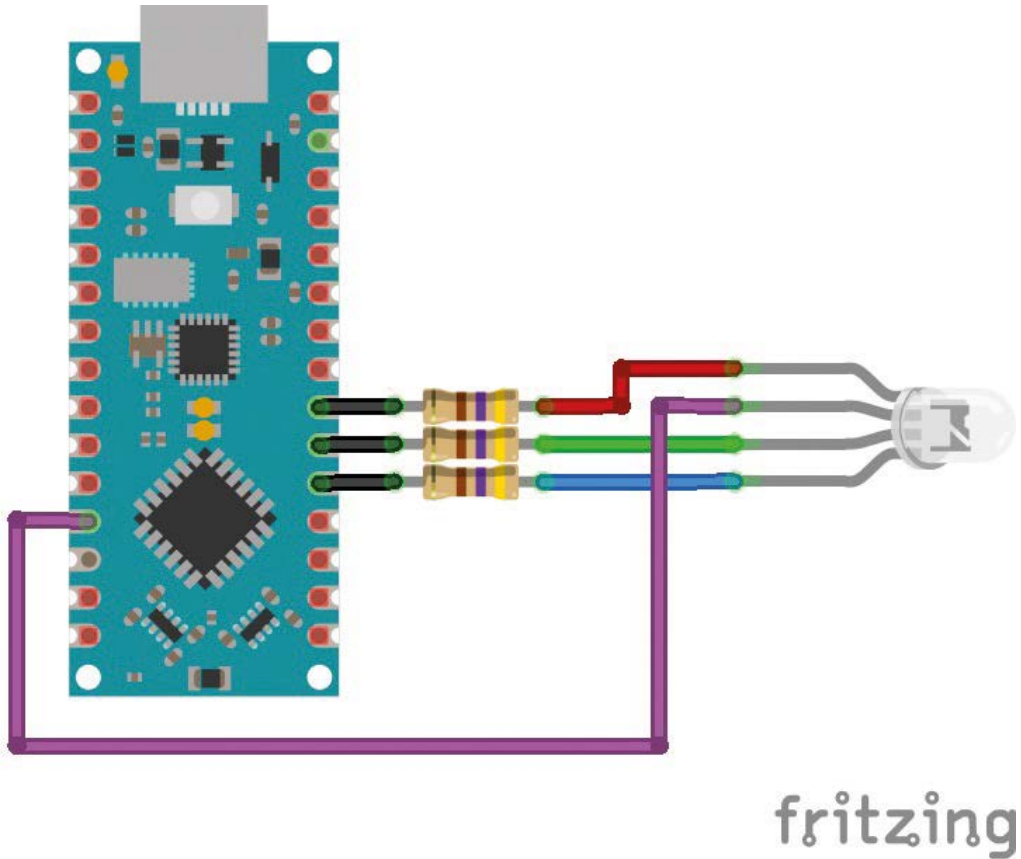


Figure 2-22: Common-anode RGB LED circuit.

The sketch though will not need any modification to suit the common-anode RGB LED.

Using Arduino Nano boards with expansion shields

It may be difficult to work with an Arduino Nano in combination with a breadboard. You may have experienced the difficulty in working with pins spaced so close together after so many demonstrations. That is why many third-party manufacturers produce various bases and I/O expansion boards. Figure 2-13 shows a simple example of such a board.

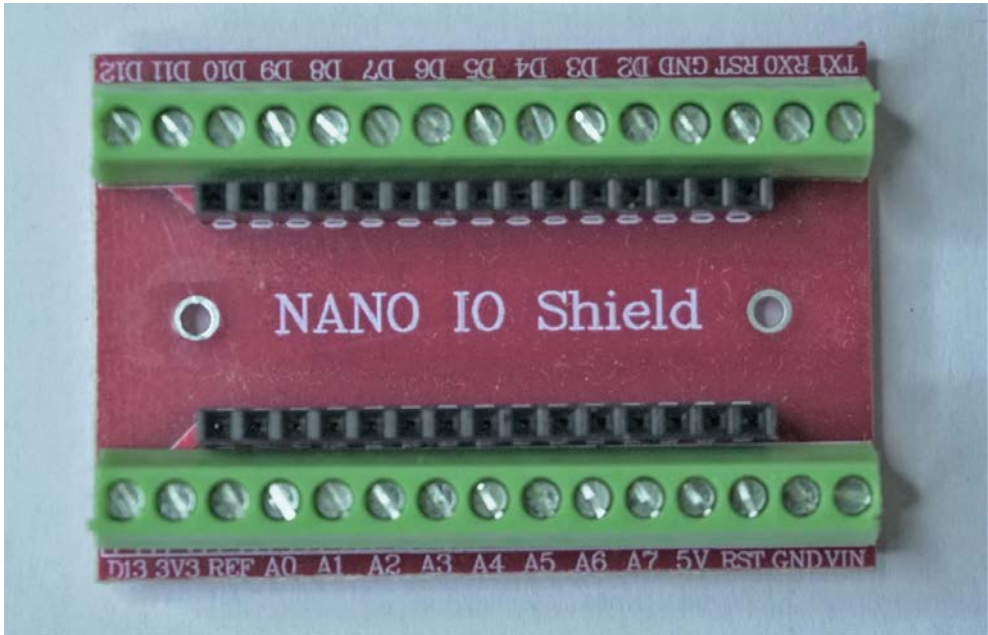


Figure 2-23: Nano I/O shield.

If you are planning to use Nano boards for industrial applications, then this is a good option. It has the bare minimum requirements such as marked headers and provision to attach wires for your control application.

In case you are looking for a more complex application with a dedicated power supply, then the shield shown in Figure 2-24 is a good choice. It has separate sections of pins for Analog, Digital, I²C, UART, and SPI. In addition, it has a dedicated power supply system.

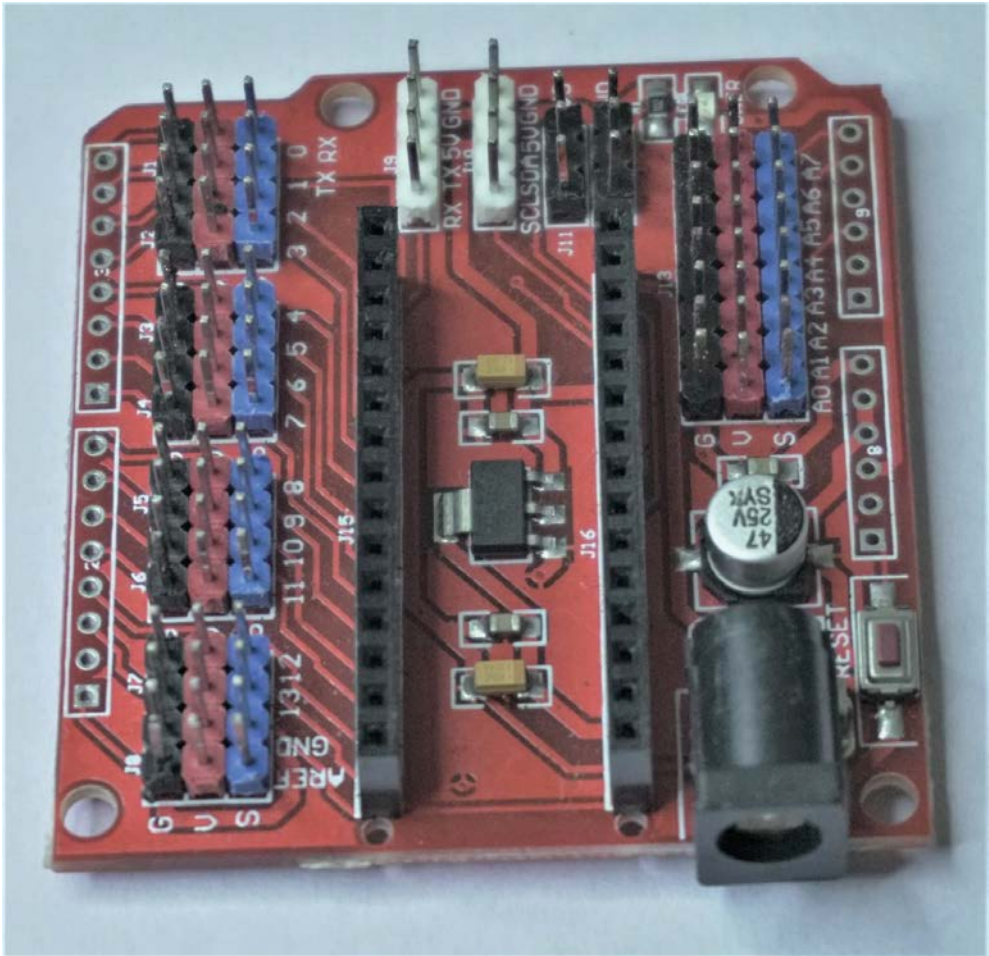


Figure 2-24: Nano I/O expansion shield.

As you can see, it has a similar form factor as the Arduino Uno. Analog and digital pins have dedicated groups for GND, VCC, and Signal. You can mount a Nano board on it as shown in Figure 2-25.

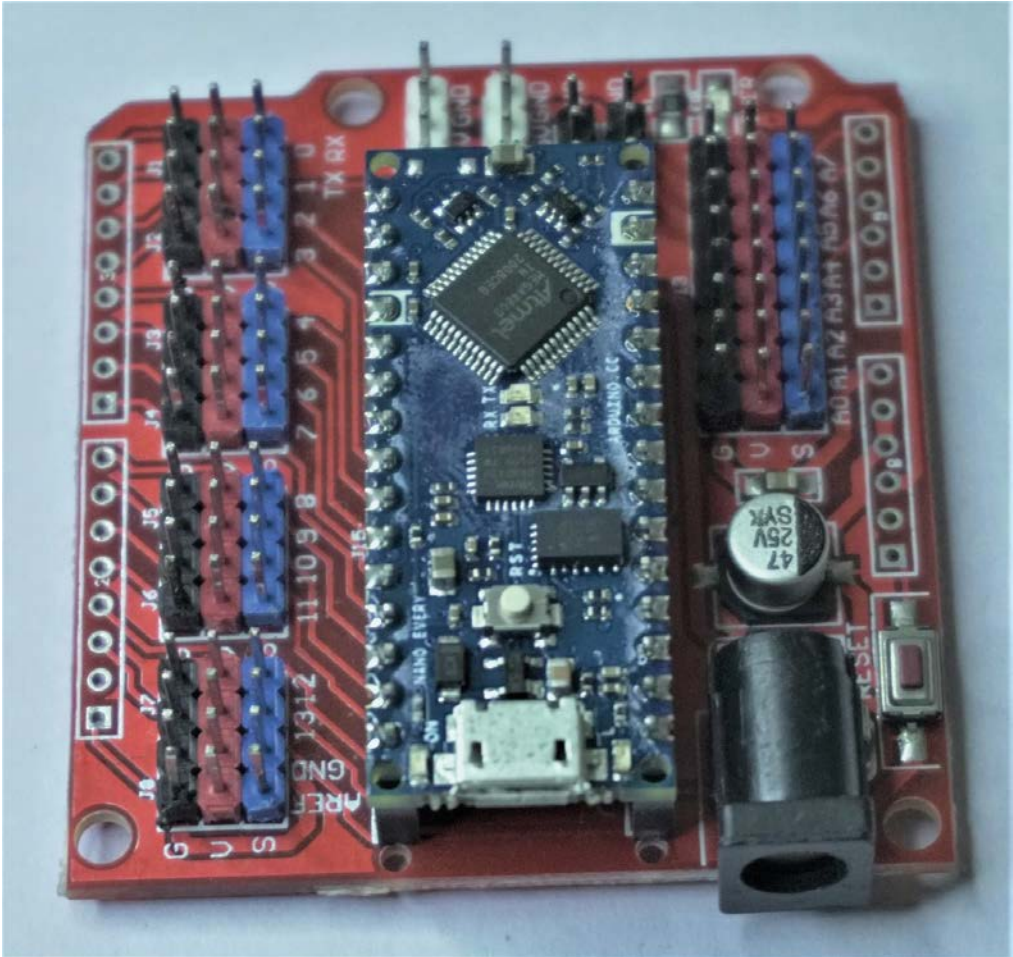


Figure 2-25: Nano I/O expansion shield with a mounted Nano Every board.

Summary

In this chapter, you explored the digital input and output aspects of the Arduino Nano in detail. You worked with various electronic components, built your first circuit, and wrote your first Arduino sketch.

In the next chapter, you will learn how to deal with various types of Arduino-supported buses. You will also learn how to work with analog input using the Arduino's analog pins.

Chapter 3 • Assorted Buses and the Analog Input

In the previous chapter, you learned the basics of electronic circuits and programming with Arduino. Now, you are reasonably comfortable with building basic circuits and beginner's level programming.

You will continue the exciting journey of Arduino programming. In this chapter, you will learn the following topics:

- parallel and serial data transfer
- Arduino serial
- SPI and I²C
- analog input
- plotting multiple variables

After finishing this chapter, you will be at ease with various types of bus systems and the concept of analog input.

Parallel and Serial Data Transfer

In the world of computing, there are two ways to transfer electronic signals (HIGH, LOW, and analog), Parallel and Serial. A bus is a communication system for data transfer. In Parallel data transfer (also known as **parallel bus**), the signals are transferred by multiple bus lines as shown in Figure 3-1.

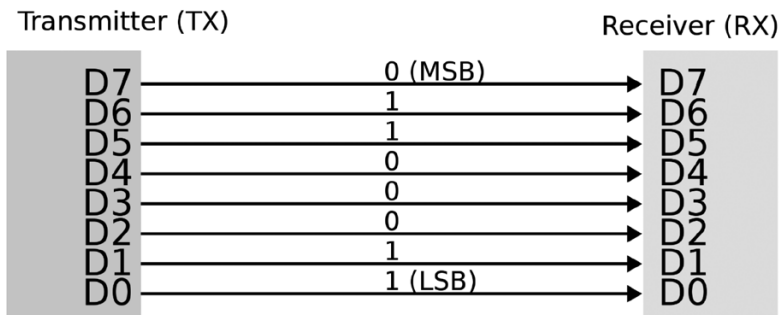


Figure 3-1: Parallel data transfer.

As you can see, there is a separate bus line for every bit. This method is very fast as more data can be carried in less time. However, this method is not particularly efficient as it requires a lot of hardware, and, consequently, is expensive to implement. Most of the internal buses in a microprocessor, such as the data bus, address bus, and control bus are parallel buses.

A more efficient data transfer technique is Serial communication (or **serial bus**). Figure 3-2 represents a serial bus.

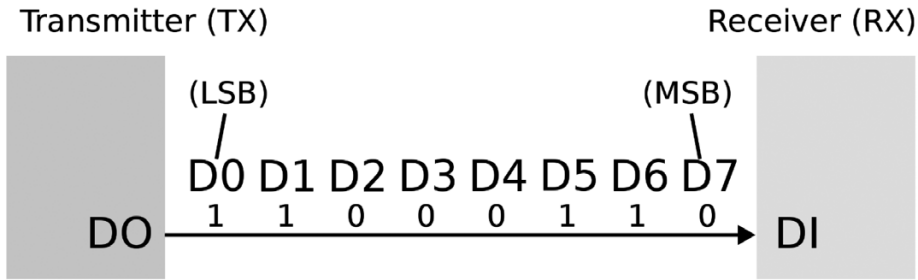


Figure 3-2: Serial data transfer.

In serial communication systems, data bits are transferred in series, i.e., sequentially, over a single channel. Usually, a serial communication system has a transmitter (TX) and receiver (RX) pair for bidirectional communication. The serial data transfer is slower but more efficient in terms of hardware and cost. There are two types of serial communication: asynchronous and synchronous. In asynchronous serial communication, the TX and RX are not synchronized by any clock but by the data bits. The best and the most commonly used example of asynchronous serial communication is RS-232. Figure 3-2 shows "schematics" of male and female connectors for a pair of RS-232 communication devices.

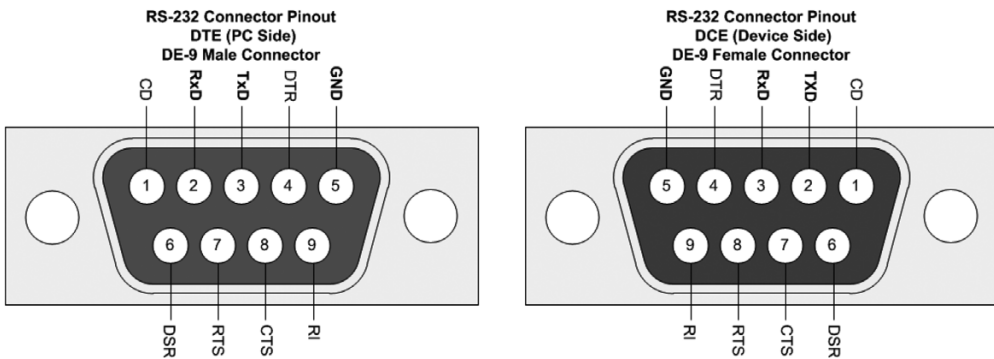


Figure 3-3: RS-232 male and female connectors (image by Cody.hyman under the <https://creativecommons.org/licenses/by-sa/3.0/deed.en> license, modified for publication)

Another example of asynchronous serial is the UART (universal asynchronous receiver-transmitter). You will learn this in detail in this chapter.

The other type of serial data transfer is synchronous data transfer. Here, the endpoints and data transfer are synchronized by clock pulses. There are many ways this can be implemented. Prominent examples are the USART (universal synchronous and asynchronous receiver-transmitter), Serial Peripheral Interface (SPI), and Inter-Integrated Circuit (I²C or simply I2C). This chapter explores them in depth.

Arduino Serial

All Arduino boards have at least one set of serial pins that are used for UART and USART. The Arduino Nano and Arduino Nano have one set of serial pins. Digital Pins D0 and D1 serve a dual purpose: they are used for digital I/O while also configurable for Serial data transfer. Since Arduino Nano boards have only one set of them, from here on, we will leave pins D0 and D1 free for serial data transfer and will use other digital pins for the digital I/O. D0 serves as RX and D1 serves as TX on both boards. We can monitor the data transfer on these boards by using the built-in tool known as the **Serial Monitor**. It can be found under the menu **Tools** as shown in Figure 3-4 .

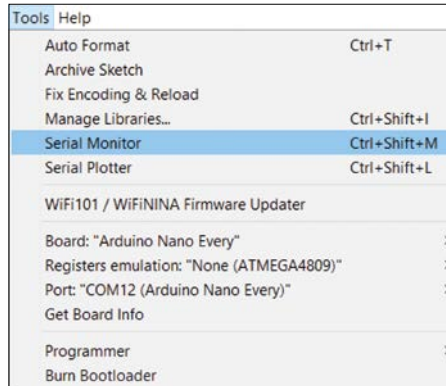


Figure 3-4: Serial Monitor found.

The window that opens looks like in Figure 3-5.

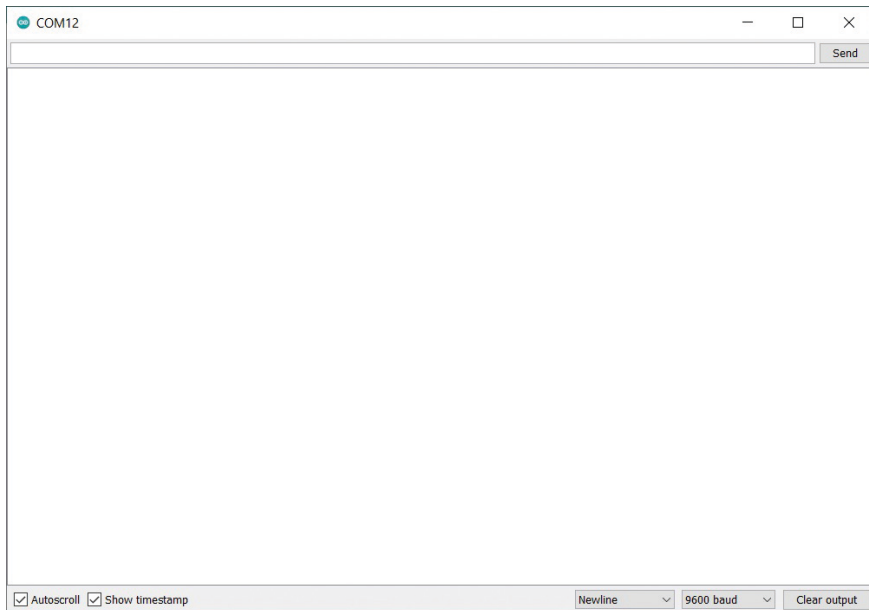


Figure 3-5: Serial Monitor opened under Tools.

Make sure that you check all the checkboxes at the bottom left. At the bottom right, select **Newline** from the first dropdown and **9600 baud** from the second dropdown as shown in Figure 3-5. Let's play with this. Keep your eyes fixed on the Arduino board and type in something in the textbox at the top. Then click the **Send** button or press the **Enter** key on the keyboard. This will send the data as input to the RX pin. The LED associated with RX will light briefly. If you send a very large string with, say, 200 characters, you will find that the LED lights for a longer time.

Let's transmit some data over TX. All the data transmitted over TX is printed on the serial console. Recall **prog03.ino** from the last chapter and modify it and save it with a new filename under a new directory for this demo (I recommend organizing your code in chapter-named directories as I am doing). Here is the modified code:

```
prog00.ino
void blink(int pin=LED_BUILTIN, int duration=1000)
{
    digitalWrite(pin, HIGH);
    Serial.println("LED On..");
    delay(duration);
    digitalWrite(pin, LOW);
    Serial.println("LED Off..");
    delay(duration);
}

void setup()
{
    pinMode(LED_BUILTIN, OUTPUT);
    Serial.begin(9600);
}

void loop()
{
    blink();
}
```

The on-board LED connected to D13 (and any external one, if connected), will start blinking. Also, the LED associated with the TX pin will blink momentarily every time the D13 LED blinks. This is because you are sending data over "Serial". In the serial monitor tool, you will see output like in Figure 3-6.

The screenshot shows a Serial Monitor window titled 'COM12'. The main area displays a series of timestamped messages: '13:02:38.048 -> LED Off..', '13:02:39.032 -> LED On...', '13:02:40.062 -> LED Off..', '13:02:41.047 -> LED On...', '13:02:42.033 -> LED Off..', '13:02:43.016 -> LED On...', '13:02:44.049 -> LED Off..', '13:02:45.033 -> LED On...', '13:02:46.016 -> LED Off..', '13:02:47.048 -> LED On...', '13:02:48.032 -> LED Off..', '13:02:49.014 -> LED On...', '13:02:50.043 -> LED Off..', and '13:02:51.028 -> LED On...'. The bottom status bar shows 'Autoscroll' and 'Show timestamp' checked, 'Newline' selected, '9600 baud', and 'Clear output'.

Figure 3-6: Serial Monitor output.

You may have missed the `printf()` statement in C programming normally used to debug programs. Well, `Serial.println()` offers the similar functionality.

SPI and I²C

SPI means **Serial Peripheral Interface** and is a synchronous serial communication standard. On the Nano board, you can find D10 (CS – Chip Select), D11 (COPI – Controller Out Peripheral In), D12 (CIPO – Controller In Peripheral Out), and D13 (SCK – Serial Clock) pins to support SPI Communication.

I²C means **Inter-Integrated Circuit** and is also a synchronous serial communication standard. Nano boards support it with the pins A4 (SDA – Serial Data) and A5 (SCL – Serial Clock). It is supported with **Two Wire Interface** using the built-in **Wire** library.

You will learn to connect various devices to Arduino Nano boards that use these special buses throughout this book.

Analog Input

In the previous chapter, you learned to work with the digital pins and pushbuttons. In this section, you will learn to use the analog input pins of Nano boards. Nano boards have 8 different analog input pins (A0 through A7) and their resolution is 10 bits. Therefore, the input can have $2^{10} = 1024$ different values. Those values range from 0 through 1023. Let's build a circuit that reads input analog value of a potentiometer or a variable resistor. Figure 3-7 shows that device.



Figure 3-7: Potentiometer. (image by oomlout under <https://creativecommons.org/licenses/by-sa/2.0/deed.en> license)

Potentiometers are usually operated with a knob fixed on a rotating axis. You can also use a trimpot shown in Figure 3-8 as a variable resistor that is a bit more breadboard-friendly.



Figure 3-8: A trimpot-style variable resistor (image by oomlout under <https://creativecommons.org/licenses/by-sa/2.0/deed.en> license)

Let's use a trimpot variable-resistor with a range 0-1000 ohms for the analog input. Nearly all trimpots acting as a variable resistance (including potentiometers) have three pins. The middle pin (called wiper) outputs the signal. It should be connected to an analog input pin of an Arduino board. Of the two remaining pins, one pin can be connected to the reference voltage, and the remaining one can be connected to Ground. Note that the order is not strict, which is why I do not mention exactly which pin in the earlier statement. Just make sure that one pin on the extreme end is connected to 5 V, the middle pin is connected to A0, and the last one is connected to the GND, as shown in Figure 3-9.

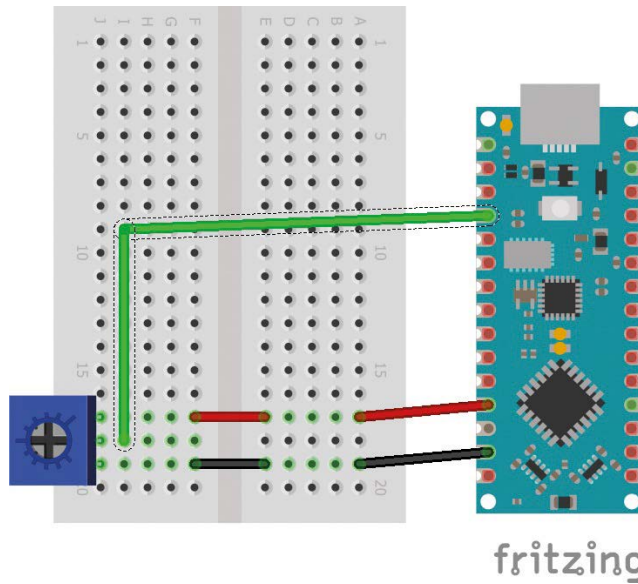


Figure 3-9: A 1 k-ohm (1000 ohm) trimpot used as a variable resistor connected to the Arduino Nano Every.

Now, create and upload the following sketch.

```
prog01.ino
int resistor;
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  resistor = analogRead(0);
  Serial.println(resistor);
  delay(100);
}
```

After uploading the sketch, open the serial monitor, and you will see something like in Figure 3-10.

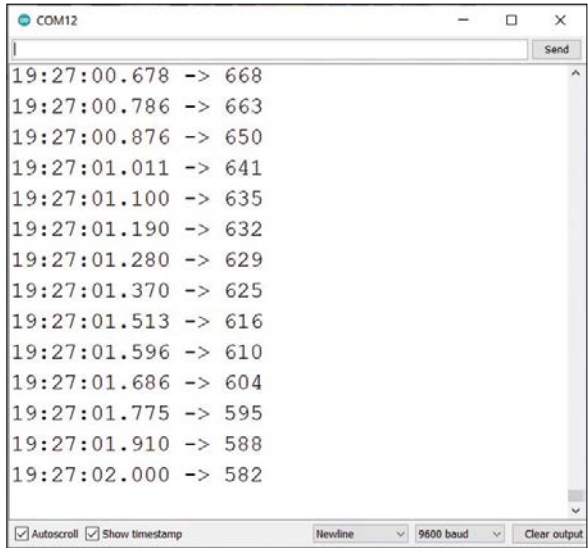


Figure 3-10: Analog input in active service.

You can also see everything graphically with the serial plotter. Figure 3-11 shows how to open the serial plotter.

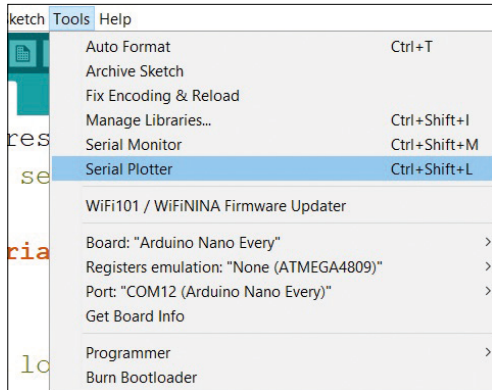


Figure 3-11: Tracking the analog input values with the Serial Plotter utility.

The graphical output will look something like Figure 3-12.

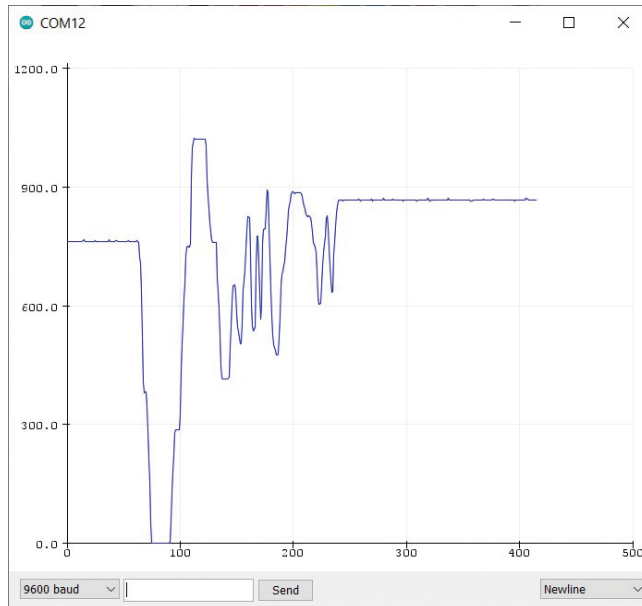


Figure 3-12: Using the Serial Plotter provides a graphical view of data values.

The trimpot resistor or potentiometer is essentially working as a variable voltage divider allowing you to change the level of the reference voltage. We can map the range of analog input (0 – 1024) to the corresponding voltage range (0 – 5 volts) as follows.

```
prog02.ino
int resistor;
float voltage;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  resistor = analogRead(0);
  voltage = resistor * ( 5.0 / 1023.0 );
  Serial.println(voltage);
  delay(100);
}
```

The Arduino library has a built-in function **map()** that accepts a variable, a pair of input ranges, and a pair of custom output ranges and then maps the input to the output fitting the custom range. For example, you may want to convert the analog input from range 0 to 1023 to 0 to 255. It is done as follows.

```
prog03.ino
int resistor;
float mapped_range;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  resistor = analogRead(0);
  mapped_range = map (resistor, 0, 1023, 0, 255) ;
  Serial.println(mapped_range);
  delay(100);
}
```

Plotting multiple variables

You can plot multiple variables on the serial plotter as follows:

```
prog04.ino
float x, y1, y2, y3;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  for(x = 0; x < 360; x = x + 5)
  {
    y1 = x * x;
    y2 = -x * x;
    y3 = 100000 * cos(x * (3.1417 / 180));
    Serial.print(y1);
    Serial.print(" ");
    Serial.print(y2);
    Serial.print(" ");
    Serial.println(y3);
    delay(1);
  }
}
```

This causes all the variables to be plotted in different colors. You can see the output in Figure 3-13.

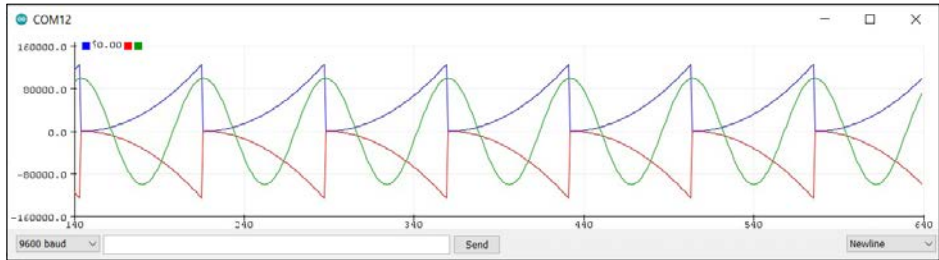


Figure 3-13: Plotting multiple variables.

Summary

In this chapter, you learned the basics of Arduino Serial and other buses. In addition, you have learned to use the analog inputs and serial plotter.

In the next chapter, you will learn to use various displays in combination with Arduino Nano boards.

Chapter 4 • Pulse Width Modulation and Driving Unipolar Stepper Motors with Digital I/O

In the previous chapter, you studied and practiced the Arduino serial, serial plotting, and analog inputs. You also learned the basics of various buses and pins associated with them.

In this chapter, you will explore the amazing world of **Pulse Width Modulation** (abbreviated as PWM; alternative spelling: pulsewidth modulation). Here are the topics you will study and demonstrate in this chapter:

- the concept of pulse width modulation
- PWM with the Arduino Nano
- working with a servo motor
- working with the 28BYJ-48 unipolar stepper motor and the ULN2003A motor driver
- using a custom library for stepper motors

Once you finish the chapter, you will be relaxed with the concept of PWM and its applications.

The Concept of Pulse Width Modulation

You know that a normal digital signal has one of two states for every period: either HIGH (5 V/3V3 or digital 1) or LOW (0 V or digital 0). The length of time the waveform takes to repeat itself is known as period. Figure 4-1 is an example of a digital signal,

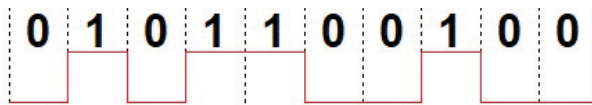


Figure 4-1: A digital signal carrying information. (adapted from the image provided by El Pak under the <https://creativecommons.org/licenses/by-sa/3.0/deed.en> license)

So, within the entire period, the signal is either HIGH or LOW. It is possible to modify the digital signal in such a way that a part of the signal is HIGH and the remaining part is LOW within a single period (or a single pulse). This is known as modulation of the width of the pulse (PWM). Figure 4-2 shows a modulated digital signal.

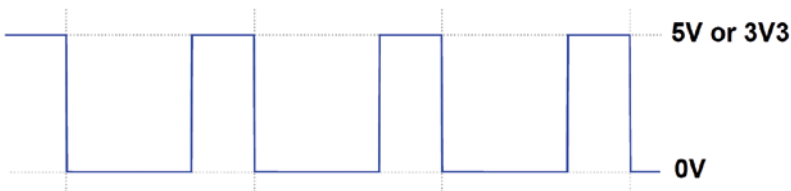


Figure 4-2: A modulated digital signal. (adapted from the image by Cyril BUTTAY under the <https://creativecommons.org/licenses/by-sa/3.0/deed.en> license)

Notice that in a single period pulse, you can see both signal levels, HIGH and LOW. Essentially, you are manipulating the width of the HIGH signal. Figure 4-3 shows the anatomy of a width-modulated digital pulse.

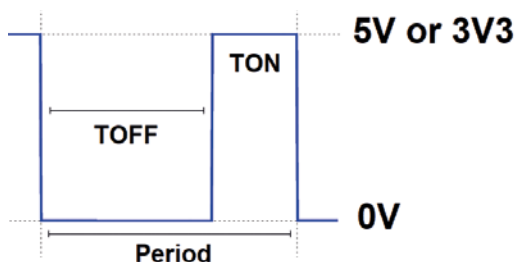


Figure 4-3: Anatomy of a width-modulated digital pulse.

In a single pulse, the time for which the pulse is LOW is known as TOFF or T_{OFF} and the time for which it is HIGH is known as TON or T_{ON} . The following is the formula that shows the relationship between the period, T_{ON} , and T_{OFF} .

$$\text{period of a signal} = T_{ON} + T_{OFF}$$

The duty cycle (also known as the power cycle) is the fraction of period for which the pulse is active. It is determined in percentage with the following formula,

$$\text{duty cycle} = (T_{ON} / \text{period}) \times 100$$

Figure 4-4 shows various duty cycles.

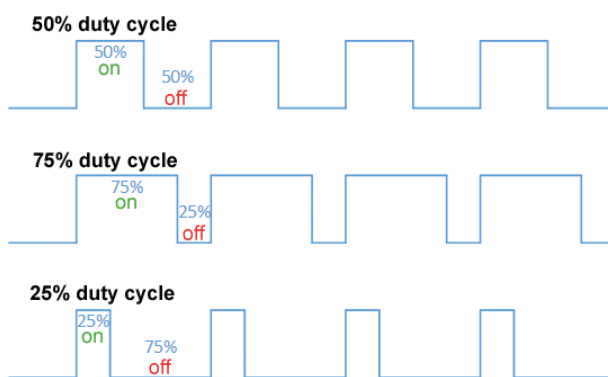


Figure 4-4: Duty cycle (adapted from the image by Thewrightstuff under the <https://creativecommons.org/licenses/by-sa/4.0/deed.en> license)

Thus, you can cleverly use Arduino PWM for delivering the desired amount of power to an LED or any other device of your choice.

PWM with Arduino Nano

You can use the digital pins D3, D5, D6, D9, and D10 for PWM on both the Nano and Nano Every boards. The digital pin D11 on the Nano can be used for PWM. The same does not apply for the Nano Every board. Since I wish the sketch to work with both boards, I will not use the digital pin D11 for PWM in any of the sketches. Let's connect the anode of an LED to the digital pin 6 and the cathode to Ground through a 470 ohm resistor as shown in Figure 4-5.

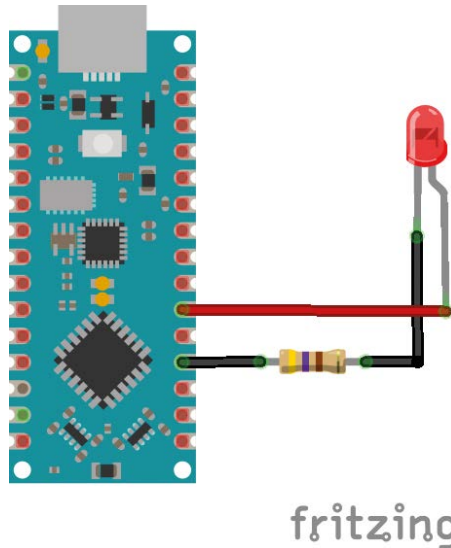


Figure 4-5: LED connected to digital pin D3.

You will have to use the function **analogWrite()** that accepts the PWM pin number and the intensity of the PWM applied as arguments. Intensity ranges from 0 to 255. If the intensity is 0, 63, 127, 255 it means 0%, 25%, 75%, 100% duty cycle, respectively. In addition, you can have an intermediate percentage of values with the following formula,

$$\text{duty cycle percentage} = (\text{intensity} / 255) \times 100$$

Let's write and upload the sketch for the demonstration of the function **analogWrite()**:

```
prog00.ino
int pwm_pin_d3 = 3;
int signal_duration = 2;
void setup()
{
  pinMode(pwm_pin_d3, OUTPUT);
}
void loop()
{
  for(int i=0; i<=255; i++)
```

```

{
  analogWrite(pwm_pin_d3, i);
  delay(signal_duration);
}
for(int i=255; i>=0; i--)
{
  analogWrite(pwm_pin_d3, i);
  delay(signal_duration);
}
}

```

This sketch supplies the intensity value, increasing first and then decreasing. This causes the LED to light bright first and then fade. The change in the intensity of the LED is pre-programmed here. You can manipulate it with an analog input. You have to connect a trimpot of any value to analog pin **A0** to the same circuit, as shown in Figure 4-6.

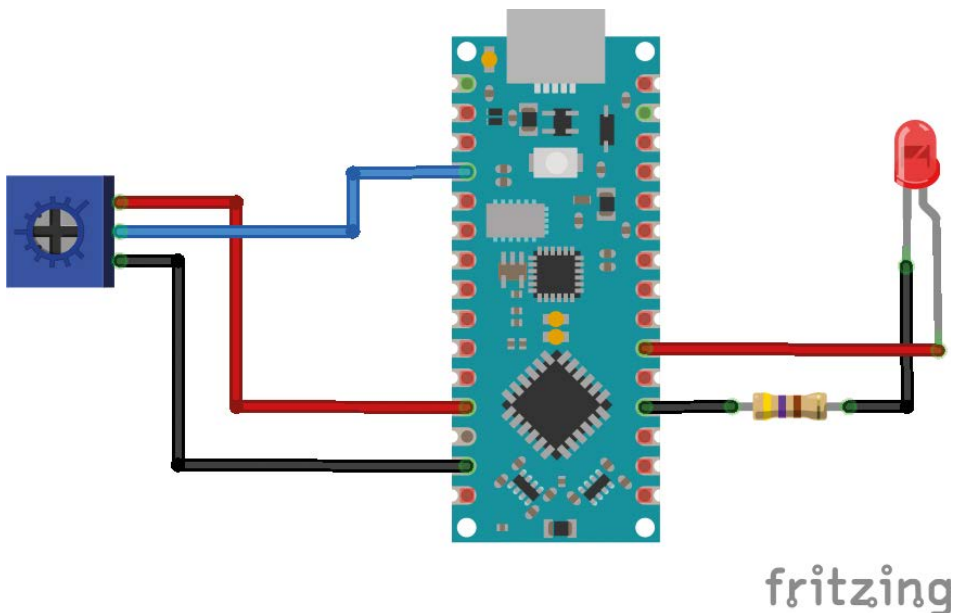


Figure 4-6: Manipulating LED intensity through an analog input.

You have to read the analog input from the analog pin A0 (which is in the range of 0 – 1023, if you remember it from the last chapter) and then map it to the range of 0 – 255. Then pass this new value to the function **analogWrite()** as the second argument, as follows:

```

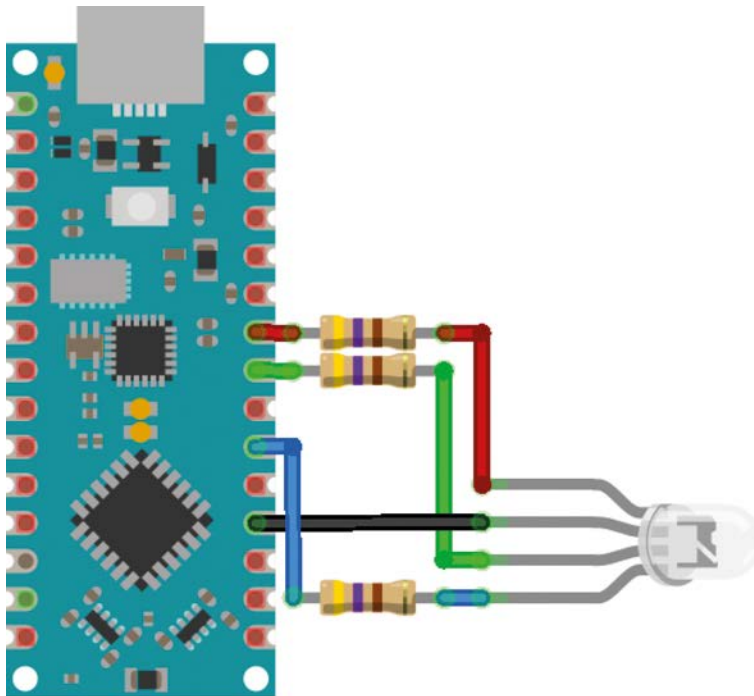
prog01.ino
int pwm_pin_d3 = 3;
int input_signal, intensity;
void setup()
{

```

```
Serial.begin(9600);  
pinMode(pwm_pin_d3, OUTPUT);  
}  
void loop()  
{  
  input_signal = analogRead(0);  
  intensity = map(input_signal, 0, 1023, 0, 255);  
  Serial.println(intensity);  
  analogWrite(pwm_pin_d3, intensity);  
}
```

Now, after uploading the sketch to the board, you can control the intensity using the trim-pot. The variable resistor can be of any value, as the analog input will always convert it to 10-bit resolution (range of 0 through 1023). It is exactly this range that you are mapping to the output intensity value.

You can even connect an RGB LED. Figure 4-7 shows the connections for a common-cathode RGB LED.



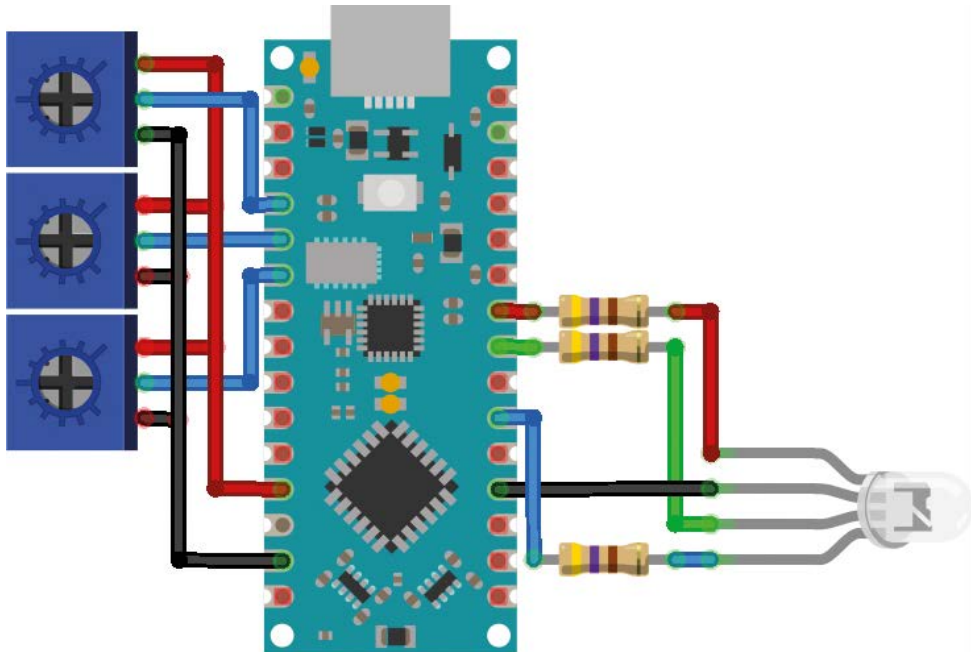
fritzing

Figure 4-7: Common-cathode RGB LED connected to PWM pins D3, D5, and D6.

The following example cycles through a few combinations of colors:

```
prog02.ino
int blue_pin_d3 = 3;
int green_pin_d5 = 5;
int red_pin_d6 = 6;
void setup()
{
  Serial.begin(9600);
  pinMode(blue_pin_d3, OUTPUT);
  pinMode(green_pin_d5, OUTPUT);
  pinMode(red_pin_d6, OUTPUT);
}
void loop()
{
  for ( int i = 0 ; i <= 255; i = i + 32 )
    for ( int j = 0 ; j <= 255; j = j + 32 )
      for ( int k = 0 ; k <= 255; k = k + 32 )
        {
          analogWrite(blue_pin_d3, i);
          analogWrite(green_pin_d5, j);
          analogWrite(red_pin_d6, k);
          Serial.println(i);
          Serial.println(' ');
          Serial.println(j);
          Serial.println(' ');
          Serial.println(k);
        }
}
```

Let's modify the RGB LED circuit and connect three trimpots to the analog pins A0, A1, and A2 as given by the schematic in Figure 4-8.



fritzing

Figure 4-8: Common-cathode RGB controlled by analog inputs.

The code is very simple. You just need to map the input analog values to the range of the output as shown in the following sketch:

```
prog03.ino
int blue_pin_d3 = 3;
int green_pin_d5 = 5;
int red_pin_d6 = 6;
void setup()
{
  Serial.begin(9600);
  pinMode(blue_pin_d3, OUTPUT);
  pinMode(green_pin_d5, OUTPUT);
  pinMode(red_pin_d6, OUTPUT);
}
void loop()
{
  analogWrite(blue_pin_d3, map(analogRead(0), 0, 1023, 0, 255));
  analogWrite(green_pin_d5, map(analogRead(1), 0, 1023, 0, 255));
  analogWrite(red_pin_d6, map(analogRead(2), 0, 1023, 0, 255));
}
```

Upload the sketch and feel free to play with the colors.

You can even use a common anode RGB LED. You just need to make a few minor changes to the circuit as shown in Figure 4-9.

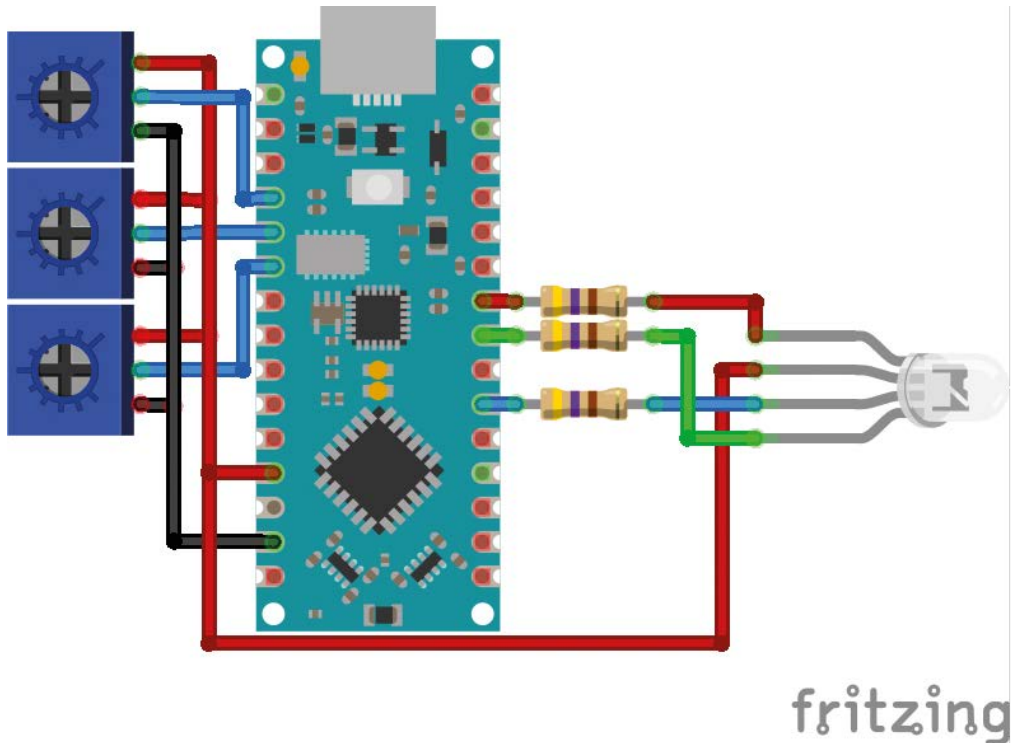


Figure 4-9: Common-anode RGB LED controlled by three analog inputs.

Working with a Servo Motor

You can attach a simple servo motor. For this demonstration, you will attach an SG90 servo motor which is commonly used in robotics-related projects. The motor has 3 pins: VCC (red), GND (black), and Signal (yellow). Attach the VCC to +5V, GND to the ground, and Signal to the PWM pin D3, as shown in Figure 4-10.

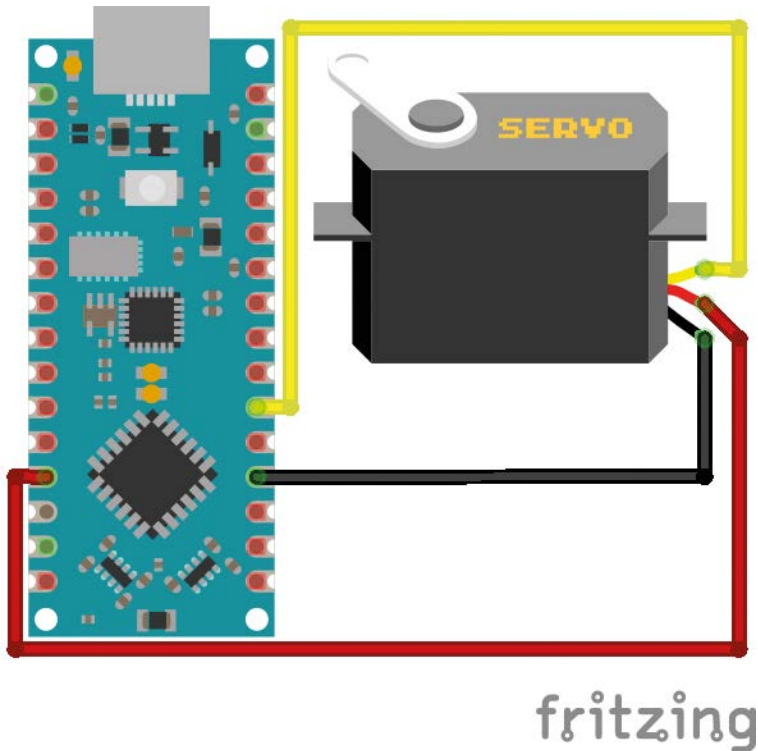


Figure 4-10: Servo motor connected to the PWM pin D3.

There is a built-in library for handling servomotors. Let's create the sketch for this servo using the necessary library:

```
prog04.ino
#include <Servo.h>
Servo servo1;
int servo1_pin = 3, pos = 0;
void setup()
{
  servo1.attach(servo1_pin);
}
void loop()
{
  for (pos = 0; pos <= 180; pos = pos + 6)
  {
    servo1.write(pos);
    delay(1000);
  }
  delay(5000);
}
```


This is a very simple program. You are importing something for the very first time. The first statement imports the required library file to the sketch. It is an integral library and you do not have to install anything. Then you are creating an object for the servo motor. The function **attach()** in the **setup()** associated the PWM pin mentioned in the argument to the object and initializes it. You can send the position angle for the servo using the function **write()**. The argument ranges from 0 to 180. This function sets the angle of the servo to the specified angle. This program successively increases the angle. Upload the sketch to the Arduino Nano to see it in action.

You can improve the project by attaching a trimpot to read the values and set the angle of the servo. Let's try that. The schematic in Figure 4-11 adds a trimpot to the analog input pin A0.

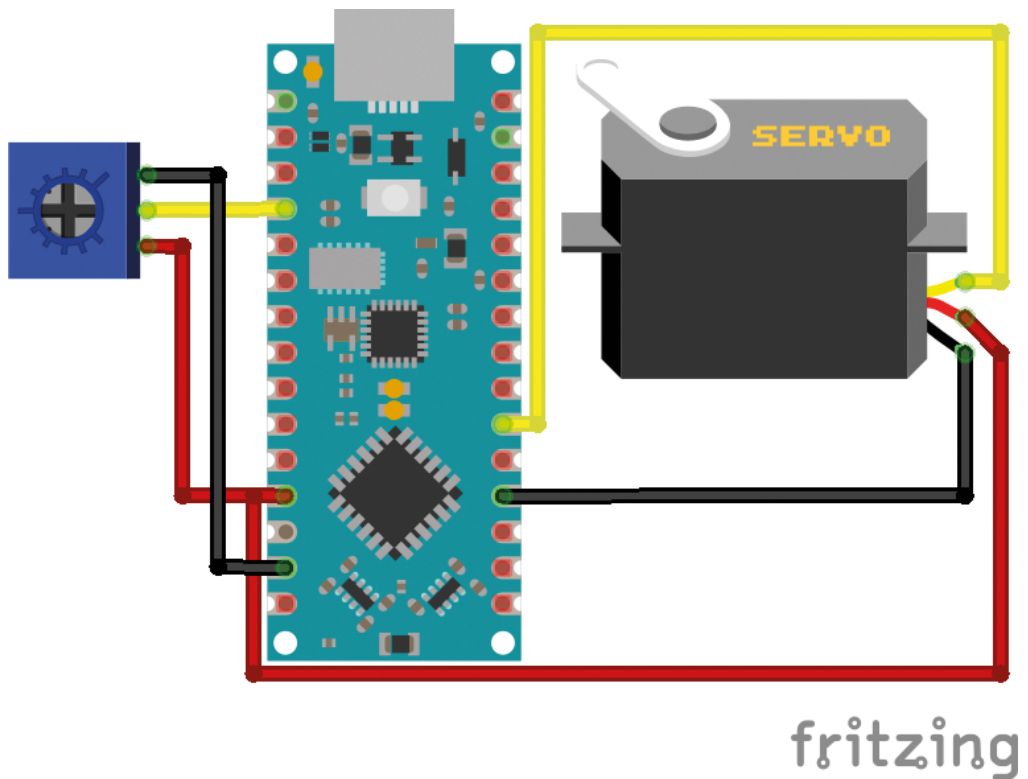


Figure 4-11: Servomotor connected to analog pin A0.

The code for the sketch is very simple. You just have to map the analog input range (0 through 1023) to the range of angle of rotation (0 through 180 degrees).

```
prog05.ino
#include <Servo.h>
Servo servo1;
int servo1_pin = 3;
void setup()
{
  servo1.attach(servo1_pin);
}
void loop()
{
  servo1.write(map(analogRead(0), 0, 1023, 0, 180));
  delay(100);
}
```

As you rotate the dial of the trimpot, you can see the angle of rotation of the servo change in real time.

Working with the 28BYJ-48 Unipolar Stepper Motor and the ULN2003A Motor Driver

This section is really a part of the digital I/O chapter since you will use the digital pin to drive the Unipolar stepper motor. Nevertheless, the topic is pretty advanced for beginners. That is why I decided to include it here, after the discussion on servo motors.

Let's learn the basics of the components we will be using in this section. The very first component is a stepper motor. A stepper motor is also known as a step motor or a stepping motor. It is a brushless electric motor that needs DC current to work. In other words, it uses magnetic fields to move the motor.

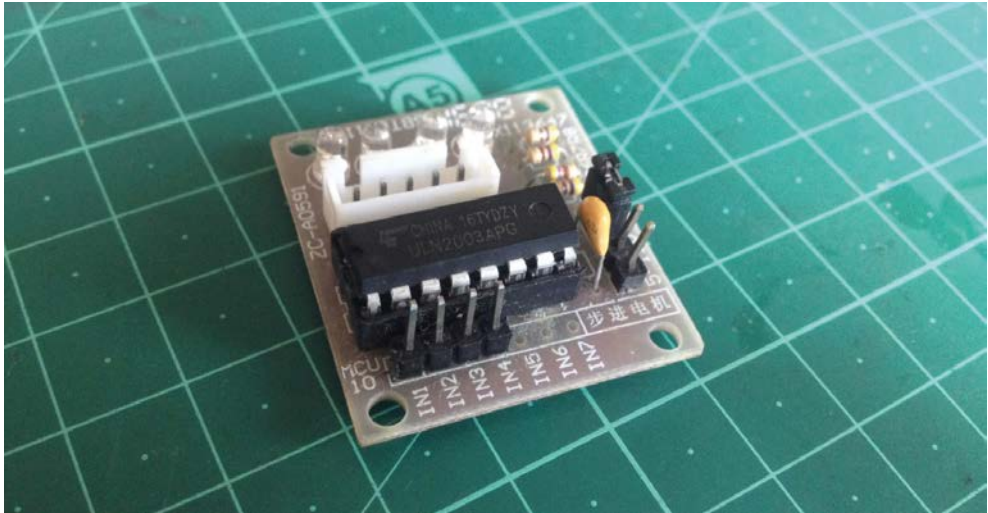
In a stepper motor, a complete rotation of 360 degrees is divided into equal number of steps. Stepper motors use four electromagnets and a cogged wheel for operation. In a unipolar stepper motor, there is one winding per phase. You can turn on each section of a winding to generate a magnetic field in that section. This requires a transistor, or rather an array of transistors, for operations. A unipolar stepper motor usually has five pins. These pins must be connected to a stepper motor controller that can activate the transistors. You will use the 28BYJ-48 unipolar stepper motor for this demonstration. The stepper motor has a Gear Reduction Ratio of 63.68395:1, resulting in approximately 4076 steps per complete revolution in half-step mode.

This prompts a discussion related to the stepper motor controller. You will use a stepper motor controller that uses an IC type ULN2003A, an array of seven NPN Darlington transistors that uses common cathode diodes for implementation. You can read the data sheet of the IC at the following URLs:

<https://www.ti.com/lit/ds/symlink/uln2003a.pdf>
<https://www.ti.com/product/ULN2003A>

You will not directly use the IC as you will have to build a circuit around it. So, you will use a ready-made stepper motor driver board containing a type ULN2003A IC. This chip sends a series of pulses to the motor. The spinning direction of the motor is decided by the sequence of the pulses. The speed of the motor is determined by the frequency of the pulses.

Figure 4-12 shows one such ready-made stepper motor driver.



*Figure 4-12: A ULN2003A-based stepper motor controller board.
(adapted from the image by Kushagra Keshari under the
<https://creativecommons.org/licenses/by-sa/4.0/deed.en> license)*

You can connect the board to a 28BYJ-48 motor as shown in Figure 4-13.

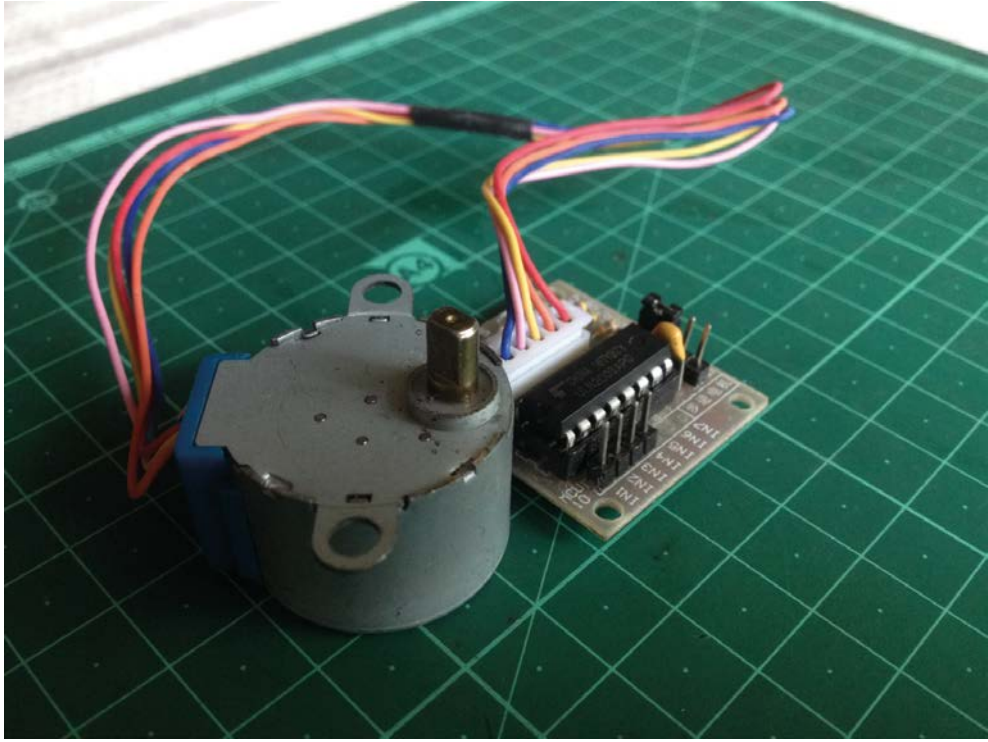


Figure 4-13: A ULN2003A-based stepper motor controller board connected to a 28BYJ-48 motor. (adapted from the image by Kushagra Keshari under the <https://creativecommons.org/licenses/by-sa/4.0/deed.en> license)

The pins of the motor come with a male-type adapter that fits perfectly on the female port of the stepper motor controller board. You cannot get it wrong; these devices are truly manufactured for each other. Let's connect this to an Arduino Nano board as shown in Figure 4-14.

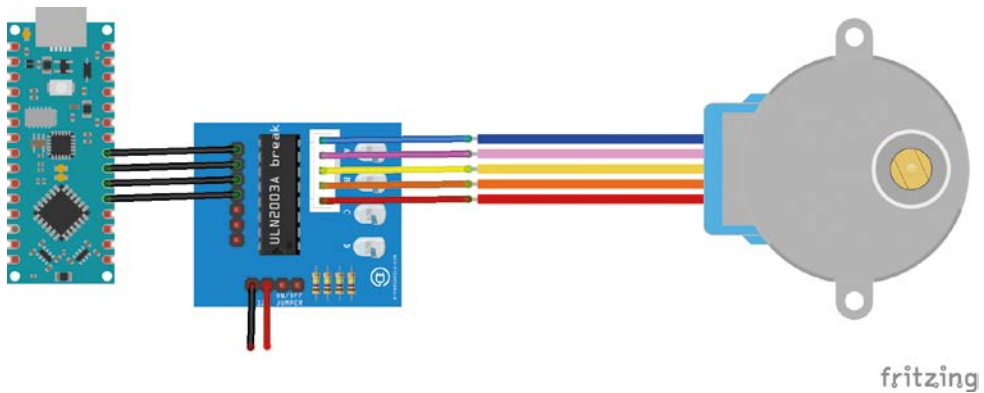


Figure 4-14: A ULN2003A and 28BYJ-48 connected to a Nano board (for anticlockwise rotations).

Note: You can download the Fritzing parts for the motor and the driver board from the following URLs:

<https://github.com/e-radionicom/e-radionica.com-Fritzing-Library-parts-/blob/master/28BYJ-48%20Stepper%20Motor.fzpz>

<https://github.com/e-radionicom/e-radionica.com-Fritzing-Library-parts-/blob/master/ULN2003A%20breakout%20board.fzpz>

The connection between the stepper motor and the driver board is straightforward as the motor has a male plug, and the board has a female port. Let's discuss the connections between the motor driver board and Nano board. Power the motor driver board separately as it may generate noise and damage the Nano board if connected to it. The best option is to use an MB 102-style breadboard power supply with it.

Near the ULN2003A IC, you will see seven pins (IN1 through IN7). The last three (IN5, IN6, IN7) are disabled in most of the motor driver boards as the stepper motor you are using only needs the first four pins IN1, IN2, IN3, and IN4. For counterclockwise (anticlockwise) rotation of the motor shaft (corresponding to Figure 4-14), the following table applies.

Arduino Pin	Motor Driver Board Pin
D2	IN1
D3	IN2
D4	IN3
D5	IN4

For clockwise rotation, the connections are as pictured in Figure 4-15.

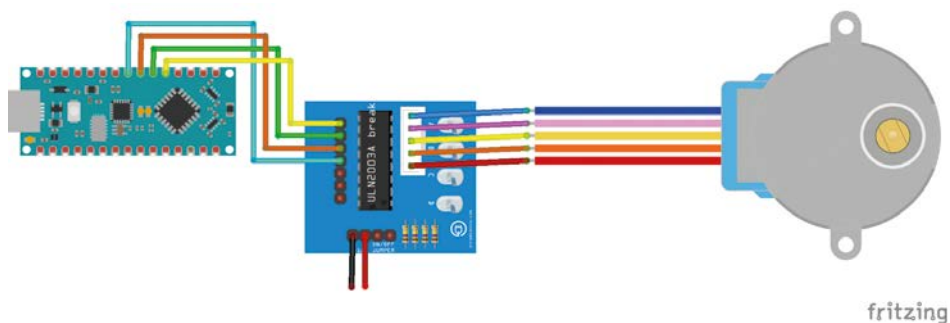


Figure 4-15: A ULN2003A and 28BYJ-48 connected to a Nano board (for clockwise rotations).

The table for the connections applicable to Figure 4-15 is as follows:

Arduino Pin	Motor Driver Board Pin
D2	IN4
D3	IN3
D4	IN2
D5	IN1

Now, let's write a program for this circuit.

```
prog06.ino
int A=2, B=3, C=4, D=5;
int steps_per_rotation=512;
int duration=5;
int sequence[8][4]={{1,0,0,0},
                    {1,1,0,0},
                    {0,1,0,0},
                    {0,1,1,0},
                    {0,0,1,0},
                    {0,0,1,1},
                    {0,0,0,1},
                    {1,0,0,1}};

void setup()
{
  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(C, OUTPUT);
  pinMode(D, OUTPUT);
}

void send_sequence(int a, int b,
                  int c, int d)
{
  digitalWrite(A, a);
  digitalWrite(B, b);
  digitalWrite(C, c);
  digitalWrite(D, d);
}

void single_step(){
  for(int i=0; i<8; i++)
  {
    send_sequence(sequence[i][0],
                  sequence[i][1],
                  sequence[i][2],
```

```

        sequence[i][3]);
    delay(duration);
}
}

void loop()
{
    int i=0;
    while(i<steps_per_rotation)
    {
        single_step();
        i++;
    }
}

```

This is a relatively large program. Let's understand it block by block. In the first line, you are defining the digital pins of the Nano board to be used. By reversing this line, you can change the direction of the motor. Next, you are defining steps per full rotation (revolution) and the delay between the microsteps. There are four coils in the motor. You are supplying the positive voltage (1 or HIGH) to one or two coils in sequence from A to D. This will make the motor shaft revolve in a particular direction (depending on your wiring). So, you created a 2D array (or matrix) of the sequences to be applied to each coil. In the **setup()**, you are initializing the pre-defined pins as the digital output pins. The custom function **send_sequence()** sends the transferred values to the respective pins. You can reverse the sequence to change the direction of rotation here too. The custom function **single_step()** sends the values in the **sequence** matrix row-wise to the motor driver. This calls **send_sequence()** once per row in the **sequence** matrix. You are calling this function for every step in the rotation in the **loop()** section. You can also change the direction of the rotation by mirroring the signal **sequence** matrix along with the Y-axis (mirroring column-wise). This is known as microstepping technique.

You can make the motor rotate faster by changing the sequence matrix as follows:

```

int sequence[4][4]={{1,1,0,0},
                   {0,1,1,0},
                   {0,0,1,1},
                   {1,0,0,1}};

```

And the function **single_step()** needs to be modified as follows:

```

void single_step(){
    for(int i=0; i<4; i++)
    {
        send_sequence(sequence[i][0],
                      sequence[i][1],
                      sequence[i][2],

```

```
        sequence[i][3]);  
    delay(duration);  
}  
}
```

This will increase the speed of the motor. I have added the modified program to the code bundle in the book. You can find it in the relevant directory for this chapter. The file is named **prog07.ino**.

Using a Custom Library for Stepper Motors

You wrote the code manually to drive the motor with the controller. Now, you will use a custom library to drive the motor. The connections will remain the same as in the earlier example (Figures 4-14 and 4-15). The library is called **AccelStepper**, and you can download it from the **Library Manager** from the **Sketch** Menu as shown in Figure 4-16.

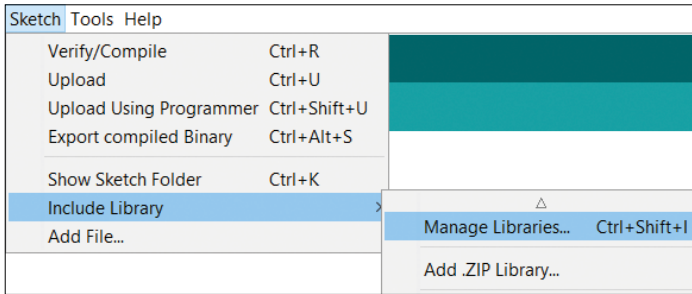


Figure 4-16: Library Manager under the Sketch menu.

Afterwards, search the **AccelStepper** library and install it (Figure 4-17).

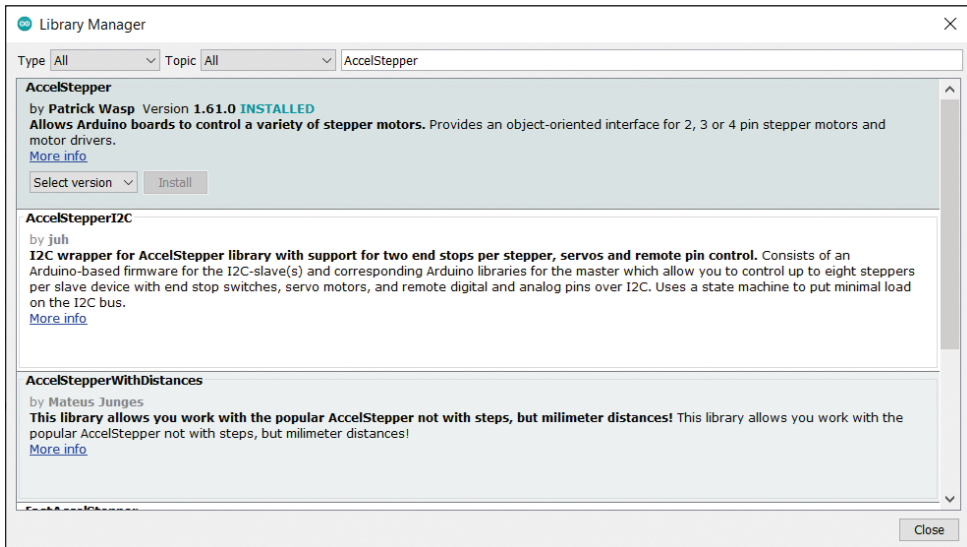


Figure 4-17: Installing a custom library (AccelStepper) for the stepper motor.

Once installed, upload the following sketch to it:

```
prog08.ino
#include<AccelStepper.h>
const int FULLSTEP=4;
AccelStepper stepper1(FULLSTEP, 2, 4, 3, 5);
void setup()
{
  stepper1.setMaxSpeed(1000.0);
  stepper1.moveTo(2038);
}
void loop()
{
  stepper1.setSpeed(500);
  stepper1.runSpeed();
}
```

Let's understand the meaning of the sketch. In the very first line, you are importing the installed third-party library. In the second line, you are deciding what step to use. There are three types of step. The first one is the microstep, which you have already implemented by manually sending the signal matrix row-by-row in the earlier example. The other two are **full step** and **half step** modes. You are going to use full-step mode in this example. Then you created the object by passing the stepping mode and the pin numbers. Please pass the pin numbers to the constructor exactly the same way it is printed in the sketch (it is not a typo or printing error, but the legitimate sequence of pin numbers). In the **setup()**, you are setting the maximum number of steps per second (maximum speed) and moving the stepper motor to the starting position. Finally, in the **loop()**, you are stating the speed (steps per second) and then stepping the motor at the end. Upload the sketch and see the motor in action.

You can modify the program to make the motor rotate in the other direction with **half step** by modifying the following two lines.

```
const int HALFSTEP=8;
AccelStepper stepper1(HALFSTEP, 5, 3, 4, 2);
```

I have included this code in a separate file in the code bundle, under this chapter's directory. The file is called **prog09.ino**.

You can rotate the motor in both directions as follows:

```
prog10.ino
#include<AccelStepper.h>
const int HALFSTEP=8;
AccelStepper stepper1(HALFSTEP, 5, 3, 4, 2);
void setup()
```

```
{
  stepper1.setMaxSpeed(1000.0);
  stepper1.moveTo(2038);
}
void loop()
{
  stepper1.setCurrentPosition(0);
  while (stepper1.currentPosition() != 2048)
  {
    stepper1.setSpeed(1000);
    stepper1.runSpeed();
  }
  delay(1000);
  stepper1.setCurrentPosition(0);
  while (stepper1.currentPosition() != -2048)
  {
    stepper1.setSpeed(-1000);
    stepper1.runSpeed();
  }
  delay(1000);
}
```

There are a couple of new functions in this example. Function **setCurrentPosition()** sets the position of the shaft. Function **currentPosition()** returns the current position of the shaft. This example makes the shaft rotate in both directions alternately because you are changing the direction of the motor by passing a positive and a negative speed to function **setSpeed()**.

You can rewrite the entire code with a few new functions too:

```
prog11.ino
#include<AccelStepper.h>
const int HALFSTEP=8;
AccelStepper stepper1(HALFSTEP, 5, 3, 4, 2);
void setup()
{
  stepper1.setMaxSpeed(1000.0);
  stepper1.setSpeed(100);
  stepper1.setAcceleration(75.0);
  stepper1.setCurrentPosition(0);
  stepper1.moveTo(2038);
}
void loop()
{
  if (stepper1.distanceToGo() == 0)
    stepper1.moveTo(-stepper1.currentPosition());
```

```

stepper1.run();
}

```

You are using a few new functions in this example. Function **setAcceleration()** sets the acceleration (change in speed over time). You can witness that the speed of the motor is changing by observing the LEDs on the motor driver board. The rate at which they blink will change once the speed changes. The function **distanceToGo()** computes the distance from the target position.

Working with Multiple Stepper Motors

You can attach multiple motors as shown in Figure 4-18.

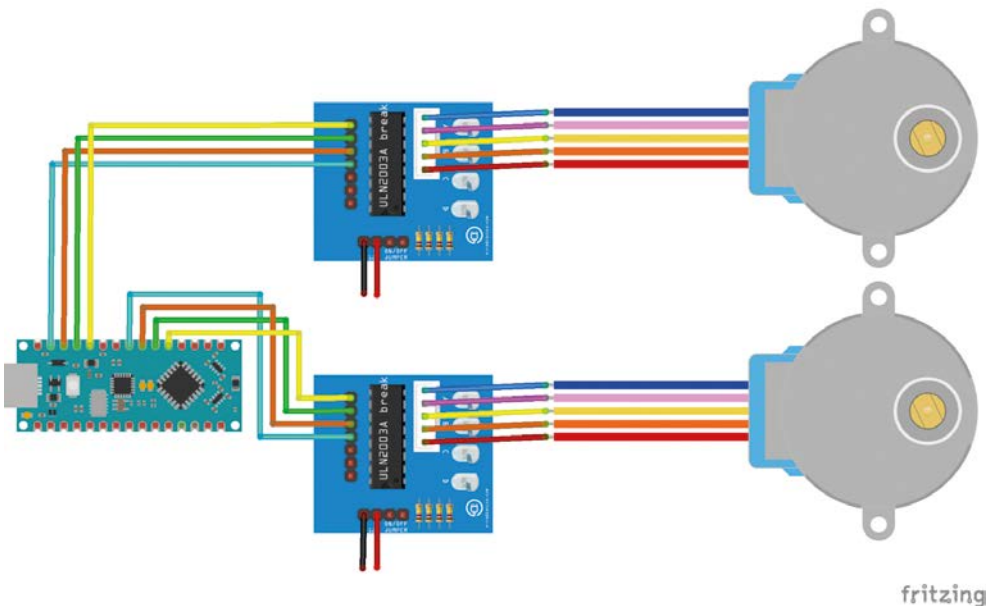


Figure 4-18: Attaching multiple motors.

Here, you need to use two different sets of the digital I/O pins for both stepper motor controllers. Here is the code:

```

prog12.ino
#include<AccelStepper.h>
const int HALFSTEP=8, FULLSTEP=4;
AccelStepper stepper1(HALFSTEP, 5, 3, 4, 2);
AccelStepper stepper2(FULLSTEP, 8, 10, 9, 11);
void setup()
{
  stepper1.setMaxSpeed(1000.0);
  stepper1.setSpeed(100);
  stepper1.setAcceleration(75.0);
}

```

```
stepper1.setCurrentPosition(0);
stepper1.moveTo(2038);

stepper2.setMaxSpeed(1000.0);
stepper2.setSpeed(150);
stepper2.setAcceleration(50.0);
stepper2.setCurrentPosition(0);
stepper2.moveTo(-2038);
}
void loop()
{
  if (stepper1.distanceToGo() == 0)
    stepper1.moveTo(-stepper1.currentPosition());
  if (stepper2.distanceToGo() == 0)
    stepper2.moveTo(-stepper2.currentPosition());
  stepper1.run();
  stepper2.run();
}
```

Since you are already familiar with all the functions, I won't explain this example. Just upload the code to the Nano board and see it in action.

Summary

In this chapter, you learned the fundamentals of PWM (Pulse Width Modulation). You used it with LEDs, RGB LEDs, and SG90 servo motors. You also worked with a unipolar stepper motor and an associated motor controller.

In the next chapter, you will explore in detail how to use various types of ready-made displays with the Arduino.

Chapter 5 • Plotting Geometric Art on an External Display

In the previous chapter, you learned to labor with PWM (Pulse Width Modulation). You used PWM to fade a normal and an RGB LED. Furthermore, you used it to control an SG90 servo motor. At this point, you should be reasonably comfortable with the concept and applications of PWM.

In this chapter, you will be introduced to an external display module based on Ilitek 9225 driver IC. You will also learn to create art — or "artwork" — with the external display.

The Ilitek 9225 Driver IC and the Display

The device designated "ILI9225" is a display driver IC created by Ilitek Taiwan. You can connect it to any TFT LCD board with a maximum resolution of 176×220 (pixels). It supports 16 bits per pixel RGB colors, i.e., 262144 different colors. The chip has 87120 bytes of RAM for graphics data and is ready for communication through the Nano's SPI interface.

You can purchase a **2.2-inch 166x220 TFT LCD with on-board 9225 driver** as a single package. Many times, the device comes with an embedded microSD card reader which has separate pins. You can check online marketplaces or hobby electronics stores for this display. The datasheet for the driver IC can be found at:

<https://www.displayfuture.com/Display/datasheet/controller/ILI9225.pdf>

Let's see how to connect this interesting unit with the Nano board. The following table explains the display module pins mapped with the Nano board pins.

Display Module Pin	Arduino Board Pin
VCC	+5 V
GND	GND
GND	No connection
NC	No connection
NC	No connection
NC	No connection
SCK	A0
SDA	A1
RS	A2
RST	A3
CS	A4

You will have to install a library for this module to make it work with your Arduino. Open the **Library Manager** from the **Sketch** Menu and search for: **9225**. Figure 5-1 is the screenshot of the menu opened up.

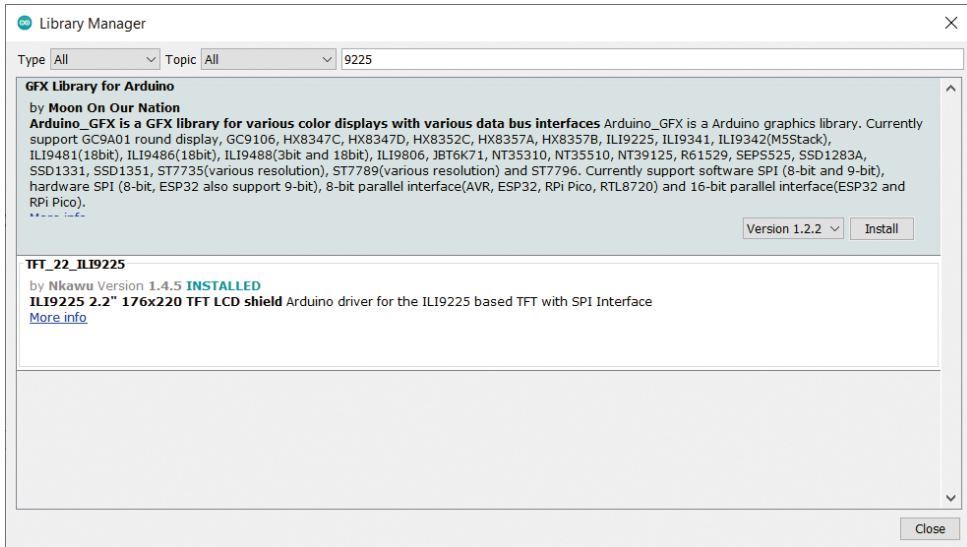


Figure 5-1: TFT_22_ILI9225 library installation.

Install the **TFT_22_ILI9225** library by **Nkawu**. You can read the documentation of the library at:

https://github.com/Nkawu/TFT_22_ILI9225

and on the Wikipage at:

https://github.com/Nkawu/TFT_22_ILI9225/wiki.

Programming the Display

The 2.2-inch TFT LCD 9225 display is one of the most versatile displays available to display static graphics and show simple animations. Let's find out how to program it with our Nano. You will learn to write the code block by block, and I will explain each line. Create a new directory (folder) for the sketches in this chapter and start writing the following code into a new sketch named as **prog00.ino**.

You are going to use the **SPI** (Serial Peripheral Interface) protocol to work with the display. You will also need to use the recently installed ILI9225 library. So, let's import both libraries:

```
#include "SPI.h"
#include "TFT_22_ILI9225.h"
```

Let's use preprocessor directives to define a few macros for pin assignments, as follows:

```
#define TFT_CS  A4
#define TFT_RST A3
#define TFT_RS  A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
```

Note that, across the manufacturers, the pins may have different names. Let's create an object corresponding to the TFT display:

```
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
```

Then, define the setup section:

```
void setup()
{
```

Let's initialize the TFT display:

```
tft.begin();
```

and turn on the backlight and display:

```
tft.setBacklight(true);
tft.setDisplay(true);
```

Why not set the background color...

```
tft.setBackgroundColor(COLOR_BLACK);
```

.. and the orientation

```
tft.setOrientation(0);
```

.. and then initialize the serial comms:

```
Serial.begin(9600);
```

The following code block is used to detect the orientation (we already set it):

```
Serial.print("Orientation: ");
switch (tft.getOrientation())
{
case 0:
Serial.print("Portrait");
```

```
    break;
  case 1:
    Serial.print("Right Rotated Landscape");
    break;
  case 2:
    Serial.print("Reverse Portrait");
    break;
  case 3:
    Serial.print("Left Rotated Landscape");
    break;
  default:
    Serial.print("Invalid Orientation...");
  }
}
```

Let's retrieve the number of rows and the number of columns:

```
Serial.print("\nNumber of rows: ");
Serial.println(tft.maxX());
Serial.print("Number of columns: ");
Serial.println(tft.maxY());
}
```

and then define the loop() section:

```
void loop()
{
```

You can clear the display with the following statement:

```
tft.clear();
```

Note that this is very computationally "heavy" and it takes a few seconds to clear the display. So, if you are planning a simple animation, you cannot afford to call this every time to clear the display as it won't be fast enough.

Let's set the font:

```
tft.setFont(Terminal6x8);
```

You can draw text by providing the coordinates, the text, and the color, like so:

```
tft.drawText(10, 10, "Gradient Demo", COLOR_WHITE);
delay(1000);
```


You can also draw lines by providing the coordinates of the endpoints. Also, the routine `setcolor()` accepts three 8-bit unsigned integer arguments R, G, and B (ranging from 0 to 255) and converts them into a single 16-bit number corresponding to an RGB color:

```
for( int i = 0; i < tft.maxY(); i++ )
{
  tft.drawLine(0, i, tft.maxX()/3, i, tft.setColor(i, 0, 0));
  tft.drawLine((tft.maxX()/3)+1, i, (2* (tft.maxX()/3)), i, tft.setColor(0, i, 0));
  tft.drawLine((2* (tft.maxX()/3))+1, i, tft.maxX(), i, tft.setColor(0, 0, i));
}
delay(1000);
```

Here, we are creating a gradient image of red, green, and blue colors. Now, consider drawing a hollow rectangle and a solid rectangle by providing the opposite vertices and the color as arguments, like this:

```
tft.clear();
tft.drawText(10, 10, "Rectangle");
tft.drawRect(20, 20, 100, 100, COLOR_BLUE);
delay(1000);
tft.drawText(10, 10, "Rectangle (Solid)");
tft.fillRect(20, 20, 100, 100, COLOR_RED);
delay(1000);
```

Now, draw a hollow circle and a solid circle by providing the coordinates of the center, radius, and color:

```
tft.clear();
tft.drawText(10, 10, "Circle");
tft.drawCircle(tft.maxX()/2, tft.maxY()/2, 50, COLOR_YELLOW);
delay(1000);
tft.drawText(10, 10, "Circle (Solid)");
tft.fillCircle(tft.maxX()/2, tft.maxY()/2, 50, COLOR_GREEN);
delay(1000);
}
```

The routines that commence with the string **draw** are for drawing hollow figures, and those commencing with the string **fill** are for solid figures. Putting it all together, we obtain the following sketch:

```
prog00.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS  A4
#define TFT_RST A3
#define TFT_RS  A2
```

```
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
  Serial.print("Orientation: ");
  switch (tft.getOrientation())
  {
    case 0:
      Serial.print("Portrait");
      break;

    case 1:
      Serial.print("Right Rotated Landscape");
      break;
    case 2:
      Serial.print("Reverse Portrait");
      break;
    case 3:
      Serial.print("Left Rotated Landscape");
      break;
    default:
      Serial.print("Invalid Orientation...");
  }
  Serial.print("\nNumber of rows: ");
  Serial.println(tft.maxX());
  Serial.print("Number of columns: ");
  Serial.println(tft.maxY());
}
void loop()
{
  tft.clear();
  tft.setFont(Terminal6x8);
  tft.drawText(10, 10, "Gradient Demo", COLOR_WHITE);
  delay(1000);
  for( int i = 0; i < tft.maxY(); i++ )
  {
    tft.drawLine(0, i, tft.maxX()/3, i, tft.setColor(i, 0, 0));
```

```
tft.drawLine((tft.maxX()/3)+1, i, (2* (tft.maxX()/3)), i, tft.setColor(0, i,
0));
tft.drawLine((2* (tft.maxX()/3))+1, i, tft.maxX(), i, tft.setColor(0, 0, i));
}
delay(10000);
tft.clear();
tft.drawText(10, 10, "Rectangle");
tft.drawRect(20, 20, 100, 100, COLOR_BLUE);
delay(10000);
tft.drawText(10, 10, "Rectangle (Solid)");
tft.fillRect(20, 20, 100, 100, COLOR_RED);
delay(10000);
tft.clear();
tft.drawText(10, 10, "Circle");
tft.drawCircle(tft.maxX()/2, tft.maxY()/2, 50, COLOR_YELLOW);
delay(10000);
tft.drawText(10, 10, "Circle (Solid)");
tft.fillCircle(tft.maxX()/2, tft.maxY()/2, 50, COLOR_GREEN);
delay(10000);
}
```

I have adjusted the delay in the above program and all the subsequent programs. It allows me to capture photographs of the output on the TFT LCD using my digital camera. Let's see the output screens step by step. The string "Gradient Demo" colored white against the black background is printed first as in Figure 5-1.



Figure 5-2: Printing a string.

I have used a Lumix GH5 in manual photography mode to capture the output. Since I am not a great photographer, the photographs may not be of the best quality. Moreover, a consumer-grade digital camera cannot truthfully capture the output produced on a digital display. Fortunately, its output looks much better in reality.

After the text, the system displays a gradient image created using individual lines, like in Figure 5-3.

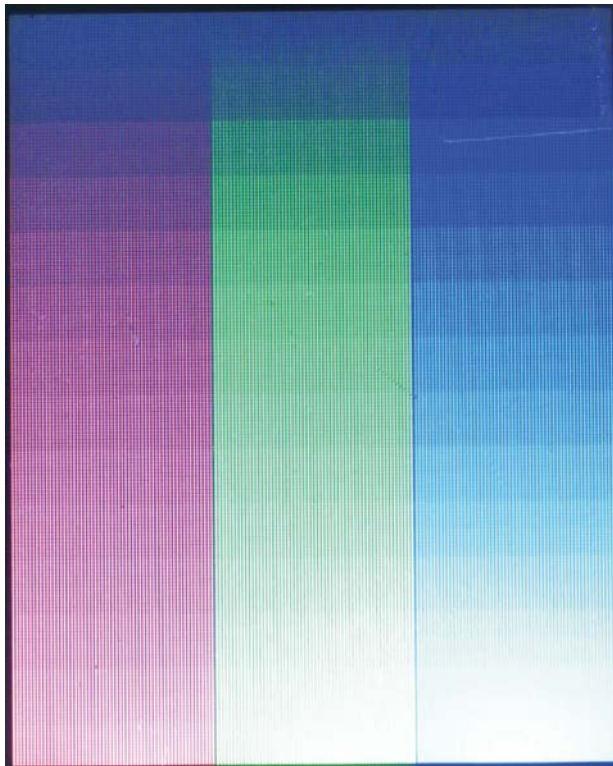


Figure 5-3: Color gradient.

It shows a hollow rectangle, as in Figure 5-3.



Figure 5-4: Hollow rectangle.

Then, it displays a solid/filled rectangle (Figure 5-5).

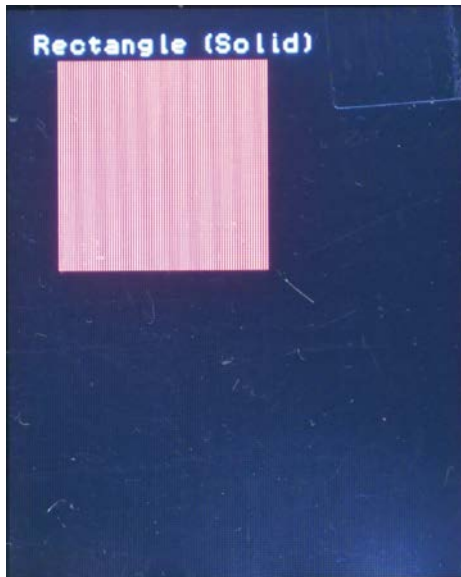


Figure 5-5: Solid (filled) rectangle.

Next up is a hollow circle; Figure 5-6.



Figure 5-6: Hollow circle.

And the final image in the sequence is a solid (filled) circle or ball; see Figure 5-7.

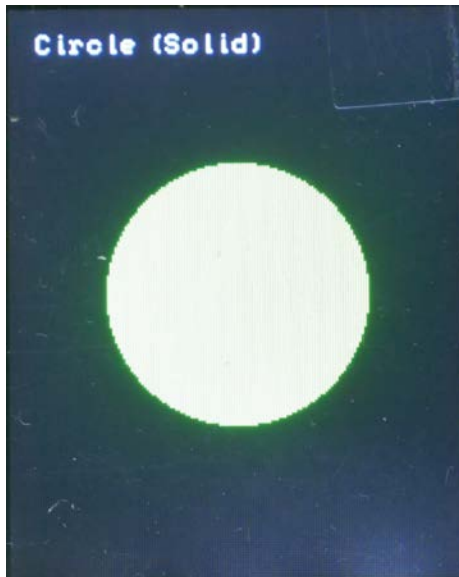


Figure 5-7: Solid (filled) circle.

Pay close attention to how the circle is filled with the assigned color. Since you have written this sequence inside the **loop()** section, it keeps on repeating as long as the Arduino board is powered on. We are following the similar pattern for most of the examples in the remainder of the chapter.

Let's check out another example. You will create circles of gradients of the colors red, green, and blue, respectively. It will be followed by dots of random color at random positions. Here is the sketch.

```

prog01.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS  A4
#define TFT_RST A3
#define TFT_RS  A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void loop()
{
  tft.clear();
  tft.setFont(Terminal6x8);
  tft.drawText(10, 10, "Animation Demo!");
  delay(1000);
  for ( int i=0; i<(tft.maxX()/2); i++)
  {
    tft.drawCircle(tft.maxX()/2, tft.maxY()/2, i, random(0xffff));
    delay(10);
  }
  tft.clear();
  for ( int i=0; i<(tft.maxX()/2); i++)
  {
    tft.drawCircle(tft.maxX()/2, tft.maxY()/2, i, tft.setColor(i, 0, 0));
    delay(10);
  }
  tft.clear();
  for ( int i=0; i<(tft.maxX()/2); i++)
  {
    tft.drawCircle(tft.maxX()/2, tft.maxY()/2, i, tft.setColor(0, i, 0));
    delay(10);
  }
}

```

```
}
tft.clear();
for ( int i=0; i<(tft.maxX()/2); i++)
{
  tft.drawCircle(tft.maxX()/2, tft.maxY()/2, i, tft.setColor(0, 0, i));
  delay(10);
}
tft.clear();
tft.drawText(10, 10, "Random Demo!");
for (int i = 0; i < 6000; i++)
{
  tft.drawPixel(random(tft.maxX()), random(tft.maxY()), random(0xffff));
  delay(1);
}
delay(1000);
}
```

Figure 5-8 is a photograph of the gradient circle of the green color.

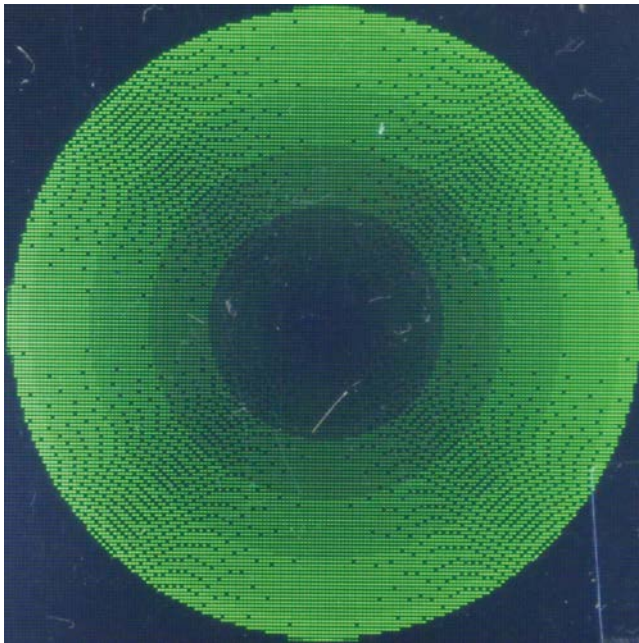


Figure 5-8: Gradient circle.

Note that you are using a routine called **drawPixel()** in this sketch. It accepts the coordinates and the color values for a pixel and draws it on the display. The pixels of random color and random position appear as in Figure 5-9



Figure 5-9: Random pixels.

We know that the resolution of the screen is 176×220 . Both numbers are divisible by 11 and 22. So, let's divide the display into a grid where individual square is of size 22×22 and use it to show repetitive patterns. As a matter of fact, I got this idea of printing repetitive geometric patterns from the floor and wall tiles of my home. Here comes the complete sketch.

```
prog02.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
```

```
Serial.begin(9600);
}
void loop()
{
  tft.clear();
  for ( int i = 0; i < 176; i = i+22 )
  {
    for (int j = 0; j < 220; j = j+22 )
    {
      tft.fillTriangle(i, j+21, i+21, j+21, i+21, j, random(0xffff));
      Serial.println(i);
    }
  }
  delay(5000);
}
```

You are using the routine **fillTriangle()** to draw a solid triangle. It accepts the coordinates of the three vertices of the triangle and the color as arguments. The output should look like Figure 5-10.



Figure 5-10: Repetitive triangle pattern.

You can also write the sketch in such a way that the triangles are oriented in a random fashion in the cell. The sketch is as follows.

```
prog03.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void loop()
{
  tft.clear();
  for ( uint8_t i = 0; i < 176; i = i+22 )
  {
    for ( uint8_t j = 0; j < 220; j = j+22 )
    {
      uint8_t x1 = i,    y1 = j;
      uint8_t x2 = i+21, y2 = j;
      uint8_t x3 = i,    y3 = j+21;
      uint8_t x4 = i+21, y4 = j+21;
      uint8_t choice = random(4);
      if (choice == 0)
        tft.drawTriangle(x1, y1, x2, y2, x3, y3, COLOR_WHITE);
      if (choice == 1)
        tft.drawTriangle(x1, y1, x2, y2, x4, y4, COLOR_WHITE);
      if (choice == 2)
        tft.drawTriangle(x1, y1, x4, y4, x3, y3, COLOR_WHITE);
      if (choice == 3)
        tft.drawTriangle(x4, y4, x2, y2, x3, y3, COLOR_WHITE);
    }
  }
  delay(5000);
}
```

The output from the sketch is pictured in Figure 5-11.

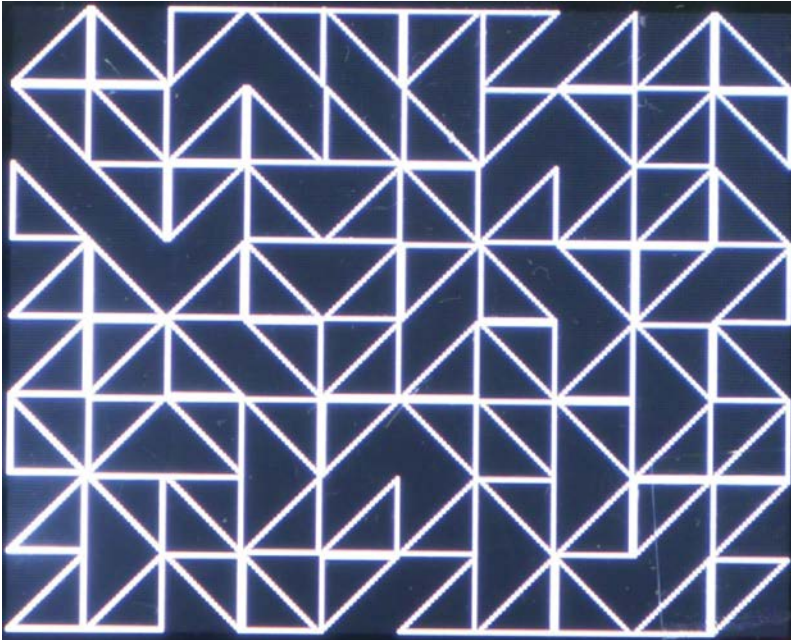


Figure 5-11: Random triangle pattern.

Since the pattern is random, there can be many combinations. It will produce a different output every time the sketch is executed.

You can even divide the display in cells of size 11×11 and draw triangles. This time let's randomize the colors as well. Use this sketch:

```
prog04.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
}
```

```
Serial.begin(9600);
}
void loop()
{
  tft.clear();
  for ( uint8_t I = 0; i < 176; i = i+11 )
  {
    for ( uint8_t j = 0; j < 220; j = j+11 )
    {
      uint8_t x1 = i,    y1 = j;
      uint8_t x2 = i+10, y2 = j;
      uint8_t x3 = i,    y3 = j+10;
      uint8_t x4 = i+10, y4 = j+10;
      uint8_t choice = random(4);
      if (choice == 0)
        tft.drawTriangle(x1, y1, x2, y2, x3, y3, random(0xffff));
      if (choice == 1)
        tft.drawTriangle(x1, y1, x2, y2, x4, y4, random(0xffff));
      if (choice == 2)
        tft.drawTriangle(x1, y1, x4, y4, x3, y3, random(0xffff));
      if (choice == 3)
        tft.drawTriangle(x4, y4, x2, y2, x3, y3, random(0xffff));
    }
  }
  delay(5000);
}
```

The output from the sketch should appear as in Figure 5-12.

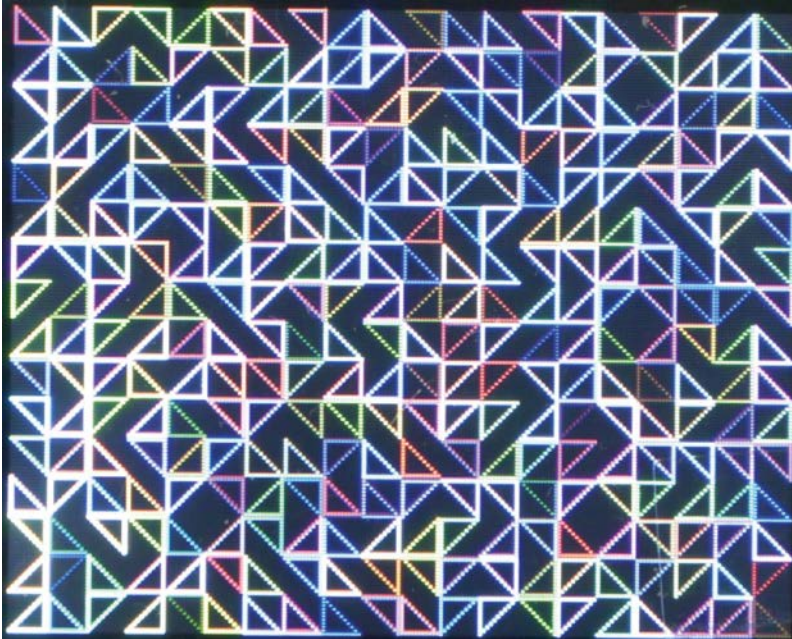


Figure 5-12: Random triangle pattern with equally random colors.

You can also draw random-colored squares as follows:

```
prog05.ino
#include""SPI.""
#include""TFT_22_ILI9225.""
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void loop()
{
```



```
tft.clear();
for ( uint8_t i = 0; i < 176; i = i+11 )
{
  for ( uint8_t j = 0; j < 220; j = j+11 )
  {
    uint8_t x1 = i,    y1 = j;
    uint8_t x2 = i+10, y2 = j;
    uint8_t x3 = i,    y3 = j+10;
    uint8_t x4 = i+10, y4 = j+10;
    tft.drawRectangle(x1, y1, x4, y4, random(0xffff));
  }
}
delay(5000);
}
```

You should be able to see the output as in Figure 5-13.

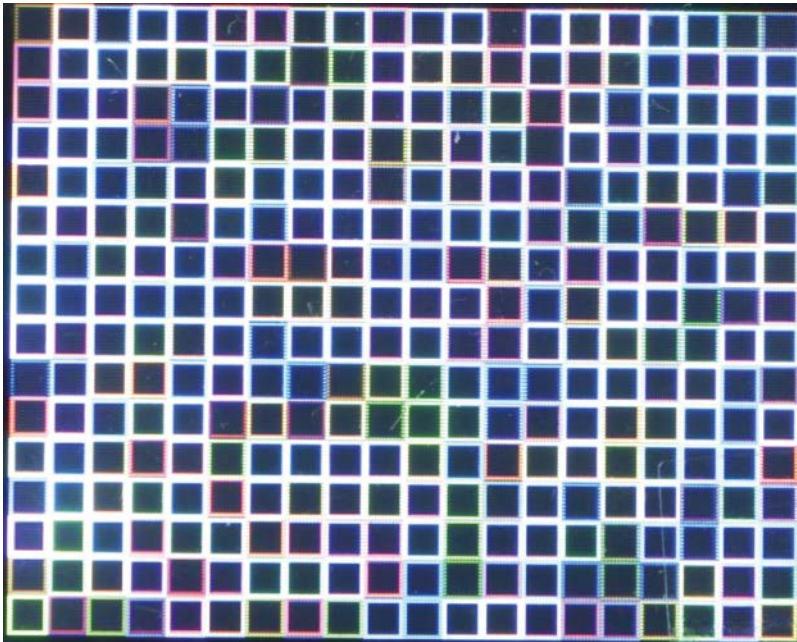


Figure 5-13: Random-colored squares pattern.

You can use **fillRectangle()** to draw random solid squares. You can find that in **prog06.ino** of the code bundle of this chapter (5). That sketch produces output as shown in Figure 5-14:

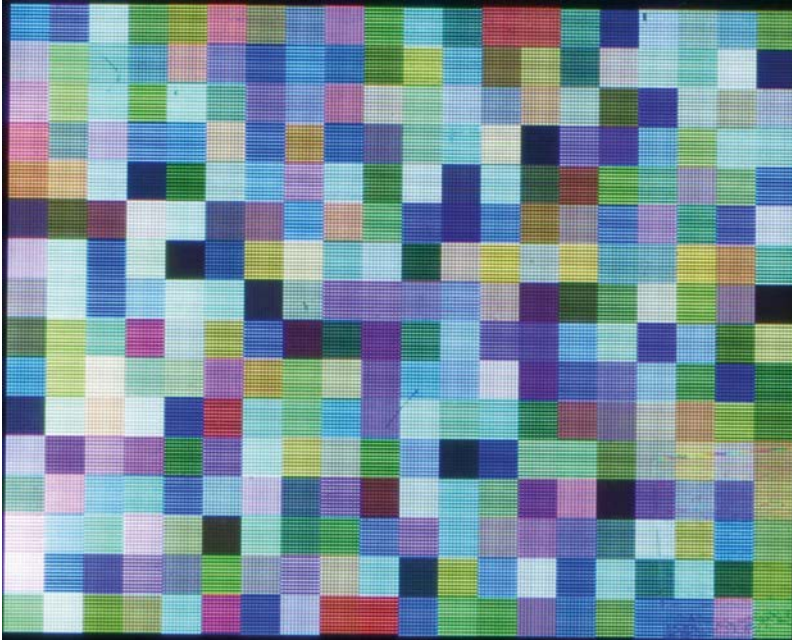


Figure 5-14: Random solid rectangle pattern.

You can even draw a pair of random-colored lines joining the opposite vertices of every cell to create a nice pattern. Check the following sketch:

```
prog07.ino
#include""SPI.""
#include""TFT_22_ILI9225.""
#define TFT_CS  A4
#define TFT_RST A3
#define TFT_RS  A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void loop()
```



```
{
  tft.clear();
  for ( uint8_t i = 0; i < 176; i = i+11 )
  {
    for ( uint8_t j = 0; j < 220; j = j+11 )
    {
      uint8_t x1 = i,    y1 = j;
      uint8_t x2 = i+10, y2 = j;
      uint8_t x3 = i,    y3 = j+10;
      uint8_t x4 = i+10, y4 = j+10;
      tft.drawLine(x1, y1, x4, y4, random(0xffff));
      tft.drawLine(x2, y2, x3, y3, random(0xffff));
    }
  }
  delay(5000);
}
```

The output is as follows (Figure 5-15):

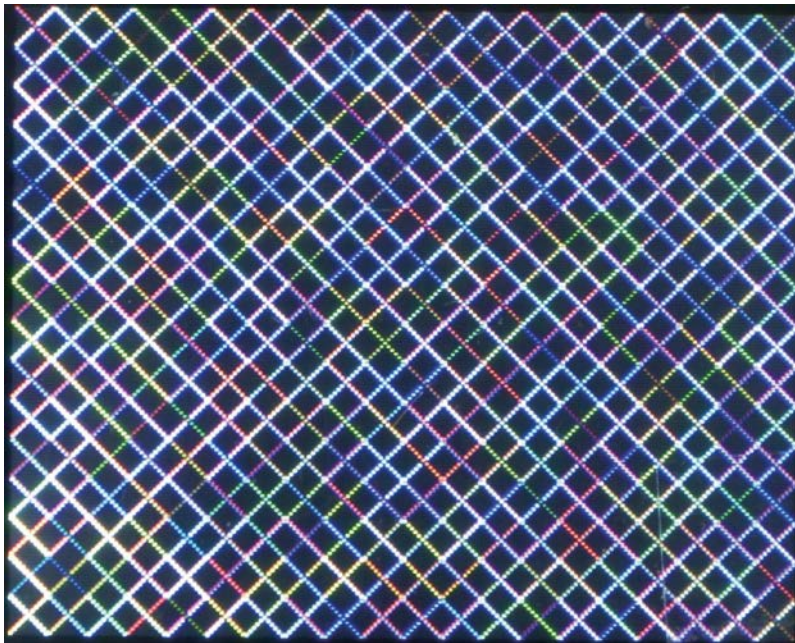


Figure 5-15: Random-colored lines creating rectangles.

You can also connect all the vertices of a cell and create pattern, as shown in the following sketch:

```
prog08.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void loop()
{
  tft.clear();
  for ( uint8_t i = 0; i < 176; i = i+11 )
  {
    for ( uint8_t j = 0; j < 220; j = j+11 )
    {
      uint8_t x1 = i,    y1 = j;
      uint8_t x2 = i+10, y2 = j;
      uint8_t x3 = i,    y3 = j+10;
      uint8_t x4 = i+10, y4 = j+10;
      tft.drawLine(x1, y1, x2, y2, random(0xffff));
      tft.drawLine(x1, y1, x3, y3, random(0xffff));
      tft.drawLine(x1, y1, x4, y4, random(0xffff));
      tft.drawLine(x2, y2, x3, y3, random(0xffff));
      tft.drawLine(x2, y2, x4, y4, random(0xffff));
      tft.drawLine(x3, y3, x4, y4, random(0xffff));
    }
  }
  delay(5000);
}
```

Which produces an output like in Figure 5-16.

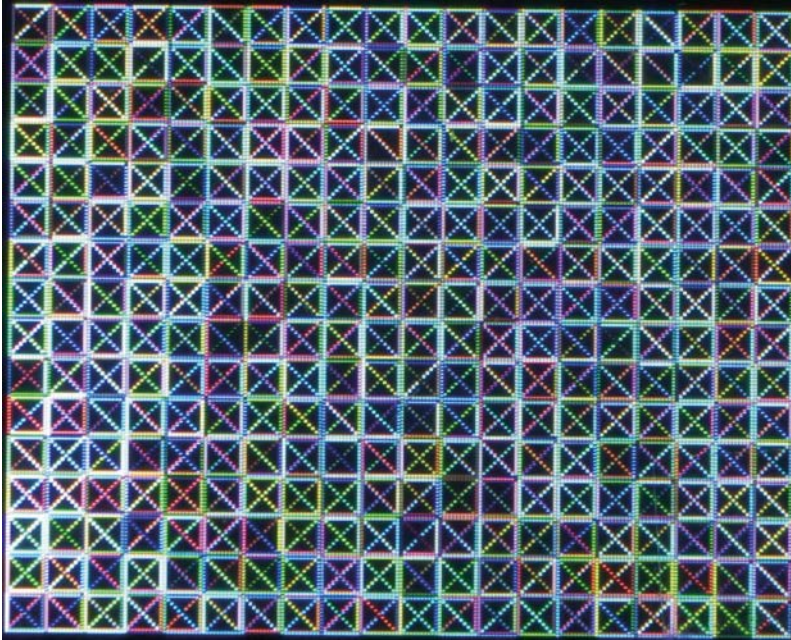


Figure 5-16: Random-colored lines creating rectangles and triangles.

Alternatively, draw circles of random colors as follows:

```
prog09.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void loop()
{
```

```
tft.clear();
for ( uint8_t i = 0; i < 176; i = i+11 )
{
  for ( uint8_t j = 0; j < 220; j = j+11 )
  {
    uint8_t x1 = i,    y1 = j;
    uint8_t x2 = i+10, y2 = j;
    uint8_t x3 = i,    y3 = j+10;
    uint8_t x4 = i+10, y4 = j+10;
    tft.drawCircle(i+5, j+5, 5, random(0xffff));
  }
}
delay(5000);
}
```

The output is like in Figure 5-17.

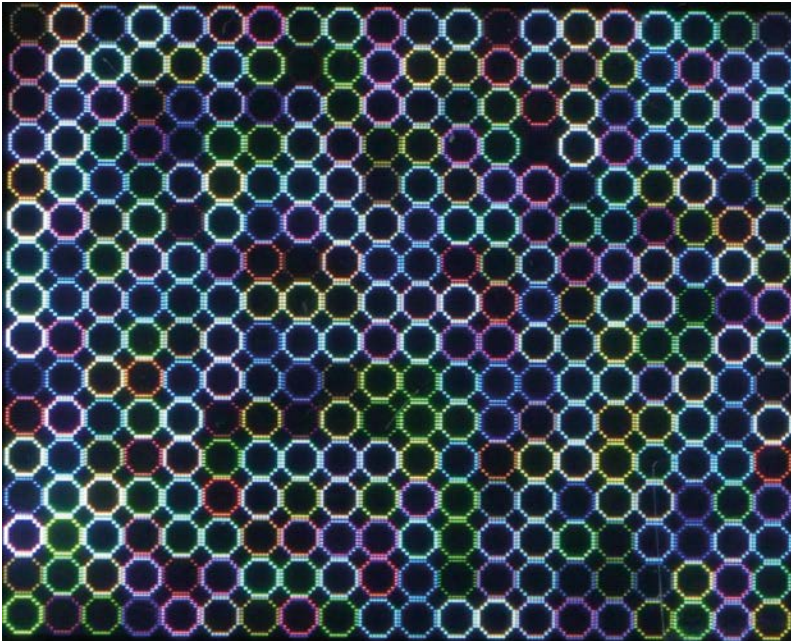


Figure 5-17: Random-colored circles.

Proceed by using the routine **fillCircle()** to draw solid-colored circles. You can find the code in the sketch called **prog10.ino** in the code bundle. The output is as follows (Figure 5-18):

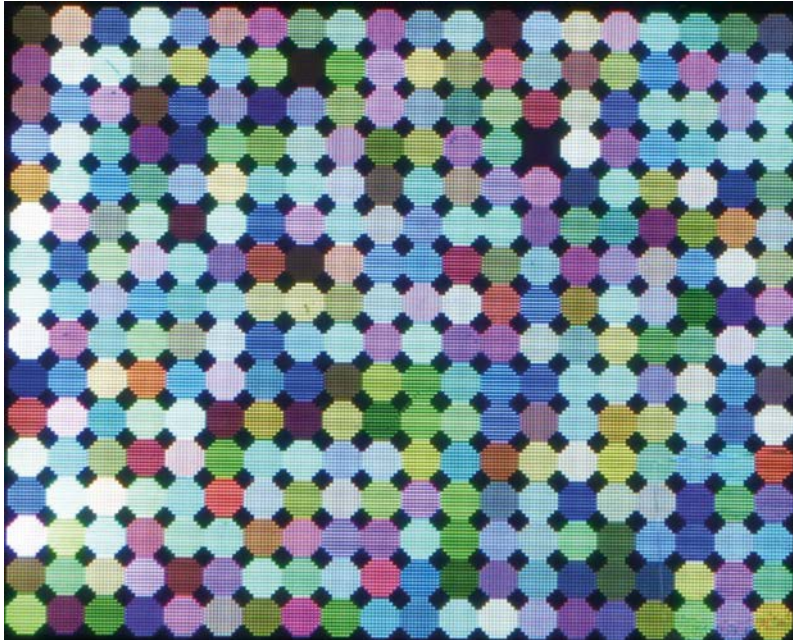


Figure 5-18: Random-filled circles.

Calling a routine within itself is known as recursion and you can use it to draw various geometric figures. There are two types of recursion: direct recursion and indirect. In direct recursion, the routine calls itself. In indirect recursion, there are multiple routines calling each other and thus creating a chain of function calls. You will use direct recursion to create a couple of geometric patterns.

In any recursive program, the function calls itself with slightly altered arguments. The recursive function has two parts: termination condition and recursive call. The termination condition decides when the recursive call should be made. In the absence of this, the recursion will run forever. The recursive call is part of the recursive function that calls itself with altered arguments.

If you are curious, you can read more about the Sierpinski triangle. Given the vertices of any triangle, you can divide that triangle in four triangles by connecting the midpoints of the sides. Go ahead and write a routine for it and pass the coordinates of the endpoints of the sides of the smaller triangles in each pass. This will create an interesting geometric pattern. The sketch is as follows:

```
prog11.ino  
#include "SPI.h"  
#include "TFT_22_ILI9225.h"  
#define TFT_CS  A4  
#define TFT_RST A3  
#define TFT_RS  A2
```

```
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void drawFractal(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t x3,
uint8_t y3, uint8_t i)
{
  if ( i != 0 )
  {
    drawFractal(x1, y1, (x1 + x2)/2 , (y1 + y2)/2 , (x1 + x3)/2 , (y1 + y3)/2,
i-1);
    drawFractal(x2, y2, (x1 + x2)/2 , (y1 + y2)/2 , (x2 + x3)/2 , (y2 + y3)/2,
i-1);
    drawFractal(x3, y3, (x3 + x2)/2 , (y3 + y2)/2 , (x1 + x3)/2 , (y1 + y3)/2,
i-1);
    tft.drawLine(x1, y1, x2, y2, random(0xffff));
    tft.drawLine(x1, y1, x3, y3, random(0xffff));
    tft.drawLine(x3, y3, x2, y2, random(0xffff));
  }
}
void loop()
{
  tft.clear();
  drawFractal(0, 0, 0, tft.maxY(), tft.maxX(), tft.maxY()/2, 8);
  delay(5000);
}
```

You are defining the routine **drawFractal()** that accepts the coordinates of the endpoints of the sides of triangles and the depth of the recursion. At every level, we are reducing the depth. The line with the **if** condition has the termination criteria. You are calling the function in the **loop** section. The image in Figure 5-19 shows the output.

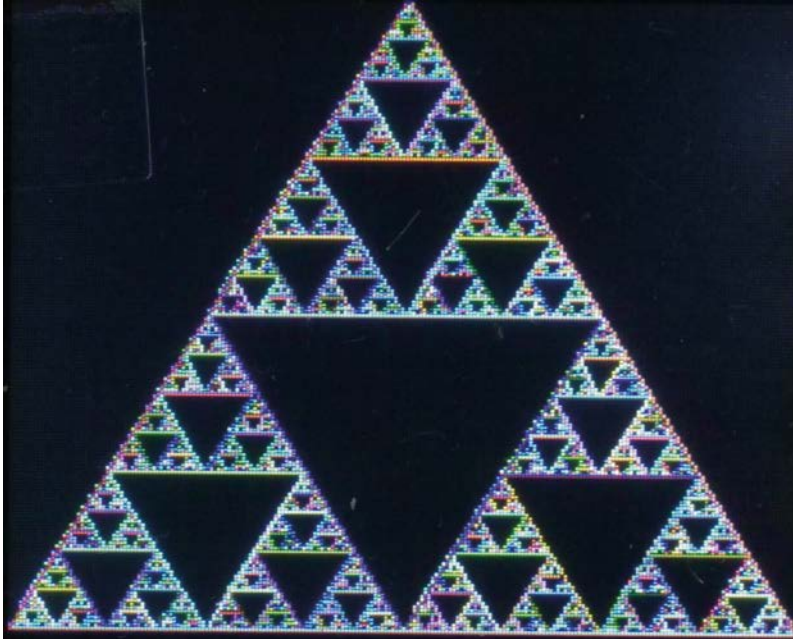


Figure 5-19: Sierpinski triangle.

You can even draw a square shape using this technique. Here is the relevant sketch:

```
prog12.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void drawFractal(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t i)
{
```

```
if ( i != 0 )
{
  drawFractal(x1, y1, (x1 + x2)/2-1, (y1 + y2)/2-1, i-1);
  drawFractal(x2, y2, (x1 + x2)/2, (y1 + y2)/2, i-1);
  drawFractal(x1, y2, (x1 + x2)/2-1, (y1 + y2)/2, i-1);
  drawFractal(x2, y1, (x1 + x2)/2, (y1 + y2)/2-1, i-1);
  if (i == 1)
  {
    uint8_t color = random(200, 256);
    switch(random(0, 7))
    {
      case 0:
        tft.drawRectangle(x1, y1, (x1 + x2)/2-1, (y1 + y2)/2-1, tft.setColor(0,
0, color));
        tft.drawRectangle(x2, y2, (x1 + x2)/2, (y1 + y2)/2, tft.setColor(0, 0,
color-30));
        tft.drawRectangle(x1, y2, (x1 + x2)/2-1, (y1 + y2)/2, tft.setColor(0, 0,
color-60));
        tft.drawRectangle(x2, y1, (x1 + x2)/2, (y1 + y2)/2-1, tft.setColor(0, 0,
color-90));
        break;
      case 1:
        tft.drawRectangle(x1, y1, (x1 + x2)/2-1, (y1 + y2)/2-1, tft.setColor(0,
color, 0));
        tft.drawRectangle(x2, y2, (x1 + x2)/2, (y1 + y2)/2, tft.setColor(0,
color-30, 0));
        tft.drawRectangle(x1, y2, (x1 + x2)/2-1, (y1 + y2)/2, tft.setColor(0,
color-60, 0));
        tft.drawRectangle(x2, y1, (x1 + x2)/2, (y1 + y2)/2-1, tft.setColor(0,
color-90, 0));
        break;
      case 2:
        tft.drawRectangle(x1, y1, (x1 + x2)/2-1, (y1 + y2)/2-1, tft.
setColor(color, 0, 0));
        tft.drawRectangle(x2, y2, (x1 + x2)/2, (y1 + y2)/2, tft.setColor(color-30,
0, 0));
        tft.drawRectangle(x1, y2, (x1 + x2)/2-1, (y1 + y2)/2, tft.
setColor(color-60, 0, 0));
        tft.drawRectangle(x2, y1, (x1 + x2)/2, (y1 + y2)/2-1, tft.
setColor(color-90, 0, 0));
        break;
      case 3:
        tft.drawRectangle(x1, y1, (x1 + x2)/2-1, (y1 + y2)/2-1, tft.
setColor(color, color, 0));
        tft.drawRectangle(x2, y2, (x1 + x2)/2, (y1 + y2)/2, tft.setColor(color-30,
color-30, 0));
```



```

        tft.drawRectangle(x1, y2, (x1 + x2)/2-1, (y1 + y2)/2, tft.
setColor(color-60, color-60, 0));
        tft.drawRectangle(x2, y1, (x1 + x2)/2, (y1 + y2)/2-1, tft.
setColor(color-90, color-90, 0));
        break;
    case 4:
        tft.drawRectangle(x1, y1, (x1 + x2)/2-1, (y1 + y2)/2-1, tft.
setColor(color, color, color));
        tft.drawRectangle(x2, y2, (x1 + x2)/2, (y1 + y2)/2, tft.setColor(color-30,
color-30, color-30));
        tft.drawRectangle(x1, y2, (x1 + x2)/2-1, (y1 + y2)/2, tft.
setColor(color-60, color-60, color-60));
        tft.drawRectangle(x2, y1, (x1 + x2)/2, (y1 + y2)/2-1, tft.
setColor(color-90, color-90, color-90));
        break;
    case 5:
        tft.drawRectangle(x1, y1, (x1 + x2)/2-1, (y1 + y2)/2-1, tft.
setColor(color, 0, color));
        tft.drawRectangle(x2, y2, (x1 + x2)/2, (y1 + y2)/2, tft.setColor(color-30,
0, color-30));
        tft.drawRectangle(x1, y2, (x1 + x2)/2-1, (y1 + y2)/2, tft.
setColor(color-60, 0, color-60));
        tft.drawRectangle(x2, y1, (x1 + x2)/2, (y1 + y2)/2-1, tft.
setColor(color-90, 0, color-90));
        break;
    case 6:
        tft.drawRectangle(x1, y1, (x1 + x2)/2-1, (y1 + y2)/2-1, tft.setColor(0,
color, color));
        tft.drawRectangle(x2, y2, (x1 + x2)/2, (y1 + y2)/2, tft.setColor(0,
color-30, color-30));
        tft.drawRectangle(x1, y2, (x1 + x2)/2-1, (y1 + y2)/2, tft.setColor(0,
color-60, color-60));
        tft.drawRectangle(x2, y1, (x1 + x2)/2, (y1 + y2)/2-1, tft.setColor(0,
color-90, color-90));
        break;
    }
}
}
}
void loop()
{
    tft.clear();
    drawFractal(0, 0, tft.maxX(), tft.maxY(), 6);
    delay(5000);
}

```

The output of this sketch is shown in Figure 5-20.

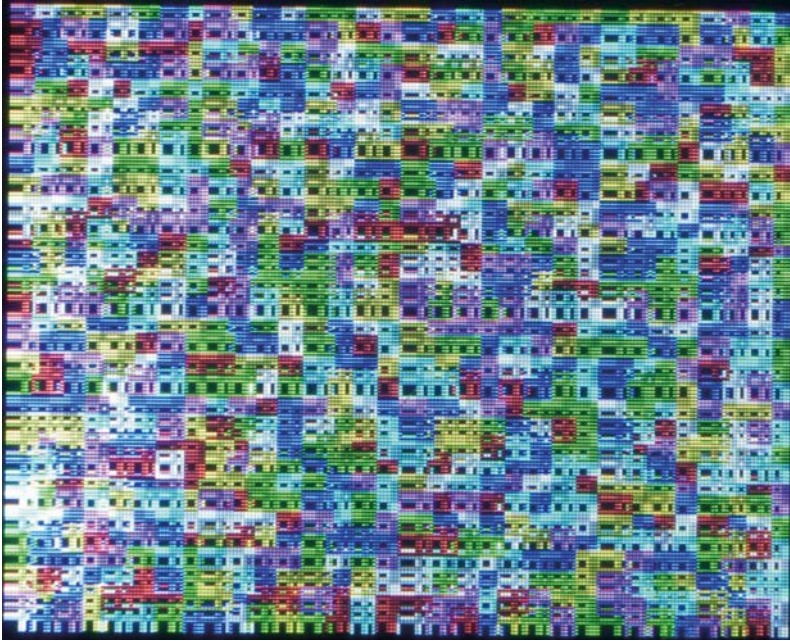


Figure 5-20: Fractal square.

The geometric shapes created using recursion sometimes fall under the category of **fractal**. Both examples you programmed are fractals. You can even apply geometric transformations on shapes. Let's see how to create the rotation effect on a triangle. I have referred the examples posted at:

https://create.arduino.cc/projecthub/Arnov_Sharma_makes/getting-started-with-ili9255-tft-lcd-378331

after making a few changes to simplify it.

Let's study that code block by block. First, we initialize the display:

```
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
```

```
Let's define the angle of rotation,
#define ROTATE_ANGLE 15
```

Let's define a custom structure for points:

```
struct Point
{
    uint8_t x;
    uint8_t y;
};
```

And then create the corresponding variables:

```
Point c1, c2, c3, cc;
```

These are the vertices and the centroid of the triangle. Let's define a routine to rotate a point **P** with respect to another point **C** with a given angle as follows:

```
Point rotatePoint( Point c, float angle, Point p )
{
    Point r;
    r.x = cos(angle) * (p.x - c.x) - sin(angle) * (p.y - c.y) + c.x;
    r.y = sin(angle) * (p.x - c.x) + cos(angle) * (p.y - c.y) + c.y;
    return r;
}
```

Let's write a custom routine for computing the centroid of a triangle,

```
Point getCoordCentroid( Point a, Point b, Point c )
{
    Point o;
    o.x = (int16_t)((a.x + b.x + c.x) / 3);
    o.y = (int16_t)((a.y + b.y + c.y) / 3);
    return o;
}
```

Carry on by writing a custom routine to rotate all the three points of a triangle:

```
void rotateTriangle( Point &a, Point &b, Point &c, Point r, int16_t deg )
{
    float angle = (float)deg * 1000 / 57296;
    a = rotatePoint( r, angle, a);
    b = rotatePoint( r, angle, b);
    c = rotatePoint( r, angle, c);
}
```

Let's define a triangle and compute the centroid,

```
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);

  c1.x = 30;  c1.y = 30;
  c2.x = 120; c2.y = 80;
  c3.x = 80;  c3.y = 130;
  cc = getCoordCentroid(c1, c2, c3);

  Serial.begin(9600);
}
```

Finally, we are computing the number of steps in the rotation and running the loop to rotate the triangle:

```
void loop()
{
  tft.clear();
  int16_t steps = (int16_t)(360 / ROTATE_ANGLE);
  for (int8_t i = 0; i < steps; i++)
  {
    tft.drawTriangle(c1.x, c1.y, c2.x, c2.y, c3.x, c3.y, COLOR_RED);
    rotateTriangle(c1, c2, c3, cc, ROTATE_ANGLE);
    delay(50);
  }
  delay(3000);
}
```

The glorious output is as in Figure 5-21.

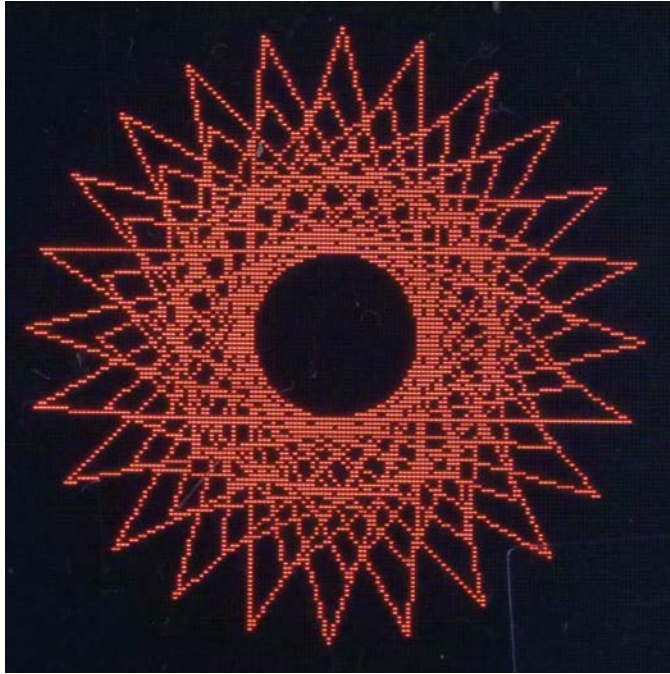


Figure 5-21: Triangle after full rotation.

This is the output after the triangle completes a full cycle of rotation. Be creative and create a zigzag pattern as follows:

```
prog13.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
```

```
void loop()
{
  tft.clear();
  for ( uint8_t i = 0; i < 176; i = i+11 )
  {
    for ( uint8_t j = 0; j < 220; j = j+11 )
    {
      uint8_t x1 = i,    y1 = j;
      uint8_t x2 = i+10, y2 = j;
      uint8_t x3 = i,    y3 = j+10;
      uint8_t x4 = i+10, y4 = j+10;
      if ( j%2 )
        tft.drawLine(x1, y1, x4, y4, random(0xffff));
      else
        tft.drawLine(x2, y2, x3, y3, random(0xffff));
      delay(5);
    }
  }
  delay(5000);
}
```

The output is as in Figure 5-22.

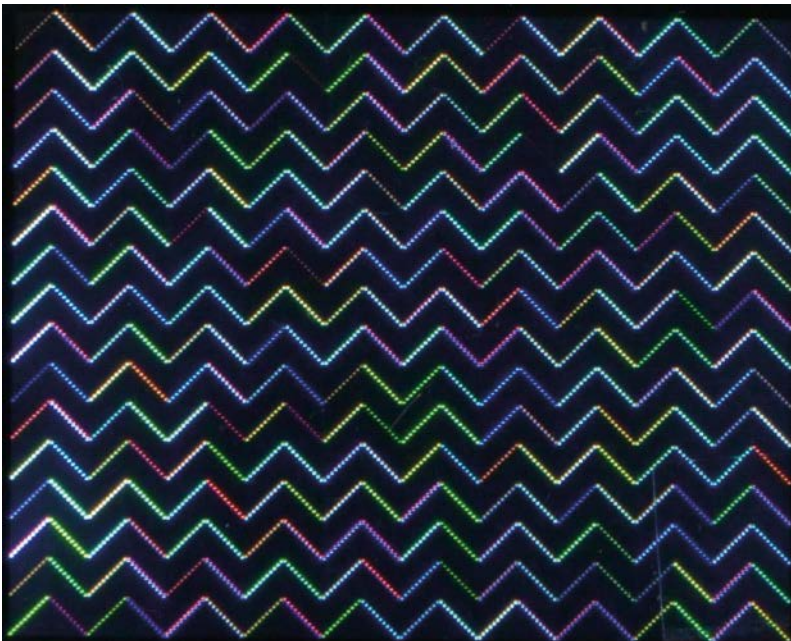


Figure 5-22: Zigzag pattern.

We can change the axis of the pattern by changing the condition to **if (i%2)**. You can find the program as **prog14_1.ino** in the code bundle for this chapter. The output is as in Figure 5-23.

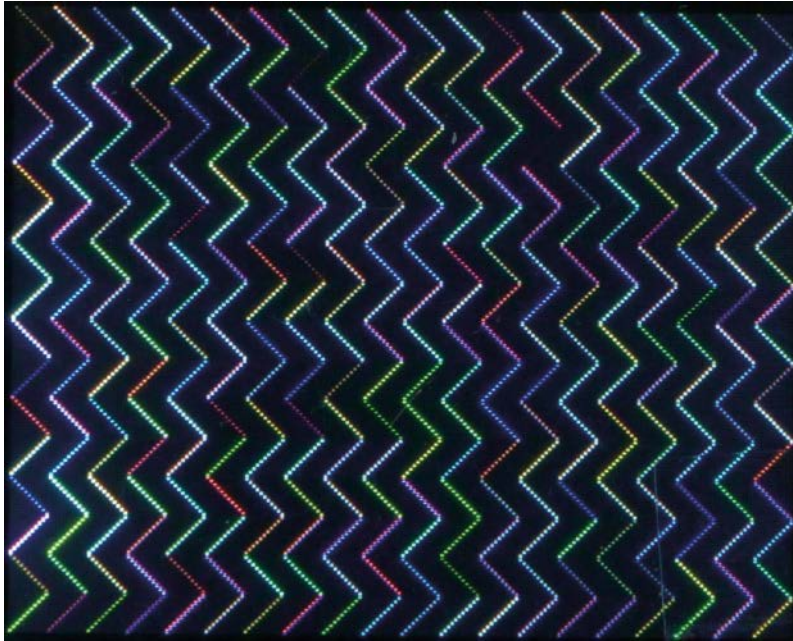


Figure 5-23: Zigzag pattern.

Moreover, you can change the condition to **if(random(2))**. You can find the code in **prog14_2.ino**. The output will look like Figure 5-24.

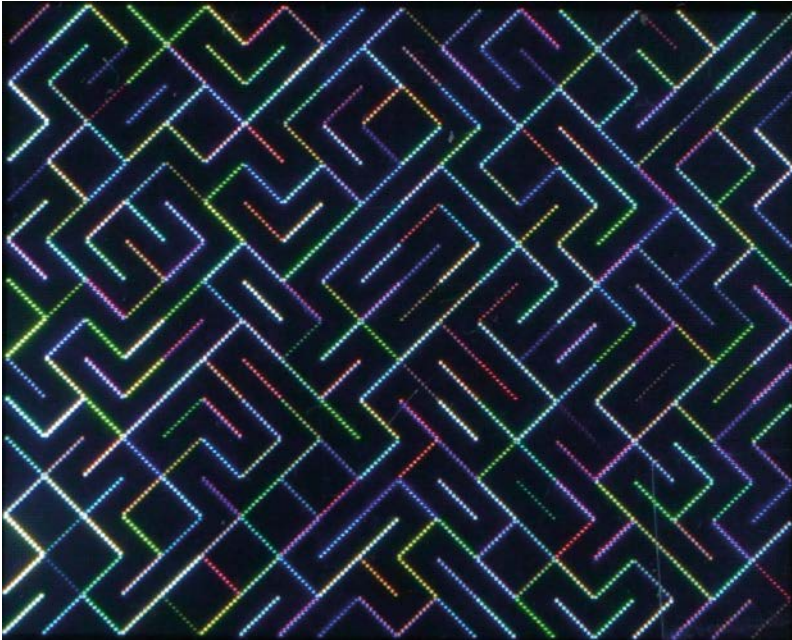


Figure 5-24: Zigzag maze pattern.

Consider creating a more intricate maze structure with the following sketch:

```
prog15.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void loop()
{
```



```
tft.clear();
for ( uint8_t i = 0; i < 176; i = i+11 )
{
  for ( uint8_t j = 0; j < 220; j = j+11 )
  {
    uint8_t x1 = i,    y1 = j;
    uint8_t x2 = i+10, y2 = j;
    uint8_t x3 = i,    y3 = j+10;
    uint8_t x4 = i+10, y4 = j+10;
    if(random(2))
      tft.drawLine(x1, y1, x4, y4, random(0xffff));
    else
      tft.drawLine(x2, y2, x3, y3, random(0xffff));
    if(random(2))
      tft.drawLine(x1, y1, x4, y4, random(0xffff));
    else
      tft.drawLine(x2, y2, x3, y3, random(0xffff));
    delay(5);
  }
}
delay(5000);
}
```

The output is as photographed for Figure 5-25.

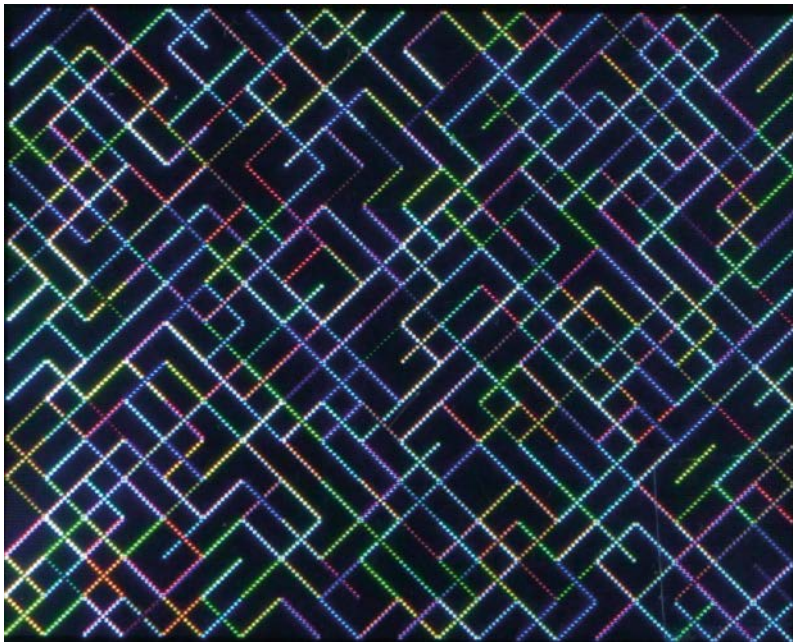


Figure 5-25: A more intricate zigzag maze pattern.

Yet another way to create a pattern using straight lines is as follows:

```
prog16.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS A4
#define TFT_RST A3
#define TFT_RS A2
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void loop()
{
  tft.clear();
  for ( uint8_t i = 0; i < 176; i = i+11 )
  {
    for ( uint8_t j = 0; j < 220; j = j+11 )
    {
      uint8_t x1 = i,    y1 = j;
      uint8_t x2 = i+10, y2 = j;
      uint8_t x3 = i,    y3 = j+10;
      uint8_t x4 = i+10, y4 = j+10;
      switch(random(6))
      {
        case 0:
          tft.drawLine(x1, y1, x4, y4, random(0xffff));
          break;
        case 1:
          tft.drawLine(x2, y2, x3, y3, random(0xffff));
          break;
        case 2:
          tft.drawLine(x1, y1, x2, y2, random(0xffff));
          break;
        case 3:
          tft.drawLine(x3, y3, x4, y4, random(0xffff));

```

```

    break;
    case 4:
    tft.drawLine(x1, y1, x3, y3, random(0xffff));
    break;
    case 5:
    tft.drawLine(x2, y2, x4, y4, random(0xffff));
    break;
    }
    delay(5);
  }
}
delay(5000);
}

```

Which produces an image like in Figure 5-26.

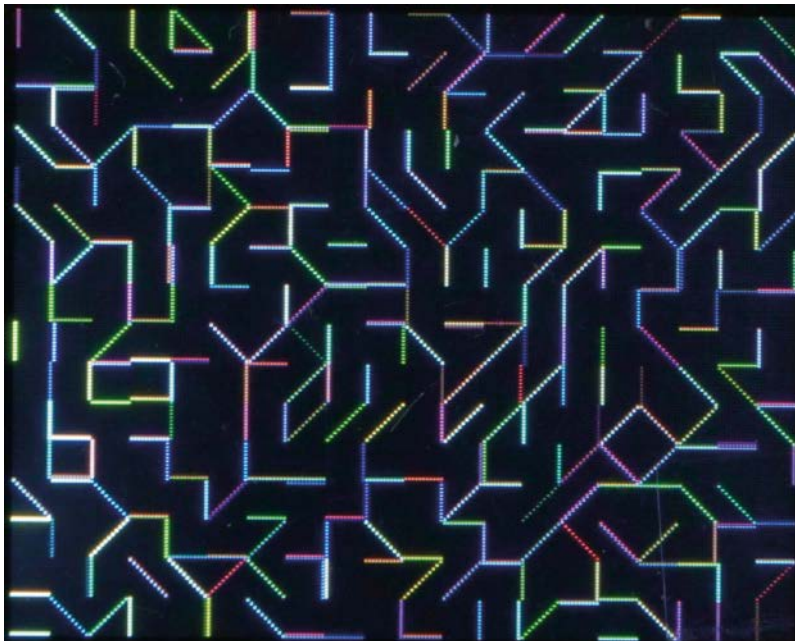


Figure 5-26: Alternative-method zigzag maze pattern.

Let's create random more patterns with random squares, circles, and lines, as follows:

```

prog17.ino
#include "SPI.h"
#include "TFT_22_ILI9225.h"
#define TFT_CS  A4
#define TFT_RST A3
#define TFT_RS  A2

```

```
#define TFT_SDI A1
#define TFT_CLK A0
#define TFT_BRIGHTNESS 200
TFT_22_ILI9225 tft = TFT_22_ILI9225(TFT_RST, TFT_RS, TFT_CS, TFT_SDI, TFT_CLK,
TFT_BRIGHTNESS);
void setup()
{
  tft.begin();
  tft.setBacklight(true);
  tft.setDisplay(true);
  tft.setBackgroundColor(COLOR_BLACK);
  tft.setOrientation(0);
  Serial.begin(9600);
}
void loop()
{
  tft.clear();
  for ( uint8_t i = 0; i < 176; i = i+11 )
  {
    for ( uint8_t j = 0; j < 220; j = j+11 )
    {
      uint8_t x1 = i,    y1 = j;
      uint8_t x2 = i+10, y2 = j;
      uint8_t x3 = i,    y3 = j+10;
      uint8_t x4 = i+10, y4 = j+10;
      switch(random(7))
      {
        case 0:
          tft.drawLine(x1, y1, x4, y4, random(0xffff));
          break;
        case 1:
          tft.drawLine(x2, y2, x3, y3, random(0xffff));
          break;
        case 2:
          tft.drawRect(x1, y1, x4, y4, random(0xffff));
          break;
        case 3:
          tft.drawRect(x1, y1, (x1+x4)/2, (y1+y4)/2, random(0xffff));
          tft.drawRect(x4, y4, (x1+x4)/2+1, (y1+y4)/2+1, random(0xffff));
          break;
        case 4:
          tft.drawCircle((x1+x4)/2, (y1+y4)/2, 5, random(0xffff));
          break;
        case 5:
          tft.drawRect(x1, y1, x4, y4, random(0xffff));
          tft.drawRect(x1+3, y1+3, x4-3, y4-3, random(0xffff));
      }
    }
  }
}
```

```
break;
case 6:
word color = random(0xffff);
tft.drawCircle((x1+x4)/2, (y1+y4)/2, 5, color);
tft.drawCircle((x1+x4)/2, (y1+y4)/2, 4, color);
tft.drawCircle((x1+x4)/2, (y1+y4)/2, 3, color);
break;
}
delay(5);
}
}
delay(5000);
}
```

The rather nice looking output is pictured in Figure 5-27.

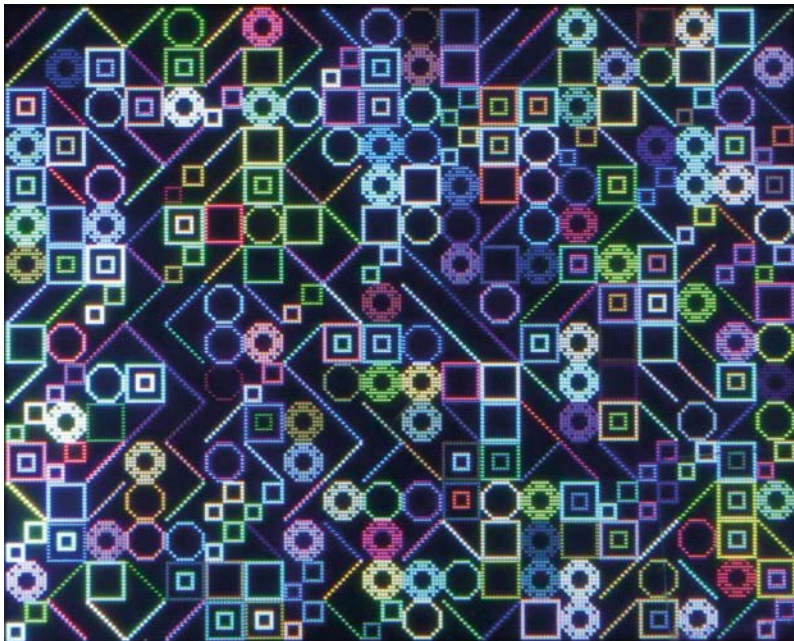


Figure 5-27: Random pattern.

This closes off our discussion on creating patterns with the ILI9225 library. At this point, you can start writing a program for a simple game of *Snake* using this display and a few buttons.

Summary

In this chapter, you learned how to work with ILI9225 display and create animations. In the next chapter, you will learn how to work with various sensors and output devices including a temperature sensor, a piezo buzzer, and a joystick.

Chapter 6 • Working with a Buzzer and a Sensor

In the previous chapter, you learned how to work with various displays with Arduino Nano family boards. You are now comfy using various types of displays with Arduino.

In this short chapter, you will learn how to work with a few sensors and buzzer. These are the topics you will learn in this chapter:

- Working with a buzzer
- Working with a joystick
- Working with a temperature sensor

After absorbing this chapter, you will be comfortable with interfacing your Arduino Nano with peripheral devices like a buzzer, a temperature sensor, and a joystick.

Working with a Buzzer

You can use a piezo buzzer to produce a single tone of a specific frequency at a time. A buzzer has two pins, one positive and one negative. Connect the positive pin to a digital output pin of the Nano board and the negative pin to GND. Let's create a circuit as shown in Figure 6-1.

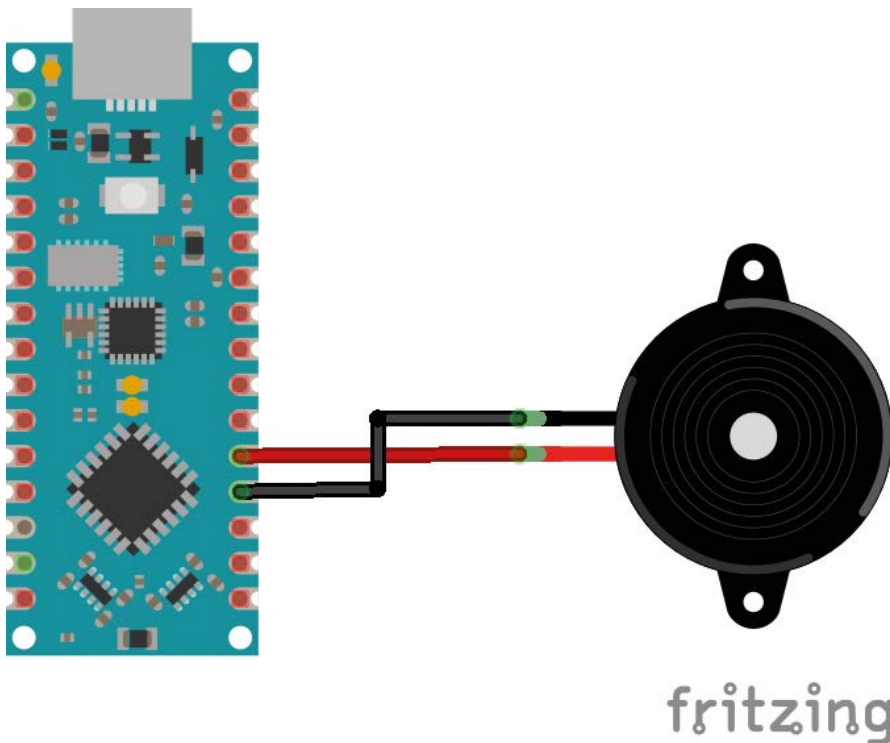


Figure 6-1: A piezo buzzer connected to an Arduino Nano.

The buzzer's positive pin (red wire) is connected to the digital I/O pin on the Nano, and the negative pin (black wire) is connected to GND. You will use this circuit for the next couple of sketches. For this, we will use a built-in routine **tone()**. It generates a square wave of a specified frequency with 50% duty cycle. You can generate only one tone at a time on a pin and board. In other words, if you are using a specific pin to emit a tone, when you call this function again with some other pin, it will not work. If you are already emitting a tone on a pin and you call this routine again with the changed frequency, it will change the frequency of the tone. The routine has three arguments. The first and the second ones are mandatory, and they are defined as the output pin and the frequency. The third argument is optional: it is the duration in milliseconds for which the tone is played on the buzzer. If the third argument is not mentioned, the buzzer keeps emitting the specified tone unless another routine **noTone()** is called. The following sketch shows the usage of the routine **tone()**.

```
prog00.ino
const int piezzo_pin = 2;
void setup()
{
  Serial.begin(9600);
  pinMode(piezzo_pin, OUTPUT);
}
void loop()
{
  for (long i = 0; i <= 65535; i=i+500)
  {
    tone(piezzo_pin, i, 2000);
    Serial.println(i);
  }
}
```

On the Arduino Nano family of boards, you can generate a square wave of frequency ranging from 31 Hz up to 65535 Hz. Since the data type **int** cannot handle this range, you have to use **long** type for storing the frequencies. We can write the similar code using the routine **noTone()** as follows:

```
prog01.ino
const int piezzo_pin = 2;

void setup()
{
  Serial.begin(9600);
  pinMode(piezzo_pin, OUTPUT);
}

void loop()
{
  for (long i = 31; i <= 20000; i=i+500)
```



```
{
  tone(piezzo_pin, i);
  Serial.println(i);
  delay(2000);
  noTone(piezzo_pin);
  delay(500);
}
```

Upload and run both sketches. You can read more about the **tone()** in the Arduino Reference docs at: <https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/> and at: <https://www.arduino.cc/en/reference/tone>.

You can also use this for more complex projects. Attach an analog input device like a trimpot to pin A0, as shown in Figure 6-2.

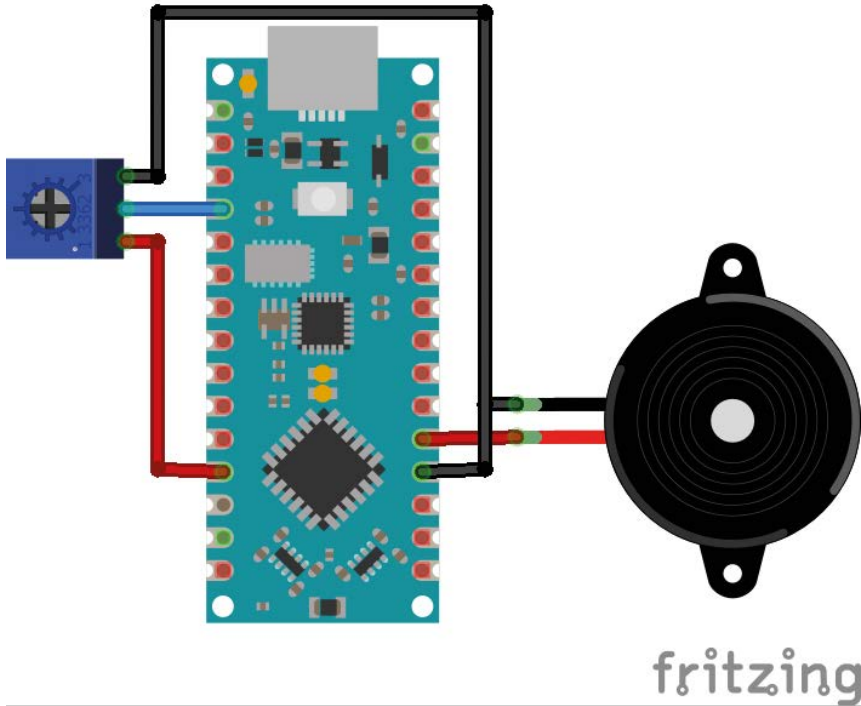


Figure 6-2: A piezo buzzer and a trimpot connected to a Nano.

I am a great fan of *Star Wars*. Using this circuit, you can emit the Darth Vader theme. The speed of the theme play can be adjusted with the trimpot knob. I borrowed the code from an online GitHub project (<https://gist.github.com/nicksort>) and made appropriate changes to the code and circuit. This is the code:

prog02.ino

```
const int c = 261;
const int d = 294;
const int e = 329;
const int f = 349;
const int g = 391;
const int gS = 415;
const int a = 440;
const int aS = 455;
const int b = 466;
const int cH = 523;
const int cSH = 554;
const int dH = 587;
const int dSH = 622;
const int eH = 659;
const int fH = 698;
const int fSH = 740;
const int gH = 784;
const int gSH = 830;
const int aH = 880;

const int buzzerPin = 2;
int delay_time = 50;

void beep(int note, int duration)
{
  tone(buzzerPin, note);
  delay(duration);
  noTone(buzzerPin);
  delay_time = map(analogRead(0), 0, 1023, 10, 90);
  Serial.println(delay_time);
  delay(delay_time);
}

void firstSection()
{
  beep(a, 500);
  beep(a, 500);
  beep(a, 500);
  beep(f, 350);
  beep(cH, 150);
  beep(a, 500);
  beep(f, 350);
  beep(cH, 150);
  beep(a, 650);
}
```

```
    delay(500);

    beep(eH, 500);
    beep(eH, 500);
    beep(eH, 500);
    beep(fH, 350);
    beep(cH, 150);
    beep(gS, 500);
    beep(f, 350);
    beep(cH, 150);
    beep(a, 650);

    delay(500);
}

void secondSection()
{
    beep(aH, 500);
    beep(a, 300);
    beep(a, 150);
    beep(aH, 500);
    beep(gSH, 325);
    beep(gH, 175);
    beep(fSH, 125);
    beep(fH, 125);
    beep(fSH, 250);

    delay(325);

    beep(aS, 250);
    beep(dSH, 500);
    beep(dH, 325);
    beep(cSH, 175);
    beep(cH, 125);
    beep(b, 125);
    beep(cH, 250);

    delay(350);
}

void setup()
{
    Serial.begin(9600);
    pinMode(buzzerPin, OUTPUT);
}
```

```
void loop()
{
  firstSection();
  secondSection();

  beep(f, 250);
  beep(gS, 500);
  beep(f, 350);
  beep(a, 125);
  beep(cH, 500);
  beep(a, 375);
  beep(cH, 125);
  beep(eH, 650);

  delay(500);

  secondSection();

  beep(f, 250);
  beep(gS, 500);
  beep(f, 375);
  beep(cH, 125);
  beep(a, 500);
  beep(f, 375);
  beep(cH, 125);
  beep(a, 650);
  delay(650);
}
```

Since I have already explained the functions used in this code earlier, I will discuss only the logic. First, you are defining the frequencies of the tone used for creating the music. Second, you are defining a custom routine **beep()** that emits a tone of a particular frequency for a specified time. The delay between the individual tones is detected by the position of the trimpot. Since a few tone sequences are repetitive, there is a need for the routines **firstSection()** and **secondSection()**. In the **loop()** section, you are calling them, along with other tones, to create the Darth Vader theme music. Upload the sketch and enjoy the music.

Working with a Joystick

The joystick is a combination of two potentiometers and a momentary switch. The potentiometers are used for choosing the X and Y positions. Let's create a simple circuit with the joystick and a Nano board, as shown in Figure 6-3.

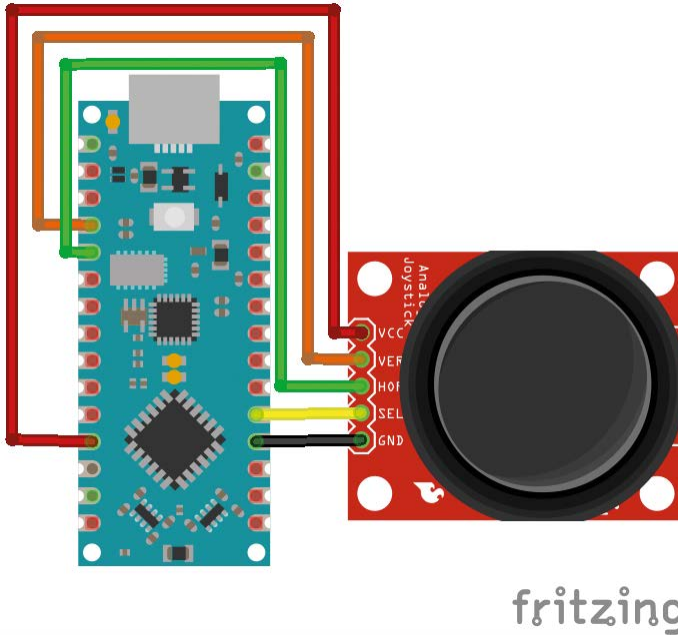


Figure 6-3: Joystick connections.

Each manufacturer will have its own naming scheme for the pins. However, the most common joysticks have 5 pins. Connect the X position pin to A1, the Y position pin to A0, and connect the pin for the switch to digital I/O pin 2. In addition, you have to connect the VCC and GND terminals to the +5 V and GND pins of the board, respectively. Now, it is time to write the code. The code is pretty simple and I have already covered all functions used.

```
prog03.ino
int hor = A1;
int ver = A0;
int button = 2;
int x = 0;
int y = 0;
int button_state = 0;
const int range = 64;
void setup() {
  Serial.begin(9600);
  pinMode(hor, INPUT);
  pinMode(ver, INPUT);
  pinMode(button, INPUT_PULLUP);
}
void loop() {
  x = analogRead(hor);
  y = analogRead(ver);
  button_state = digitalRead(button);
```

```

Serial.print("X: ");
Serial.print(map(x, 0, 1023, -range, range-1));
Serial.print(" | Y: ");
Serial.print(map(y, 0, 1023, -range, range-1));
Serial.print(" | Button: ");
Serial.println(button_state);
delay(100);
}

```

We are mapping the X and Y positions to a custom range (-64 to 63) and displaying the status of all the inputs using a serial monitor. Figure 6-4 shows the joystick in action.

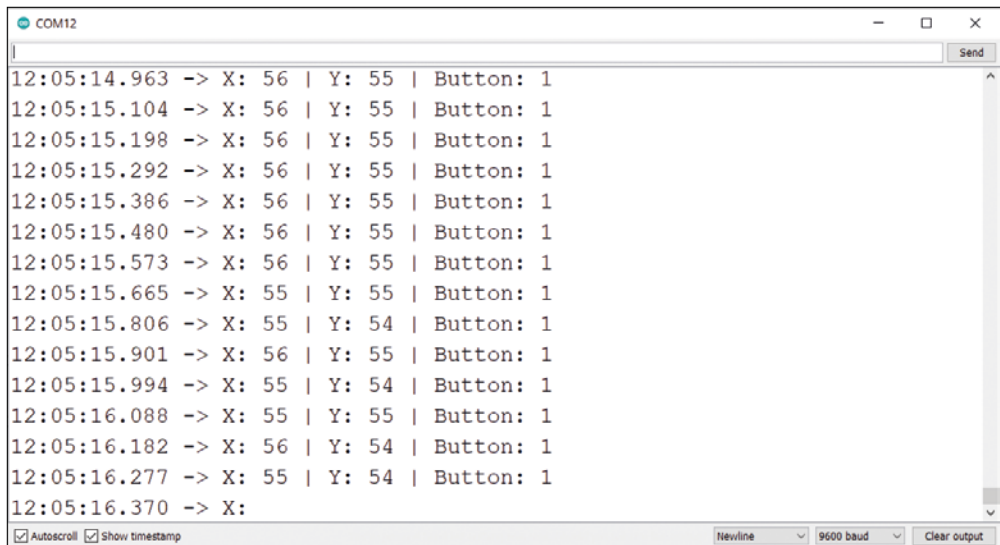


Figure 6-4: Joystick in action.

This is how you can work with a joystick. In the previous chapter, I suggested you create a snake game with Nokia 5110 display. You can add this joystick to control the snake.

Working with DS18B20 Temperature Sensor

Now, Let's connect a DS18B20 sensor to the Nano board. Connect the middle pin of the sensor to the digital pin 2. Then connect + and - pins to the +5 V and GND pins of the Nano board, respectively, as shown in Figure 6-5.

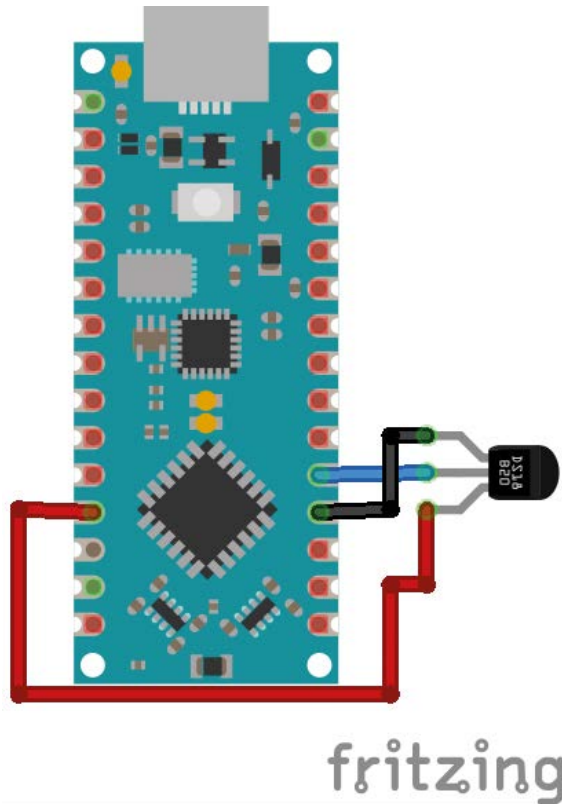


Figure 6-5: DS18B20 sensor connections.

This sensor requires **OneWire** protocol and a special library. Open the library manager and search for the term "DS18B20". It will show multiple libraries in the result. Install the **DallasTemperature** library. While installing, it will prompt you for installing another library, **OneWire**. Install that library too. Once done, upload the following sketch:

```
prog04.ino
#include <OneWire.h>
#include <DallasTemperature.h>
const int sensor_pin = 2;
OneWire oneWire(sensor_pin);
DallasTemperature sensors(&oneWire);
void setup()
{
  Serial.begin(9600);
  sensors.begin();
}
void loop()
{
  sensors.requestTemperatures();
```

```
Serial.print("\nThe current temperature is: ");  
Serial.print(sensors.getTempCByIndex(0));  
delay(1000);  
}
```

The code uses all the library functions of OneWire and DallasTemperature library. The output is as shown in Figure 6-6.

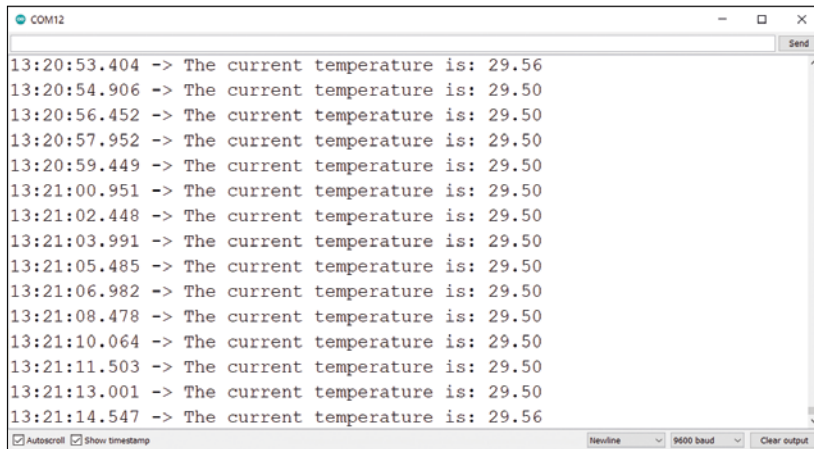
A screenshot of a serial monitor window titled 'COM12'. The window displays a series of timestamped messages: '13:20:53.404 -> The current temperature is: 29.56', '13:20:54.906 -> The current temperature is: 29.50', '13:20:56.452 -> The current temperature is: 29.50', '13:20:57.952 -> The current temperature is: 29.50', '13:20:59.449 -> The current temperature is: 29.50', '13:21:00.951 -> The current temperature is: 29.50', '13:21:02.448 -> The current temperature is: 29.50', '13:21:03.991 -> The current temperature is: 29.50', '13:21:05.485 -> The current temperature is: 29.50', '13:21:06.982 -> The current temperature is: 29.50', '13:21:08.478 -> The current temperature is: 29.50', '13:21:10.064 -> The current temperature is: 29.50', '13:21:11.503 -> The current temperature is: 29.50', '13:21:13.001 -> The current temperature is: 29.50', and '13:21:14.547 -> The current temperature is: 29.56'. At the bottom of the window, there are checkboxes for 'Autoscroll' and 'Show timestamp', and a dropdown menu for 'Newline' set to '9600 baud' and a 'Clear output' button.

Figure 6-6: DS18B20 temperature sensor in action.

Summary

In this chapter, you learned how to work with diverse input and output devices such as a buzzer, a joystick, and a temperature sensor.

In the next chapter, you will learn how to create IoT projects.

Chapter 7 • Working with the Arduino Nano 33 IoT

In the last chapter, you studied how to connect the temperature sensor, piezo buzzer, and joystick to Arduino Nano. In this chapter, you will learn to use another member of the Arduino Nano family, **Arduino Nano 33 IoT**. This is the list of topics you will learn about and demonstrate in this chapter:

- Introduction to the Nano 33 IoT board
- Getting started
- Working with the WiFININA library
- A Telnet-based group chat server
- Pinging a remote server
- A simple web client
- Working with a real-time clock
- Using DS18B20 temperature sensors jointly with an RTC
- Visualizing a temperature graph with ThingSpeak
- Programming the built-in IMU

After completing this chapter, you will be more than comfortable with the Arduino Nano 33 IoT board.

Introduction to the Nano 33 IoT board

Let's get acquainted with the **Arduino Nano 33 IoT**. This board is pin-to-pin compatible with all the other microcontroller boards in the Arduino Nano family. IoT stands for **Internet of Things**. You can use the Nano 33 IoT for running IoT projects that have small footprints in terms of hardware and functionality. This chapter covers a few small web-based IoT projects, such as controlling LEDs from the web and displaying the temperature over the web. Also, you can just replace any other board from the Arduino Nano family with a Nano 33 IoT.

Let's discuss the hardware specifications of the Nano 33 IoT. The Nano 33 IoT comes in two versions: with and without pinheaders. Due to the shortage of semiconductors and electronic components, I found it difficult to procure them in my region while authoring this book (May 2022). I hope you can procure it without problems once the semiconductor shortage is resolved. I am using the version of this board with pinheaders. However, in case you are only able to procure the one without headers, soldering the headers is an easy job.

The Arduino Nano 33 IoT comes with a Microchip **SAMD21 Cortex®-M0+ 32bit low power ARM MCU** as the main chip and its datasheet is at:

https://content.arduino.cc/assets/mkr-microchip_samd21_family_full_datasheet-ds40001882d.pdf.

The Wi-Fi and Bluetooth connectivity are provided by **u-blox NINA-W102** module, whose datasheet is at:

https://content.arduino.cc/assets/Arduino_NINA-W10_DataSheet_%28UBX-17065507%29.pdf.

The Nano 33 IoT also has a **Microchip ATECC608A** chip for secure communication. You can check the datasheet at:

https://content.arduino.cc/assets/microchip_atecc608a_cryptoauthentication_device_summary_datasheet-DS40001977B.pdf.

Furthermore, there is also an Inertial Measurement Unit chip type **IMU LSM6DS3** and you can check its datasheet at:

https://content.arduino.cc/assets/st_imu_lsm6ds3_datasheet.pdf.

The following table summarizes other salient features of the board.

Operating voltage	3.3V
Input voltage (limit)	21V
DC current per I/O pin	7 mA
Clock speed	48 MHz
CPU flash memory	256 KB
SRAM	32 KB
EEPROM	none
Digital input / output pins	14
PWM pins	11 (2, 3, 5, 6, 9, 10, 11, 12, 16, 17, 19)
UART	1
SPI	1
I2C	1
Analog input pins	8 (ADC 8/10/12 bit)
Analog output pins	1 (DAC 10 bit)
LEDs, on-board	13
Length	45 mm
Width	18 mm
Weight	5 gr (with headers)

You can download the Fritzing part from:

<https://content.arduino.cc/assets/Arduino%20Nano%2033%20IoT.fzpz>.

Figures 7-1, 7-2, and 7-3 refer to the document https://content.arduino.cc/assets/Pinout-NANO33IoT_latest.pdf which is published under the <https://creativecommons.org/licenses/by/4.0/> license. I have modified the images for the purpose of this publication.

The board's pinout can be seen in Figure 7-1.

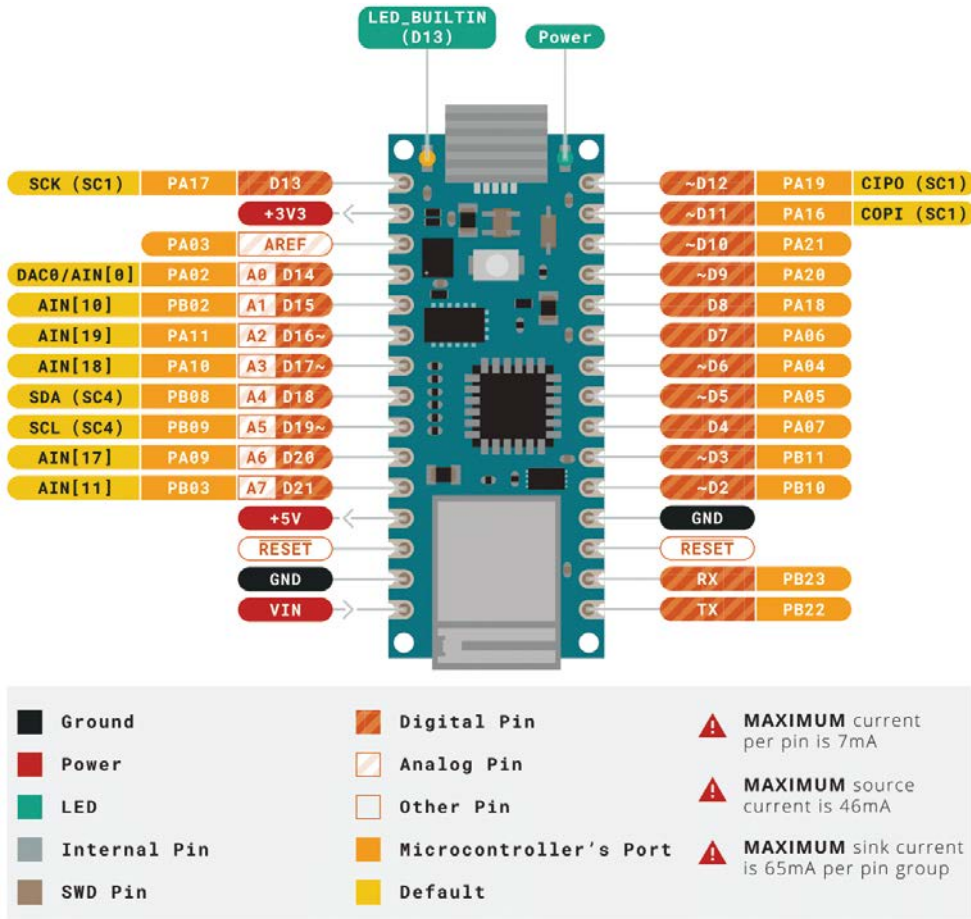


Figure 7-1: Arduino Nano 33 IoT analog and digital pins.

Next, Figure 7-2 shows the communication and interrupt pins.

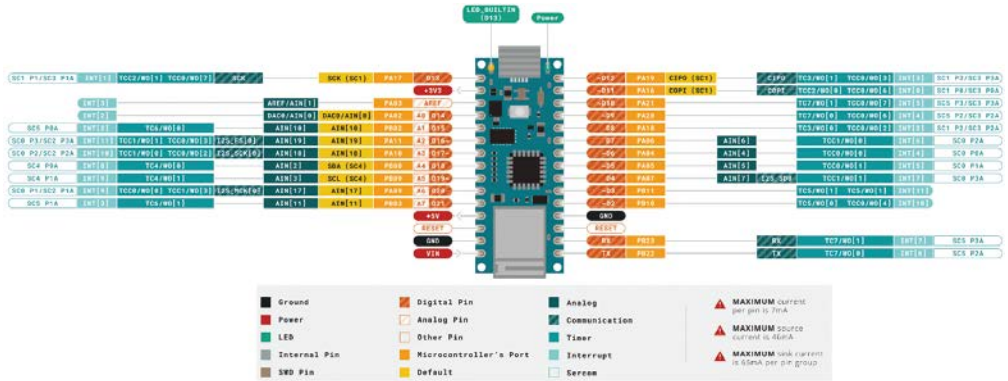


Figure 7-2: Arduino Nano 33 IoT communication and Interrupt.

Lastly, Figure 7-3 shows the bottom of the pins.

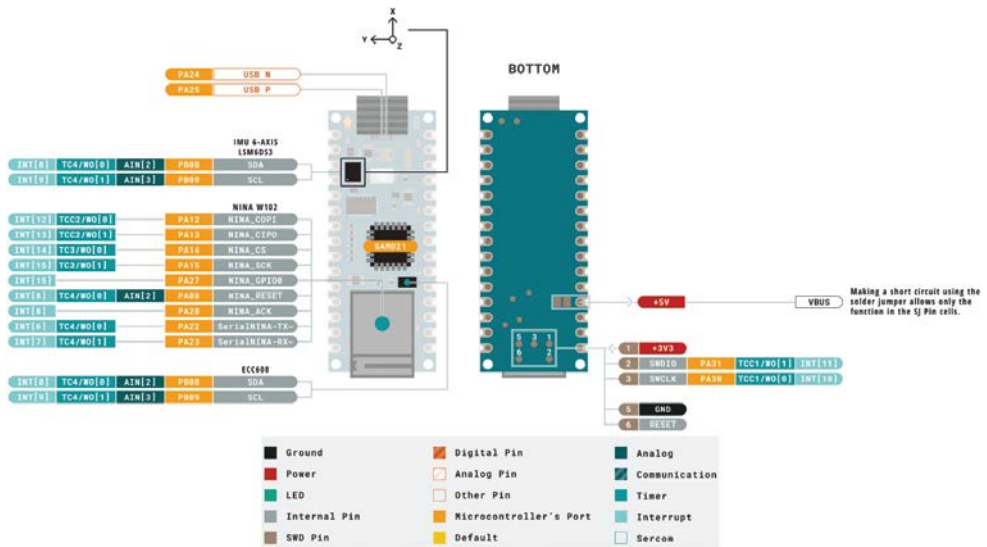


Figure 7-3: Arduino Nano 33 IoT bottom view pinout.

Getting Started

By default, the Arduino IDE does not support the Arduino Nano 33 IoT board, so you have to install the support for this board. Open the **Boards Manager** from the **Tools** in the menu and type in the search bar the string: **samd**. It will return the search results depicted in Figure 7-4.



Figure 7-4: Installing, the support for SAMD board.

Install the option that reads **Arduino SAMD Boards (32-bits ARM Cortex-M0+)**. It will take some time as it is a big package. Once the installation is done, you can choose the board by navigating as shown in Figure 7-5.

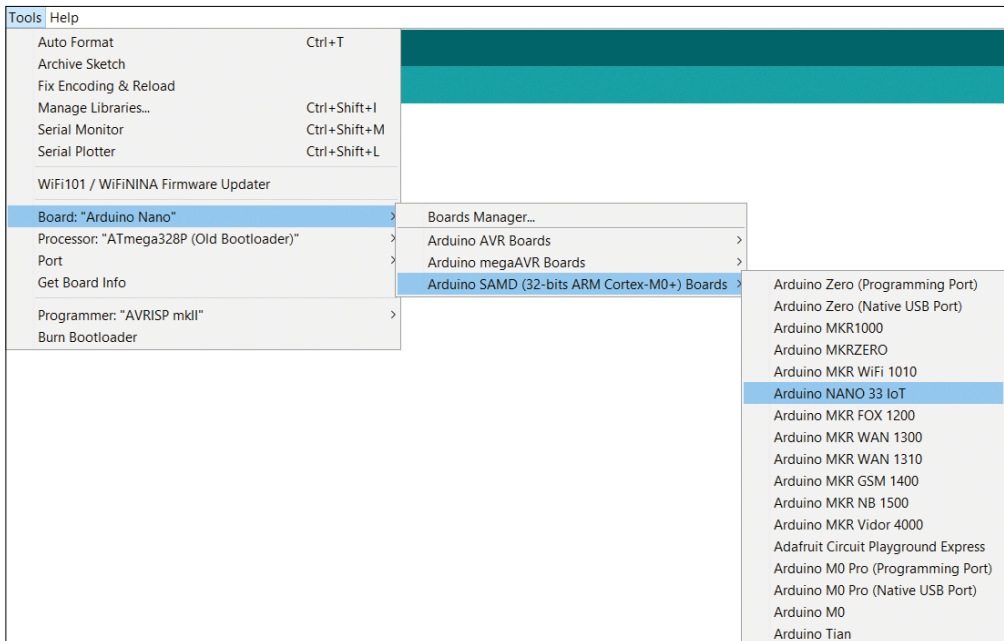


Figure 7-5: Choosing the board.

Connect the board to the computer and then choose the port as pictured in Figure 7-6.

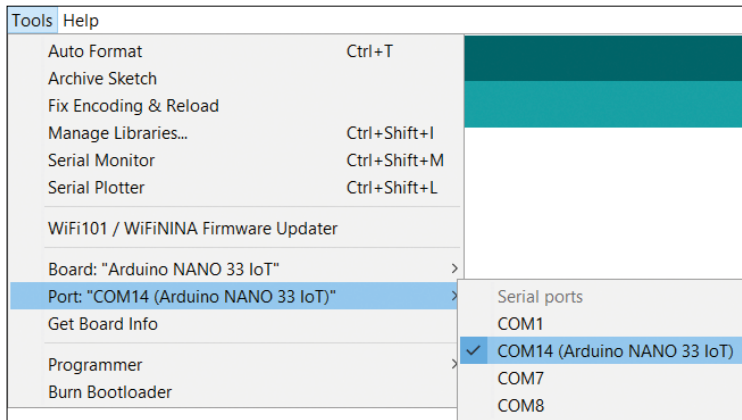


Figure 7-6: Choosing the right communication port on your PC.

Now try to upload a simple program to blink the built-in LED and to check if everything is working fine. In principle, you can achieve all the functionality of the Arduino Nano and the Arduino Nano Every with the Nano 33 IoT.

Working with WiFinina library

We will now install **WiFinina** library. Open the **Library Manager** and search for the string **WiFinina** as shown in Figure 7-7.

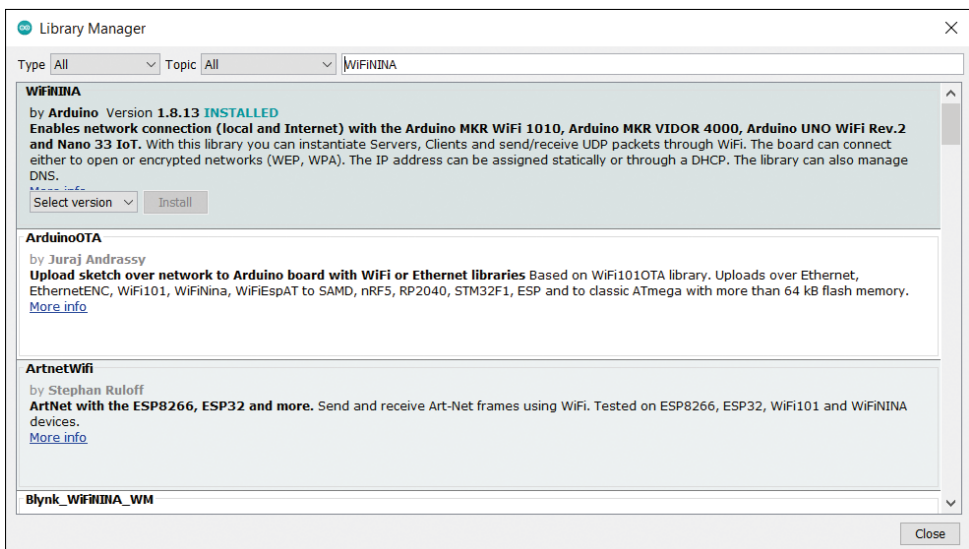


Figure 7-7: Installing the WiFinina library.

Before proceeding any further, I recommend updating the firmware for Wi-Fi with the built-in tool. Open the firmware updater tool as shown in Figure 7-8.

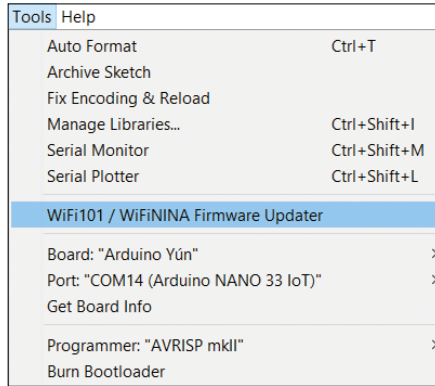


Figure 7-8: Opening the WiFinINA Firmware Updater utility.

For Arduino IDE 1.8, this pops up a window like in Figure 7-9.

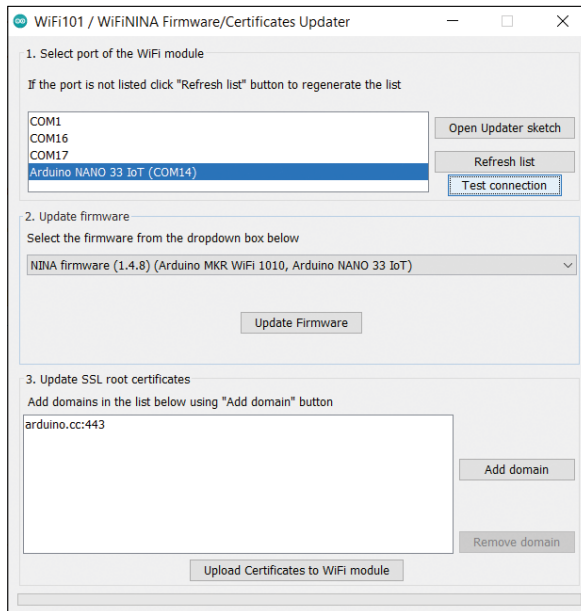


Figure 7-9: WiFinINA Firmware Updater in Arduino IDE 1.8.

Select the port and the firmware version and then click on the button **Update Firmware**. In IDE 2.0 RC5, you will find the utility window like in Figure 7-10.

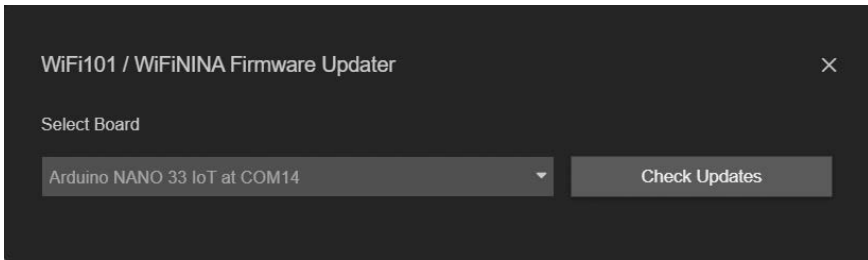


Figure 7-10: Wi-FiNINA Firmware Updater in Arduino IDE 2.0 RC5.

Click on the button **Check Updates**. The window will expand as shown in Figure 7-11.

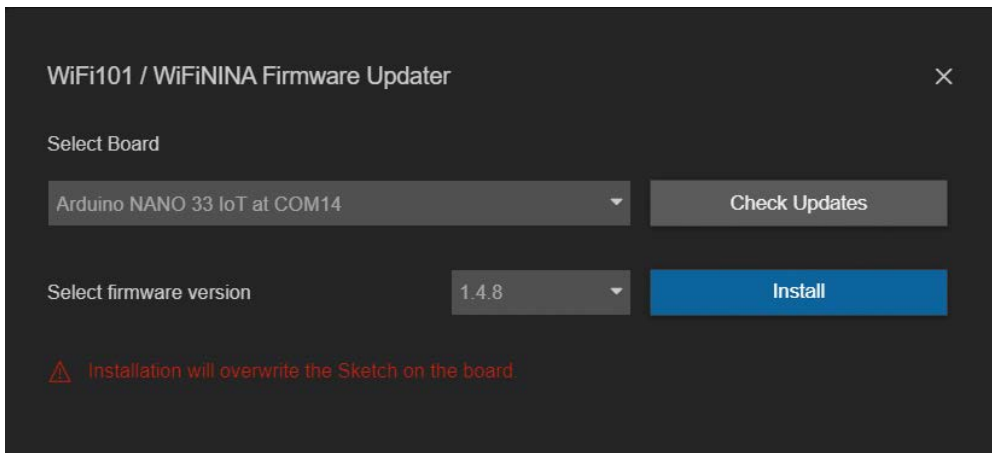


Figure 7-11: Expanded Wi-FiNINA Firmware Updater running in Arduino IDE 2.0 RC5.

Select the firmware version and click the button **Install**. This will update the firmware.

Note: For many sketches, I am referring to the examples provided on the library's home-page at <https://github.com/arduino-libraries/WiFiNINA/> with modifications.

After all this installation stuff, now let's write a simple program to check all the available connections. Let's import the libraries first:

```
#include <SPI.h>
#include <WiFiNINA.h>
```

Lets' check if the Wi-Fi module is working properly in the **setup()** section as follows:

```
void setup()
{
  Serial.begin(9600);
  if (WiFi.status() == WL_NO_MODULE)
  {
```

```

    Serial.println("Communication with WiFi module failed!");
    while (true);
  }
}

```

In the **loop()** section, let's scan for the available Wi-Fi networks, like so:

```

void loop()
{
  Serial.println("\nScanning for available Networks...");
  int num = WiFi.scanNetworks();

```

If no network is found, it returns **-1**, else it returns the number of networks. Let's handle both possibilities:

```

if (num == -1)
{
  Serial.println("Couldn't find a WiFi network");
  while (true);
}
Serial.print("Number of available networks: ");
Serial.println(num);

```

Finally, let's print the details of the available networks,

```

for (int i = 0; i < num; i++)
{
  Serial.print(i);
  Serial.print(" WiFi Network Name: ");
  Serial.print(WiFi.SSID(i));
  Serial.print("\nSignal Strength: ");
  Serial.print(WiFi.RSSI(i));
  Serial.print(" dBm");
  Serial.print("\nMethod of Encryption: ");
  switch (WiFi.encryptionType(i))
  {
    case ENC_TYPE_WEP:
      Serial.println("WEP");
      break;
    case ENC_TYPE_TKIP:
      Serial.println("WPA");
      break;
    case ENC_TYPE_CCMP:
      Serial.println("WPA2");
      break;
    case ENC_TYPE_NONE:
      Serial.println("None");
      break;

```



```

    case ENC_TYPE_AUTO:
        Serial.println("Auto");
        break;
    case ENC_TYPE_UNKNOWN:
    default:
        Serial.println("Unknown");
        break;
    }
}
delay(10000);
}

```

Putting all these snippets together, we have the following sketch:

```

prog00.ino
#include <SPI.h>
#include <WiFiNINA.h>
void setup()
{
    Serial.begin(9600);
    if (WiFi.status() == WL_NO_MODULE)
    {
        Serial.println("Communication with WiFi module failed!");
        while (true);
    }
}
void loop()
{
    Serial.println("\nScanning for available Networks...");
    int num = WiFi.scanNetworks();
    if (num == -1)
    {
        Serial.println("Couldn't find a WiFi network");
        while (true);
    }
    Serial.print("Number of available networks: ");
    Serial.println(num);
    for (int i = 0; i < num; i++)
    {
        Serial.print(i);
        Serial.print(" WiFi Network Name: ");
        Serial.print(WiFi.SSID(i));
        Serial.print("\nSignal Strength: ");
        Serial.print(WiFi.RSSI(i));
        Serial.print(" dBm");
        Serial.print("\nMethod of Encryption: ");
    }
}

```

```
switch (WiFi.encryptionType(i))
{
  case ENC_TYPE_WEP:
    Serial.println("WEP");
    break;
  case ENC_TYPE_TKIP:
    Serial.println("WPA");
    break;
  case ENC_TYPE_CCMP:
    Serial.println("WPA2");
    break;
  case ENC_TYPE_NONE:
    Serial.println("None");
    break;
  case ENC_TYPE_AUTO:
    Serial.println("Auto");
    break;
  case ENC_TYPE_UNKNOWN:
  default:
    Serial.println("Unknown");
    break;
}
}
delay(10000);
}
```

The sample output is as follows:

Scanning for available Networks....Number of available networks: 2

0) Wi-Fi Network Name: TP-Link_710E

Signal Strength: -57 dBm

Method of Encryption: WPA2

1) Wi-Fi Network Name: PanditHome

Signal Strength: -93 dBm

Method of Encryption: WPA2

Let's write another program to create a simple web server on the Nano 33 IoT step-by-step. First, import the libraries:

```
#include <SPI.h>
#include <WiFiNINA.h>
```

Store the SSID and the password in two variables:

```
char ssid[] = "TP-Link_710E";
char pass[] = "internet1";
```

Change the value of these variables with the credentials of your own Wi-Fi network. Let's define variables for status, server, and client.

```
int status = WL_IDLE_STATUS;
WiFiServer server(80);
WiFiClient client;
```

In the **setup()** section, initialize the serial and built-in LED pin to the output mode,

```
void setup()
{
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
```

Check for the Wi-Fi module:

```
if (WiFi.status() == WL_NO_MODULE)
{
  Serial.println("Communication with WiFi module failed!");
  while (true);
}
```

Let's write a polling loop to connect with the Wi-Fi:

```
while (status != WL_CONNECTED)
{
  Serial.print("Attempting to connect to Network named: ");
  Serial.println(ssid);
  status = WiFi.begin(ssid, pass);
  delay(10000);
}
```

Now, initialize the server and print the details of the connection:

```
server.begin();
Serial.print("SSID: ");
Serial.println(WiFi.SSID());
IPAddress ip = WiFi.localIP();
```

```

Serial.println("Started the web server on port 80.");
Serial.print("IP Address: ");
Serial.println(ip);
long rssi = WiFi.RSSI();
Serial.print("Signal strength (RSSI):");
Serial.print(rssi);
Serial.println(" dBm");
Serial.print("Open this address in a browser http://");
Serial.println(ip);
}

```

Proceed by writing the **loop()** section. Here you need to check if a client is connected to the server. The following code creates a simple webpage with two clickable links, which, when clicked, send either **H** or **L** to the server. Based on these values, you are turning the on-board LED off or on.

```

void loop()
{
  client = server.available();
  delay(1000);
  if (client)
  {
    Serial.println("A new client has connected to the server...");
    String currentLine = "";
    while (client.connected())
    {
      if (client.available())
      {
        {
          char c = client.read();
          Serial.write(c);
          if (c == '\n')
          {
            if (currentLine.length() == 0)
            {
              client.println("HTTP/1.1 200 OK");
              client.println("Content-type:text/html");
              client.println();
              client.print("Click <a href=\"/H\">here</a> turn the Built-in LED on
pin 13 on<br>");
              client.print("Click <a href=\"/L\">here</a> turn the Built-in LED on
pin 13 off<br>");
              client.println();
              break;
            }
          }
          else
          {

```

```
        currentLine = "";
    }
}
else if (c != '\r')
{
    currentLine += c;
}
if (currentLine.endsWith("GET /H")) {
    digitalWrite(LED_BUILTIN, HIGH);
}
if (currentLine.endsWith("GET /L")) {
    digitalWrite(LED_BUILTIN, LOW);
}
}
}
client.stop();
Serial.println("The client disconnected...");
}
}
```

Now, put all these sections together and save that file as **prog01.ino**. You will find this file in the code bundle released for this publication. Upload the sketch and the Console shows the following output:

SSID: TP-Link_710E

Started the web server on port 80.

IP Address: 192.168.0.105

Signal strength (RSSI): -48 dBm

Open this address in a browser <http://192.168.0.105>

As you can see, the Wi-Fi network automatically and dynamically assigns an IP address to the Nano using the DHCP protocol. You can check for this IP address in the active client table of your Wi-Fi router or using the **nmap** command (<https://nmap.org/>). Figure 7-12 shows the output in the Zenmap utility.

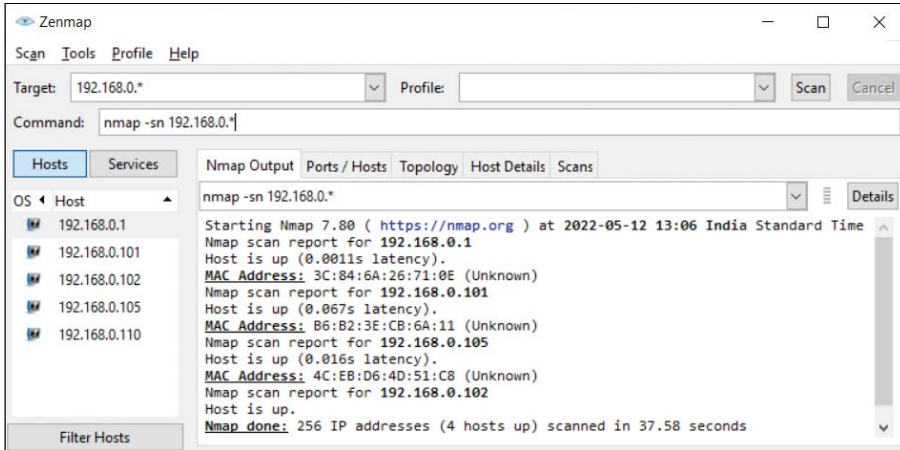


Figure 7-12: Output resulting from the **NMAP** command.

Open a browser window and use the given IP address. You should see the webpage data as in Figure 7-13.



Figure 7-13: Webpage data gleaned with the aid of a browser.

Once you connect using a browser, the Serial Console of Arduino shows the following details of the connections:

Connection: keep-alive

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.54 Safari/537.36

DNT: 1

Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8

Referer: http://192.168.0.105/

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.9

If you click on the hyperlinks, you can send appropriate data to the web server and change the state of the on-board LED. It takes a few seconds for the changes to take effect. In case you are closely observing, you can spot the changed URL in the browser's address bar.

You can connect a relay to Digital I/O pin 13 in order to control a fluorescent tube or a fan. Be careful while working with AC powerline voltages — an electric shock is fatal in most cases.

You can extend this application with a common-cathode RGB LED. Have a look at the circuit depicted in Figure 7-14.

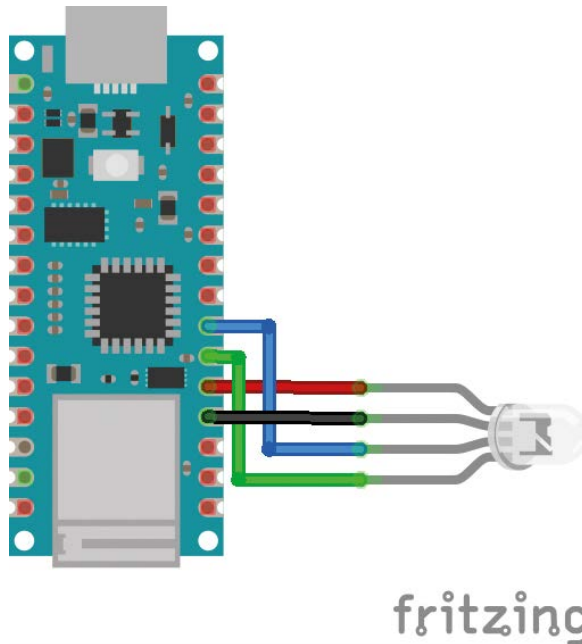


Figure 7-14: Common-cathode RGB LED driving.

You can enable its operation over the local network with the following program:

```
prog02.ino
#include <SPI.h>
#include <WiFiNINA.h>
char ssid[] = "TP-Link_710E";
char pass[] = "internet1";
int status = WL_IDLE_STATUS;
WiFiServer server(80);
WiFiClient client;

const int RED = 2;
const int GREEN = 3;
```

```
const int BLUE = 4;
void setup()
{
  Serial.begin(9600);
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(BLUE, OUTPUT);
  if (WiFi.status() == WL_NO_MODULE)
  {
    Serial.println("Communication with WiFi module failed!");
    while (true);
  }
  while (status != WL_CONNECTED)
  {
    Serial.print("Attempting to connect to Network named: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);
    delay(10000);
  }
  server.begin();
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());
  IPAddress ip = WiFi.localIP();
  Serial.println("Started the web server on port 80.");
  Serial.print("IP Address: ");
  Serial.println(ip);
  long rssi = WiFi.RSSI();
  Serial.print("Signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
  Serial.print("Open this address in a browser http://");
  Serial.println(ip);
}
void loop()
{
  client = server.available();
  delay(1000);
  if (client) {
    Serial.println("A new client has connected to the server...");
    String currentLine = "";
    while (client.connected())
    {
      if (client.available())
      {
        char c = client.read();
        Serial.write(c);
      }
    }
  }
}
```



```
if (c == '\n') {
  if (currentLine.length() == 0)
  {
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println();
    client.print("Click <a href=\"/HR\">here</a> turn the Red LED on pin
2 on<br>");
    client.print("Click <a href=\"/LR\">here</a> turn the Red LED on pin
2 off<br>");
    client.print("Click <a href=\"/HG\">here</a> turn the Green LED on
pin 3 on<br>");
    client.print("Click <a href=\"/LG\">here</a> turn the Green LED on
pin 3 off<br>");
    client.print("Click <a href=\"/HB\">here</a> turn the Blue LED on pin
4 on<br>");
    client.print("Click <a href=\"/LB\">here</a> turn the Blue LED on pin
4 off<br>");
    client.println();
    break;
  }
  else
  {
    currentLine = "";
  }
} else if (c != '\r') {
  currentLine += c;
}
if (currentLine.endsWith("GET /HR"))
{
  digitalWrite(RED, HIGH);
}
if (currentLine.endsWith("GET /LR"))
{
  digitalWrite(RED, LOW);
}
if (currentLine.endsWith("GET /HG"))
{
  digitalWrite(GREEN, HIGH);
}
if (currentLine.endsWith("GET /LG"))
{
  digitalWrite(GREEN, LOW);
}
if (currentLine.endsWith("GET /HB"))
{
```

```
        digitalWrite(BLUE, HIGH);
    }
    if (currentLine.endsWith("GET /LB"))
    {
        digitalWrite(BLUE, LOW);
    }
}
}
client.stop();
Serial.println("The client disconnected...");
}
}
```

Run the program and control the RGB LED using a browser. Now, you can extend this by connecting a relay board with multiple relays and control your home appliances over the web.

A Telnet-based Group Chat Server

You can use Telnet protocol to create a simple text-based group chat server. Simply modify the earlier example to use **port 23** (which is the default port for the Telnet application).

```
prog03.ino
#include <SPI.h>
#include <WiFiNINA.h>
char ssid[] = "TP-Link_710E";
char pass[] = "internet1";
int status = WL_IDLE_STATUS;
WiFiServer server(23);
boolean alreadyConnected = false;
WiFiClient client;
void setup()
{
    Serial.begin(9600);
    if (WiFi.status() == WL_NO_MODULE)
    {
        Serial.println("Communication with WiFi module failed!");
        while (true);
    }
    while (status != WL_CONNECTED)
    {
        Serial.print("Attempting to connect to Network named: ");
        Serial.println(ssid);
        status = WiFi.begin(ssid, pass);
        delay(10000);
    }
    server.begin();
}
```

```
Serial.print("SSID: ");
Serial.println(WiFi.SSID());
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");
Serial.println(ip);
long rssi = WiFi.RSSI();
Serial.print("Signal strength (RSSI):");
Serial.print(rssi);
Serial.println(" dBm");
Serial.print("Open this address in a browser http://");
Serial.println(ip);
}
void loop()
{
  client = server.available();
  if (client)
  {
    if (!alreadyConnected)
    {
      client.flush();
      Serial.println("We have a new client");
      client.println("Hello, client!");
      alreadyConnected = true;
    }
    if (client.available() > 0)
    {
      char thisChar = client.read();
      server.write(thisChar);
      Serial.write(thisChar);
    }
  }
}
```

Upload this sketch. This program listens for incoming connections and broadcasts messages to all clients. Once the sketch is uploaded, you can see the IP address of the Nano in your Serial Console. From the command prompt of Linux and macOS, run the following command:

```
telnet 192.168.0.105 23
```

This will show the following message in the Console:

Trying 192.168.0.105...

Connected to 192.168.0.105.

Escape character is '^'.

You can type in text and it will be broadcasted everywhere,

test

Hello, client!

test

test1

abc

You can exit with the following command:

Ctrl +]

You have to enable Telnet in Windows. Here's a good guide for that:

<https://www.lifewire.com/what-is-telnet-2626026>

Pinging a Remote Server

It's interesting and fun to ping a remote server as follows:

```
prog04.ino
#include <SPI.h>
#include <WiFiNINA.h>
char ssid[] = "TP-Link_710E";
char pass[] = "internet1";
int status = WL_IDLE_STATUS;
const String hostName = "www.google.com";
int pingResult;
void setup()
{
  Serial.begin(9600);
  if (WiFi.status() == WL_NO_MODULE)
  {
    Serial.println("Communication with WiFi module failed!");
    while (true);
  }
  while (status != WL_CONNECTED)
  {
    Serial.print("Attempting to connect to Network named: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);
```

```
    delay(10000);
  }
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  long rssi = WiFi.RSSI();
  Serial.print("Signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
  Serial.println(ip);
}
void loop()
{
  Serial.print("Pinging ");
  Serial.print(hostName);
  Serial.print(": ");
  pingResult = WiFi.ping(hostName);
  if (pingResult >= 0)
  {
    Serial.print("SUCCESS! RTT = ");
    Serial.print(pingResult);
    Serial.println(" ms");
  }
  else
  {
    Serial.print("FAILED! Error code: ");
    Serial.println(pingResult);
  }
  delay(5000);
}
```

Upload this sketch. This is a typical output:

IP Address: 192.168.0.105

Signal strength (RSSI): -51 dBm

192.168.0.105

Pinging www.google.com: SUCCESS! RTT = 10 ms

Pinging www.google.com: SUCCESS! RTT = 10 ms

A simple Web Client

In addition to pinging, you can write a sketch for a simple web-based client to retrieve information from a host. In the following sketch, you are retrieving the search results for the term **elektor** from the search engine www.google.com.

prog05.ino

```
#include <SPI.h>
#include <WiFiNINA.h>
char ssid[] = "TP-Link_710E";
char pass[] = "internet1";
int status = WL_IDLE_STATUS;
char server[] = "www.google.com";
WiFiClient client;
void setup()
{
  Serial.begin(9600);
  if (WiFi.status() == WL_NO_MODULE)
  {
    Serial.println("Communication with WiFi module failed!");
    while (true);
  }
  while (status != WL_CONNECTED)
  {
    Serial.print("Attempting to connect to Network named: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);
    delay(10000);
  }
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  long rssi = WiFi.RSSI();
  Serial.print("Signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
  Serial.println(ip);
  Serial.println("\nStarting connection to server...");
  if (client.connect(server, 80))
  {
    Serial.println("connected to server");
    client.println("GET /search?q=elektor HTTP/1.1");
    client.println("Host: www.google.com");
    client.println("Connection: close");
    client.println();
  }
}
```

```

    }
}
void loop()
{
    while (client.available())
    {
        char c = client.read();
        Serial.write(c);
    }
    if (!client.connected())
    {
        Serial.println();
        Serial.println("Disconnecting from server.");
        client.stop();
        while (true);
    }
}
}

```

Using the Serial Console, all the results returned by the remote server will get printed.

Working with a Real-Time Clock

The Arduino Nano 33 IoT has a built-in real-time clock (RTC) that can be synchronized over the Internet. Let's see how to use that clock. You have to synchronize the on-board clock by using the NTP (Network Time Protocol). For that, install the **RTCZero** library from the **Library Manager** tool.

Let's start by importing libraries:

```

#include "SPI.h"
#include <WiFiNINA.h>
#include <WiFiUdp.h>
#include <RTCZero.h>

```

Let's declare the variables for Wi-Fi:

```

char ssid[] = "TP-Link_710E";
char pass[] = "internet1";
int status = WL_IDLE_STATUS;
Let's declare the variables for the Real-time clock,
RTCZero rtc;
unsigned long epoch;
uint8_t attempts = 0;
const uint8_t maxAttempts = 7;
Let's initialize the WiFi,
void setup()
{

```

```
Serial.begin(9600);
if (WiFi.status() == WL_NO_MODULE)
{
  Serial.println("Communication with WiFi module failed!");
  while (true);
}
while (status != WL_CONNECTED)
{
  Serial.print("Attempting to connect to Network named: ");
  Serial.println(ssid);
  status = WiFi.begin(ssid, pass);
  delay(10000);
}
Serial.print("SSID: ");
Serial.println(WiFi.SSID());
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");
Serial.println(ip);
long rssi = WiFi.RSSI();
Serial.print("Signal strength (RSSI):");
Serial.print(rssi);
Serial.println(" dBm");
Serial.println(ip);
Let's initialize the real-time clock,
rtc.begin();
```

Let's fetch the epoch (i.e., the time since 1 January 1970) from an NTP server:

```
do
{
  epoch = WiFi.getTime();
  attempts = attempts + 1;
}
while ((epoch == 0) && (attempts < maxAttempts));
```

If the NTP Server cannot be connected, you get an error. Else, set the current time:

```
if (attempts == maxAttempts)
{
  Serial.print("NTP is unreachable!");
  while (1);
}
else
{
  Serial.print("Epoch received: ");
  Serial.println(epoch);
}
```



```
    rtc.setEpoch(epoch);  
    Serial.println();  
  }  
}
```

Finally, in the **loop()** section, you are displaying the current date and time every second.

```
void loop()  
{  
  Serial.print(rtc.getDay());  
  Serial.print("/");  
  Serial.print(rtc.getMonth());  
  Serial.print("/");  
  Serial.print(rtc.getYear());  
  Serial.print(" ");  
  Serial.print(rtc.getHours());  
  Serial.print(":");  
  Serial.print(rtc.getMinutes());  
  Serial.print(":");  
  Serial.print(rtc.getSeconds());  
  Serial.println();  
  delay(1000);  
}
```

Upload the sketch and observe the output on the Serial Console.

Using the DS18B20 Temperature Sensor Jointly with the RTC

In the previous chapter, we learned to connect the DS18B20 temperature sensor to a Nano board. You can use the same connections for the Nano 33 IoT, as shown in Figure 7-15.

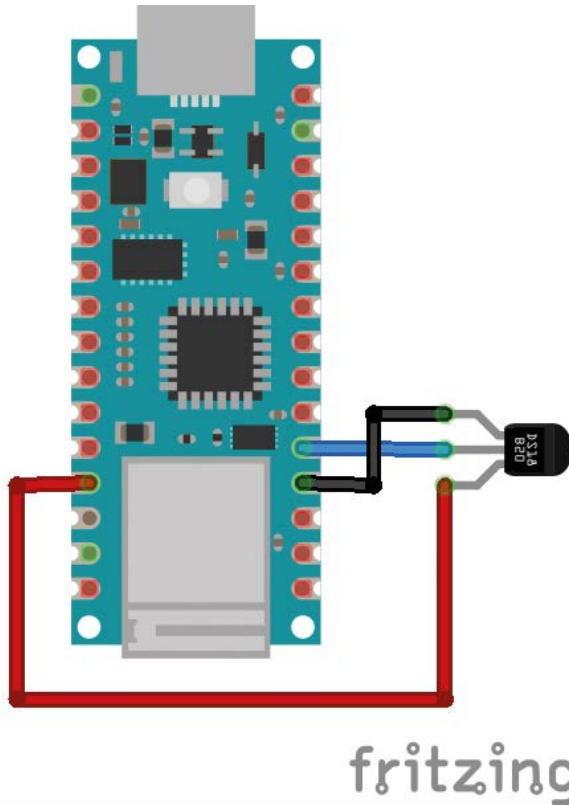


Figure 7-15: DS18B20 temperature sensor connected to the Arduino Nano 33 IoT.

You can mix the code of the RTC with the code of the temperature sensor to get a temperature reading with a timestamp, as follows:

```
prog07.ino
#include "SPI.h"
#include <WiFiNINA.h>
#include <WiFiUdp.h>
#include <RTCZero.h>
#include <OneWire.h>
#include <DallasTemperature.h>
const int sensor_pin = 2;
OneWire oneWire(sensor_pin);
DallasTemperature sensors(&oneWire);
char ssid[] = "TP-Link_710E";
char pass[] = "internet1";
int status = WL_IDLE_STATUS;
char msg[100];
RTCZero rtc;
unsigned long epoch;
```

```
uint8_t attempts = 0;
const uint8_t maxAttempts = 7;
void setup()
{
  Serial.begin(9600);
  sensors.begin();
  if (WiFi.status() == WL_NO_MODULE)
  {
    Serial.println("Communication with WiFi module failed!");
    while (true);
  }
  while (status != WL_CONNECTED)
  {
    Serial.print("Attempting to connect to Network named: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);
    delay(10000);
  }
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  long rssi = WiFi.RSSI();
  Serial.print("Signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
  Serial.println(ip);
  rtc.begin();
  do
  {
    epoch = WiFi.getTime();
    attempts = attempts + 1;
  }
  while ((epoch == 0) && (attempts < maxAttempts));
  if (attempts == maxAttempts)
  {
    Serial.print("NTP is unreachable!");
    while (1);
  }
  else
  {
    Serial.print("Epoch received: ");
    Serial.println(epoch);
    rtc.setEpoch(epoch);
    Serial.println();
  }
}
```

```
    }  
}  
  
void loop()  
{  
    sensors.requestTemperatures();  
    float temp = sensors.getTempCByIndex(0);  
    sprintf(msg, "%d/%d/%d %d:%d:%d ",  
        rtc.getDay(), rtc.getMonth(),  
        rtc.getYear(), rtc.getHours(),  
        rtc.getMinutes(), rtc.getSeconds());  
    Serial.print(msg);  
    Serial.print(temp);  
    Serial.println();  
    delay(1000);  
}
```

This will print the temperature with the current timestamp in the Serial Console at seconds intervals. You can even display the results on a webpage. Check the following sketch:

```
prog08.ino  
#include "SPI.h"  
#include <WiFiNINA.h>  
#include <WiFiUdp.h>  
#include <RTCZero.h>  
#include <OneWire.h>  
#include <DallasTemperature.h>  
const int sensor_pin = 2;  
OneWire oneWire(sensor_pin);  
DallasTemperature sensors(&oneWire);  
char ssid[] = "TP-Link_710E";  
char pass[] = "internet1";  
int status = WL_IDLE_STATUS;  
char msg[100];  
RTCZero rtc;  
unsigned long epoch;  
uint8_t attempts = 0;  
const uint8_t maxAttempts = 7;  
WiFiServer server(80);  
WiFiClient client;  
void setup()  
{  
    Serial.begin(9600);  
    sensors.begin();  
    if (WiFi.status() == WL_NO_MODULE)  
    {
```

```
    Serial.println("Communication with WiFi module failed!");
    while (true);
}
while (status != WL_CONNECTED)
{
    Serial.print("Attempting to connect to Network named: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);
    delay(10000);
}
server.begin();
Serial.print("SSID: ");
Serial.println(WiFi.SSID());
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");
Serial.println(ip);
long rssi = WiFi.RSSI();
Serial.print("Signal strength (RSSI):");
Serial.print(rssi);
Serial.println(" dBm");
Serial.println(ip);
rtc.begin();
do
{
    epoch = WiFi.getTime();
    attempts = attempts + 1;
}
while ((epoch == 0) && (attempts < maxAttempts));
if (attempts == maxAttempts)
{
    Serial.print("NTP is unreachable!");
    while (1);
}
else
{
    Serial.print("Epoch received: ");
    Serial.println(epoch);
    rtc.setEpoch(epoch);
    Serial.println();
}
}
void loop()
{
    sensors.requestTemperatures();
    float temp = sensors.getTempCByIndex(0);
    sprintf(msg, "%d/%d/%d %d:%d:%d %s",
```

```
rtc.getDay(), rtc.getMonth(),
rtc.getYear(), rtc.getHours(),
rtc.getMinutes(), rtc.getSeconds(),
String(temp, 3).c_str());
client = server.available();
delay(1000);
if (client)
{
  Serial.println("A new client has connected to the server...");
  String currentLine = "";
  while (client.connected())
  {
    if (client.available())
    {
      char c = client.read();
      Serial.write(c);
      if (c == '\n') {
        if (currentLine.length() == 0)
        {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-type:text/html");
          client.println("Connection: close");
          client.println();
          client.println("<!DOCTYPE html>");
          client.println("<html>");
          client.println("<head>");
          client.println("<title>Arduino Webserver</title>");
          client.println("<meta http-equiv=\"refresh\" content=\"1\">");
          client.println("</head>");
          client.println("<body>");
          client.println(msg);
          client.println("</body>");
          client.println("</html>");
          client.println();
          break;
        }
      }
      else
      {
        currentLine = "";
      }
    } else if (c != '\r') {
      currentLine += c;
    }
  }
}
// close the connection:
```

```

client.stop();
Serial.println("The client disconnected...");
}
delay(1000);
}

```

Just copy the IP address from the Serial Console and paste it into the browser to see the temperature.

Visualizing the Temperature graph with ThingSpeak

ThingSpeak is an online platform for data visualization by **MathWorks**. Let's visualize the temperature data using ThingSpeak. First, you need to create an account at <https://thingspeak.com/>. Then log into the account and from the menu bar on the webpage, navigate to the option **My Channels** from the Menu, as shown in Figure 7-16.



Figure 7-16: Navigation to **My Channels** in ThingSpeak.

Once you click this option, you will be taken to a new page. There, click on the green button **New Channel** — check Figure 7-17.

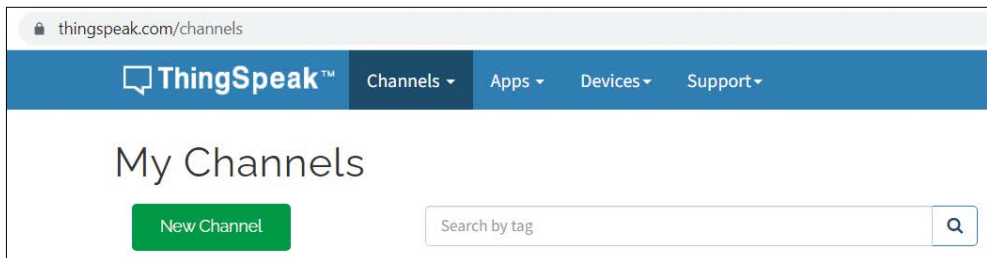


Figure 7-17: **My Channels** page.

Once you click the button, a form will open as pictured in Figure 7-18.

Figure 7-18: Creating a New Channel.

Assign a suitable name to your channel. Since you just need to show the graph for temperature, check only one field and assign a suitable name to it. Complete the description and metadata fields. Finally, click on the **Save Channel** button at the bottom of the page. Once done, your channels list will appear like the one shown in Figure 7-19.

Name	Created	Updated
Temperature	2022-05-13	2022-05-13 12:15

Figure 7-19: List of channels.

Now, you have to change the settings. Go to the **Sharing** section and share the channel with everyone, as shown in Figure 7-20.

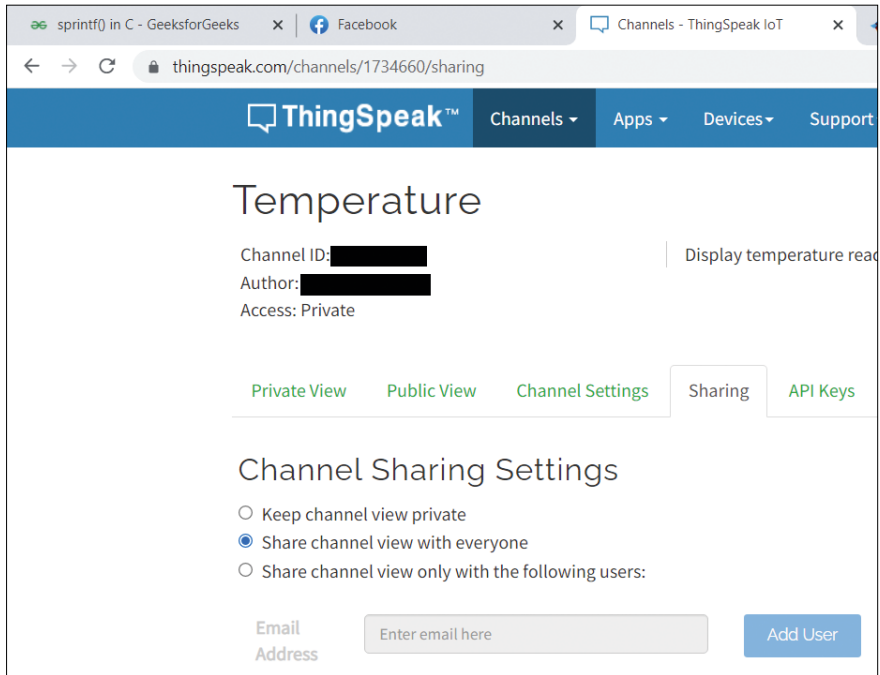


Figure 7-20: Share the channel with everyone!

Afterward, navigate to the tab **API Keys**, and note down the **Write API Key** for your channel.

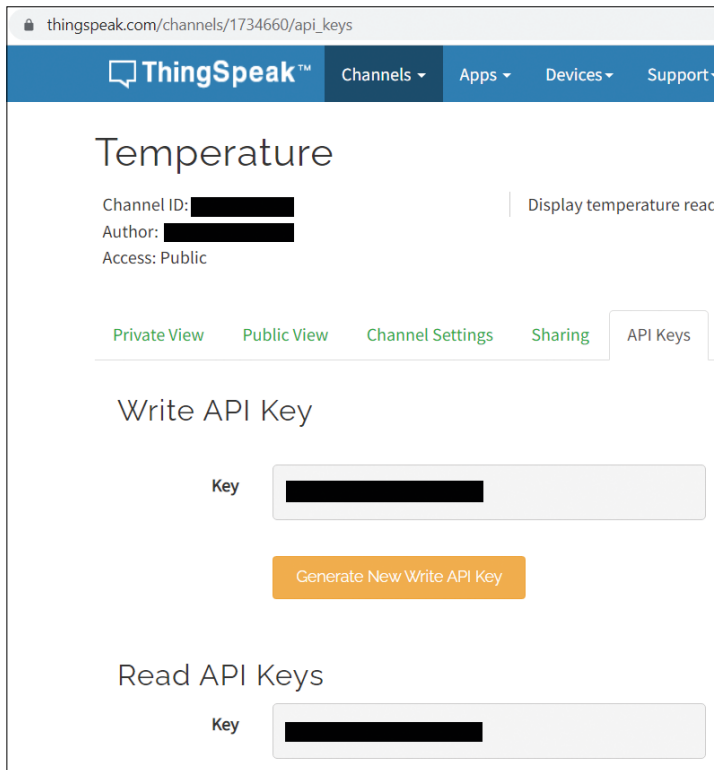


Figure 7-21: Writing the API Key.

Now, install the **ThingSpeak** library from the **Library Manager**. Upload the following sketch to the Nano 33 IoT:

```
prog09.ino
#include "SPI.h"
#include <WiFiNINA.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include "ThingSpeak.h"
unsigned long mychannelnumber = 1111111;
const char * myAPIkey = "A1A1A1A1A1A1A1A1";
const int sensor_pin = 2;
OneWire oneWire(sensor_pin);
DallasTemperature sensors(&oneWire);
char ssid[] = "TP-Link_710E";
char pass[] = "internet1";
int status = WL_IDLE_STATUS;
WiFiClient client;
void setup()
{
```

```

Serial.begin(9600);
ThingSpeak.begin(client);
sensors.begin();
if (WiFi.status() == WL_NO_MODULE)
{
  Serial.println("Communication with WiFi module failed!");
  while (true);
}
while (status != WL_CONNECTED)
{
  Serial.print("Attempting to connect to Network named: ");
  Serial.println(ssid);
  status = WiFi.begin(ssid, pass);
  delay(10000);
}
Serial.print("SSID: ");
Serial.println(WiFi.SSID());
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");
Serial.println(ip);
long rssi = WiFi.RSSI();
Serial.print("Signal strength (RSSI):");
Serial.print(rssi);
Serial.println(" dBm");
Serial.println(ip);
}
void loop()
{
  sensors.requestTemperatures();
  float temp = sensors.getTempCByIndex(0);
  ThingSpeak.writeField(mychannelnumber, 1, temp, myAPIkey);
  delay(16000);
}

```

You are already familiar with most of the code. Let's understand the new lines of code. The new file included is as follows,

```
#include "ThingSpeak.h"
```

You need to get the channel number and **API Write Key** from your ThingSpeak account. I have entered dummy data to mask the details of my own account,

```

unsigned long mychannelnumber = 1111111;
const char * myAPIkey = "A1A1A1A1A1A1A1A1";
You have to initialize the ThingSpeak as follows,
  ThingSpeak.begin(client);

```

You can write the data to the first field as follows:

```
ThingSpeak.writeField(mychannelnumber, 1, temp, myAPIkey);
```

The argument **1** refers to the field numbers, and **temp** is the variable for numerical value to be visualized. Upload the sketch and go to your channel using a browser. If the number of your channel is **11111111**, then the URL of your channel is <https://thingspeak.com/channels/11111111>. Everything working properly so far, you can see the graph of all temperature data.

Programming the Built-in IMU

The Arduino Nano 33 IoT comes with a built-in Inertial Measurement Unit (IMU) type **LS-M6DS3**. You can program its accelerometer and gyroscope easily. But first, you need to install the library for the LSM6DS3 by opening the **Library Manager** and installing the library named **Arduino_LSM6DS3**. Next, create a sketch and save it as **prog10.ino**. Let's add code to it in a step-by-step manner. Be sure to include the required library first:

```
#include <Arduino_LSM6DS3.h>
```

In the **setup()** section, you are initializing the IMU and printing the sample rates for the accelerometer and the gyroscope in Hz in the Serial Console.

```
void setup()
{
  Serial.begin(9600);
  if (!IMU.begin())
  {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
  Serial.print("Accelerometer sample rate = ");
  Serial.print(IMU.accelerationSampleRate());
  Serial.println(" Hz");
  Serial.println();
  Serial.print("Gyroscope sample rate = ");
  Serial.print(IMU.gyroscopeSampleRate());
  Serial.println(" Hz");
  Serial.println();
}
```

In the **loop()** section, we are printing the acceleration and orientation values on all three axes, one after another, every second.

```
void loop()
{
  float x, y, z;
  if (IMU.accelerationAvailable())
  {
    Serial.println("Acceleration X, Y, Z");
    IMU.readAcceleration(x, y, z);
    Serial.print(x);
    Serial.print(", ");
    Serial.print(y);
    Serial.print(", ");
    Serial.print(z);
    Serial.print(", \n");
  }
  if (IMU.gyroscopeAvailable())
  {
    Serial.println("Gyroscope X, Y, Z");
    IMU.readGyroscope(x, y, z);
    Serial.print(x);
    Serial.print(", ");
    Serial.print(y);
    Serial.print(", ");
    Serial.print(z);
    Serial.print(", \n");
  }
  delay(1000);
}
```

You can exploit this interesting feature of the Nano 33 IoT board creatively for interactive projects.

Summary

In this chapter, you learned to work with the Wi-Fi module aiming to create simple IoT applications. You have also learned to work with the IMU included on the board. You can extend this knowledge to build more creative and interactive projects in the future.

Conclusion

The journey that you started to explore the Arduino Ecosystem is far from over. There are plenty more Arduino boards, sensors, I/O devices, and projects to explore. You can start exploring the documentation section at <https://www.arduino.cc/> and the project hub at <https://create.arduino.cc/projecthub>. Happy exploring!

Index

28BYJ-48	82, 92	Digital Pins	18
9225	103	digitalRead()	65
9600 baud	74	DIP	46
		DS18B20	154, 179
		DS18B20 Temperature Sensor	151
		Dual in Package	46
A		E	
AccelStepper	98	Ecosystem	13
analog input	71	expansion shields	67
Analog Input	75	F	
analogWrite()	84	Fritzing	57
API Write Key	189	full step	99
Arduino	11	G	
Arduino Clones	14	Geometric Art	103
Arduino Ecosystem	11	global variable	45
Arduino IDE	14	Group Chat Server	172
Arduino Nano	11	H	
Arduino Nano 33 IoT	154	half step	99
Arduino Nano Every	11	I	
Arduino Platform	11	I2C	19, 71, 75
Arduino Serial	73	Ilitek 9225	103
ATmega328	15	IMU	190
ATmega4809	16	IMU LSM6DS3	155
ATMega4809	15	INPUT_PULLUP	65
attach()	91	Inter-Integrated Circuit	75
		I/O expansion shield	69
B		J	
binary counter	63	Joystick	149
Boards Manager	32	Jumper cables	50
breadboard	56	L	
Breadboards	46	LED	43
built-in constants	44	LED blink	42
Buzzer	144	LEDs	51
		Light Emitting Diodes	51
C		local variables	45
CH340	39	loop()	42
CIPO	19, 75	LSM6DS3	190
common-anode	65		
common-cathode	65		
COPI	19, 75		
Counterfeits	14		
D			
Derivatives	14		
Device Manager	36		
Digital I/O	82		

M		Serial Peripheral Interface	75
map()	79	serial plotter	78
MB 102	46	Serial.println()	75
Microchip ATECC608A	155	Servo Motor	89
microcontroller	11	setup()	42
MISO	19	SG90 servo motor	89
MOSI	19	shield	68
motor driver	82	SPI	19, 71, 75
multiple LEDs	59	Star Wars	146
N		T	
Nano Family	13	Telnet	154, 172
Newline	74	temperature sensor	144
nmap	167	TFT LCD	103
O		ThingSpeak	185
OneWire	152	TOFF	83
P		TON	83
parallel	71	Tools	73
Ping	174	trimpot	76
port	37	Two Wire Interface	75
portable power supply	50	U	
potentiometer	75	UART	19
power supplies	46	ULN2003A	82, 92
Pulse Width Modulation	82	Unipolar Stepper Motors	82
pushbutton	63	Update Firmware	160
Pushbuttons	53	Upload	39
PWM	19, 82	V	
R		V	
Raspberry Pi OS	30	Visualizing	185
Real-Time Clock	177	W	
Remote Server	174	web client	154
Resistors	52	Web Client	176
RGB	65	WiFinINA	154, 159
RGB LEDs	42	Wire	75
RTCZero	177	S	
S		samd	157
samd	157	SAMD	158
SAMD	158	Send	74
Send	74	Sensor	144
Sensor	144	serial	71
serial	71	Serial Monitor	73
Serial Monitor	73		

Kickstart to Arduino Nano

Get Cracking with the Arduino Nano V3, Nano Every, and Nano 33 IoT

The seven chapters in this book serve as the first step for novices and microcontroller enthusiasts wishing to make a head start in Arduino programming. The first chapter introduces the Arduino platform, ecosystem, and existing varieties of Arduino Nano boards. It also teaches how to install various tools needed to get started with Arduino Programming. The second chapter kicks off with electronic circuit building and programming around your Arduino. The third chapter explores various buses and analog inputs. In the fourth chapter, you get acquainted with the concept of pulse width modulation (PWM) and working with unipolar stepper motors.

In the fifth chapter, you are sure to learn about creating beautiful graphics and basic but useful animation with the aid of an external display. The sixth chapter introduces the readers to the concept of I/O devices such as sensors and the piezo buzzer, exploring their methods of interfacing and programming with the Arduino Nano. The last chapter explores another member of Arduino Nano family, Arduino Nano 33 IoT with its highly interesting capabilities. This chapter employs and deepens many concepts learned from previous chapters to create interesting applications for the vast world of the Internet of Things.

The entire book follows a step-by-step approach to explain concepts and the operation of things. Each concepts is invariably followed by a to-the-point circuit diagram and code examples. Next come detailed explanations of the syntax and the logic used. By closely following the concepts, you will become comfortable with circuit building, Arduino programming, the workings of the code examples, and the circuit diagrams presented. The book also has plenty of references to external resources wherever needed.

An archive file (.zip) comprising the software examples and Fritzing-style circuit diagrams discussed in the book may be downloaded free of charge from the book's product and resources page on www.elektor.com (search for: book title and author).



Ashwin Pajankar graduated from IIT Hyderabad. With over 25 years of experience in computer programming and electronics, he published over two dozen books on these topics. Currently, he teaches these topics online to over 93,000 students worldwide on Udemy and YouTube. He is proficient in Linux, Shell Scripting, Windows PowerShell, C, C++, BASIC, Assembly Language of 8085 and x86, Java, JavaScript, Python, HTML, and Golang. He has extensively worked on single-board computers such as the Raspberry Pi and Banana Pro, Arduino, BBC Micro:bit, and Raspberry Pi Pico.

Elektor International Media BV
www.elektor.com

