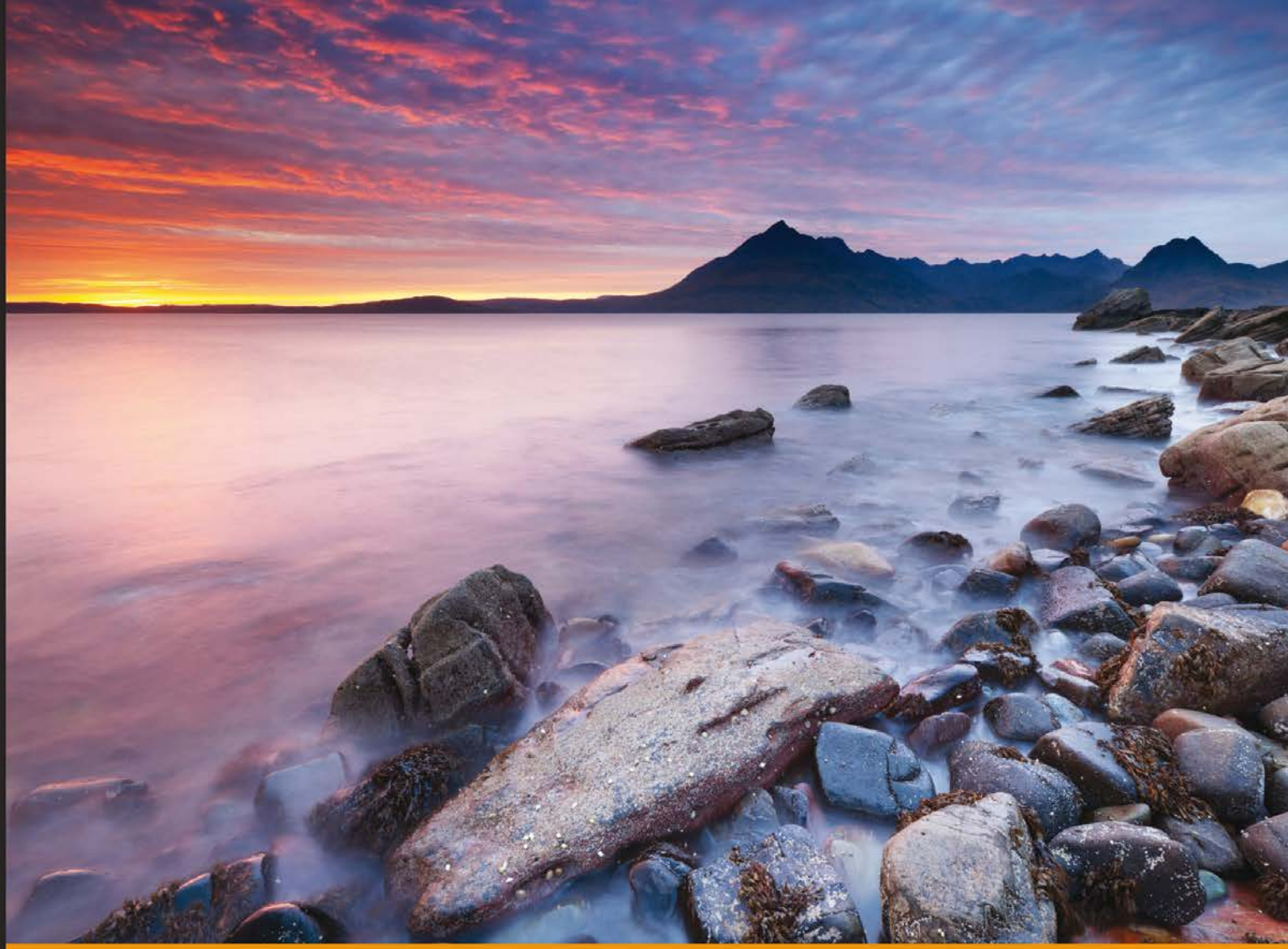


Community Experience Distilled

Programming Arduino with LabVIEW

Build interactive and fun learning projects with Arduino using LabVIEW

Marco Schwartz Oliver Manickum

[PACKT] PUBLISHING

Programming Arduino with LabVIEW

Build interactive and fun learning projects with Arduino using LabVIEW

Marco Schwartz

Oliver Manickum

[PACKT] PUBLISHING

BIRMINGHAM - MUMBAI

Programming Arduino with LabVIEW

First published: January 2015

Production reference: 1210115

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84969-822-1

www.packtpub.com

Credits

Authors

Marco Schwartz

Oliver Manickum

Reviewers

Adith Jagadish Boloor

Aaron Srivastava

Fangzhou Xia

Commissioning Editor

Amarabha Banerjee

Acquisition Editor

Harsha Bharwani

Content Development Editor

Rikshith Shetty

Technical Editor

Bharat Patil

Copy Editor

Karuna Narayanan

Project Coordinator

Sanchita Mandal

Proofreaders

Ameesha Green

Sandra Hopper

Indexer

Rekha Nair

Production Coordinator

Shantanu N. Zagade

Cover Work

Shantanu N. Zagade

About the Authors

Marco Schwartz is an electrical engineer, entrepreneur, and blogger. He has a master's degree in electrical engineering and computer science from SUPELEC in France and a master's degree in micro engineering from the EPFL in Switzerland.

He has more than 5 years of experience working in the domain of electrical engineering. His interests gravitate around electronics, home automation, the Arduino and Raspberry Pi platforms, open source hardware projects, and 3D printing.

He also runs several websites on Arduino, including the `http://www.openhomeautomation.net/` website, which is dedicated to building home automation systems using open source hardware.

He has written another book called *Arduino Home Automation Projects*, *Packt Publishing*, on home automation and Arduino and also published a book called *Internet of Things with the Arduino*, on how to build Internet-of-Things projects with Arduino.

Oliver Manickum has been working in the embedded development scene for almost 20 years. His favorite development platform is Arduino. He has delivered thousands of projects and is a big fan of ATMEL and the Arduino platform. He currently writes high-performance games on mobile platforms; however, developing prototypes with Arduino is his main hobby.

He has also reviewed *Netduino Home Automation Projects*, *Matt Cavanagh*.

I would like to thank my wife, Nazia Osman, for her patience while I was building devices that would sometimes burn down parts of our house, over and over again.

About the Reviewers

Adith Jagadish Boloor is an undergraduate student at the School of Mechanical Engineering at Purdue University, West Lafayette. He was born and brought up in the beautiful coastal city of Mangalore, India. Having lived there for 18 years, he came to the United States of America to pursue his higher education, with the desire to acquire new skills pertaining to the latest technological developments, and with this knowledge, he hopes to revolutionize the robotics sector.

Having built a couple of robots in his high-school days, his primary interest lies in the field of robotics. However, he occasionally occupies himself in areas that are still at their infancy, such as 3D Printing and Speech Recognition. More recently, he has begun his exploration in home automation, wireless networking, the Internet of Things, and smart security systems.

His passion for kindling the benefits of technology is what drives him towards open source and to create a smarter planet.

Aaron Srivastava is a biomedical engineer from North Carolina State University. He is currently working on a neurosurgery project to aid patients undergoing spinal cord stimulation treatments. His main interests are in entrepreneurship, business development, and programming languages. Aaron also does web designing, on the side, as a hobby.

Fangzhou Xia is a dual-degree senior student at University of Michigan, with a background in both mechanical engineering and electrical engineering. His areas of interest in mechanical engineering are system control, product design, and manufacturing automation. His areas of interest in electrical engineering are web application development, embedded system implementation, and data acquisition system setup.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit `www.PacktPub.com`.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`https://www2.packtpub.com/books/subscription/packtlib`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

Arduino is a powerful electronics prototyping platform used by millions of people around the world to build amazing projects. Using Arduino, it is possible to easily connect sensors and physical objects to a microcontroller, without being an expert in electronics.

However, using Arduino still requires us to know how to write code in C/C++, which is not easy for everyone. This is where LabVIEW comes into play. LabVIEW is software used by many professionals and universities around the world, mainly to automate measurements without having to write a single line of code.

Thanks to a module called LINX, it is actually very easy to interface Arduino and LabVIEW. This means that we will be able to control Arduino projects without having to type a single line of code. The possibilities are endless, and in this book, we will focus on several exciting projects in order for you to discover the key features of the LabVIEW Arduino interface.

What this book covers

Chapter 1, *Welcome to LabVIEW and Arduino*, introduces you to the Arduino platform and the LabVIEW software.

Chapter 2, *Getting Started with the LabVIEW Interface for Arduino*, shows you how to install and use the LabVIEW interface for Arduino via the LINX module.

Chapter 3, *Controlling a Motor from LabVIEW*, explains how to make your first real project with Arduino and LabVIEW by controlling a DC motor from LabVIEW.

Chapter 4, *A Simple Weather Station with Arduino and LabVIEW*, talks about how to automate measurements from several sensors that are connected to the Arduino platform.

Chapter 5, *Making an XBee Smart Power Switch*, shows you how to make our own *do-it-yourself* (DIY) version of a smart wireless power switch. We will make a device that can control electrical devices, measure their current consumption, and control the whole power switch from LabVIEW.

Chapter 6, *A Wireless Alarm System with LabVIEW*, helps you connect motion sensors to an Arduino board and monitor their state remotely via LabVIEW to create a simple alarm system.

Chapter 7, *A Remotely Controlled Mobile Robot*, teaches you how to use everything you learned so far to control a small mobile robot from LabVIEW. You will be able to wirelessly move the robot and also continuously measure the distance in front of the robot.

What you need for this book

For this book, you will mainly need the LabVIEW software that is available for all major operating systems. You can either buy it or download an evaluation version for free.

You will also need the LINX module to interface LabVIEW and Arduino, which we will see how to set up and use in *Chapter 2*, *Getting Started with the LabVIEW Interface for Arduino* of the book.

Who this book is for

This book is for people who already have some experience with the LabVIEW software and who want to use the Arduino platform. For example, if you want to automate measurements from sensors and control physical objects with Arduino, but without writing Arduino code, this book is for you.

It is also for people who already have some knowledge of the Arduino platform and who want to learn another way to control their Arduino projects, using LabVIEW instead of coding.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from: `http://www.packtpub.com/sites/default/files/downloads/8221OT_ColorImages.pdf`.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Welcome to LabVIEW and Arduino

National Instruments Corporation, NI, is a world leader when it comes to automated test equipment and virtual instrumentation software. LabVIEW is a product that they have developed, and it is being used in many labs throughout the world. LabVIEW, which stands for Laboratory Virtual Instrument Engineering Workbench, is programmed with a graphical language known as G; this is a dataflow programming language. LabVIEW is supported by **Visual Package Manager (VIPM)**. VIPM contains all the tools and kits to enhance the LabVIEW product.

Arduino is a single-board microcontroller. The hardware consists of an open source hardware board that is designed around the Atmel AVR Microcontroller. The intention of Arduino was to make the application of interactive components or environments more accessible. Arduinos are programmed via an **integrated development environment (IDE)** and run on any platform that supports Java. An Arduino program is written in either C or C++ and is programmed using its own IDE.

Welcome to programming Arduino with LabVIEW. During the course of this book, we will take you through working with Arduino through NI's LabVIEW product. The following are what you will need:

- A Windows or Mac-based machine
- Arduino (Uno preferred)
- LabVIEW 13 for students (or any other LabVIEW 13 distribution)

We will work with Servos, LEDs, and Potentiometers in both analog and digital configurations.

What makes Arduino ideal for LabVIEW

The Arduino community is extremely vast with thousands and even hundreds of thousands of projects that can be found using simple searches on Google. Integrating LabVIEW with Arduino makes prototyping even simpler using the GUI environment of LabVIEW with the Arduino platform.

Officially, LabVIEW will work with the Uno and Mega 2560; however, you should be able to run it on other Arduino platforms such as the Nano. Building your own Uno board is just as simple as linking up the Arduino to LabVIEW. For detailed instructions on how to build your own Arduino Uno, check out the following URL: `http://www.instructables.com/id/Build-Your-Own-Arduino/`.

Significance of using LabVIEW

LabVIEW is a graphical programming language built for engineers and scientists. With over 20 years of development behind it, it is a mature development tool that makes automation a pleasure.

The graphical system design takes out the complexity of learning C or C++, which is the native language of Arduino, and lets the user focus on getting the prototype complete.

LabVIEW significantly reduces the learning curve of development, because graphical representations are more intuitive design notations than text-based code. Tools can be accessed easily through interactive palettes, dialogs, menus, and many function blocks known as **virtual instruments (VIs)**. You can drag-and-drop these VIs onto the **Block Diagram** to define the behavior of your application. This point-and-click approach shortens the time it takes to get from the initial setup to a final solution.

Skills required to use LabVIEW and Arduino

With LabVIEW primarily being designed for and targeted at scientists and engineers, it has not excluded itself from being used by hobbyists. Users who have zero programming skills have been able to take entire projects to completion by just following the intuitive process of dragging controls onto the diagram and setting it up to automate.

We have designed this book to be completely intuitive, using parts that can be easily found at your local electronic store.

To get additional support when using LabVIEW with Arduino, have a look at their forum at `https://decibel.ni.com/content`.

Downloading LabVIEW

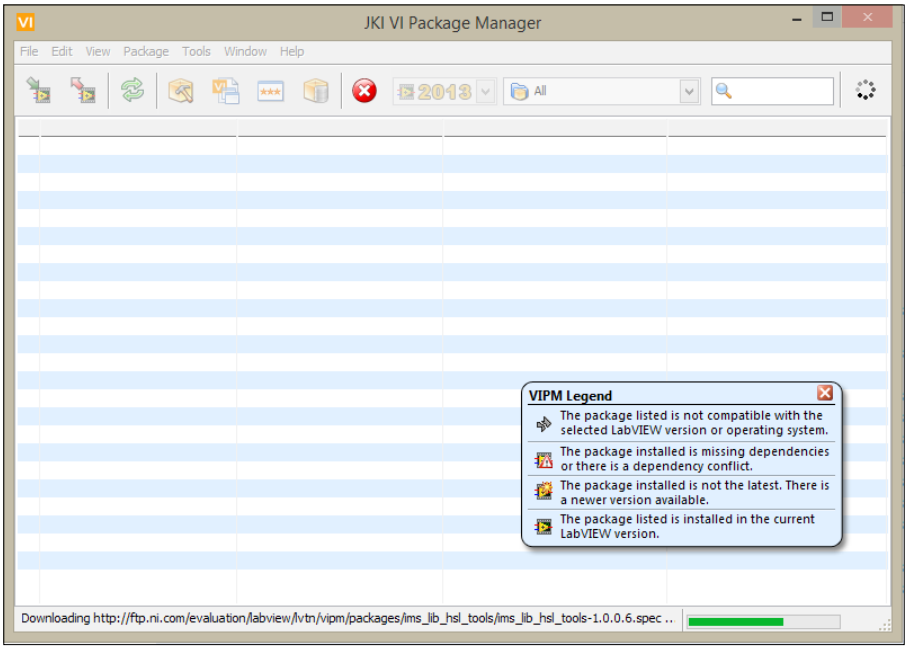
To download or purchase LabVIEW, head out to `http://www.ni.com/trylabview/`. LabVIEW can also be purchased with an Arduino Uno bundle from SparkFun. At the time of writing this book, the URL for this bundle is `https://www.sparkfun.com/products/11225`.

If you did not download LabVIEW, do so now. To try LabVIEW without purchasing it, click on **Launch LabVIEW**.

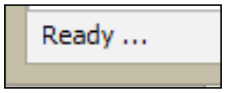
To install the product, click on all the default options. Note that the Arduino plugin is not found in the initial install of LabVIEW.

Once LabVIEW is installed, launch the Visual Package Manager.

The VIPM will now launch. The VIPM application will look like this:



The VIPM will start downloading references to the package bundles into its repository. The status bar is located at the bottom of the application; when the references are downloaded, the status bar will switch to **Ready**.



Downloading the Arduino IDE

To download the Arduino IDE, go to `http://arduino.cc/en/main/software`. This book covers the Windows versions of LabVIEW and Arduino; however, the Mac versions will work just as well.

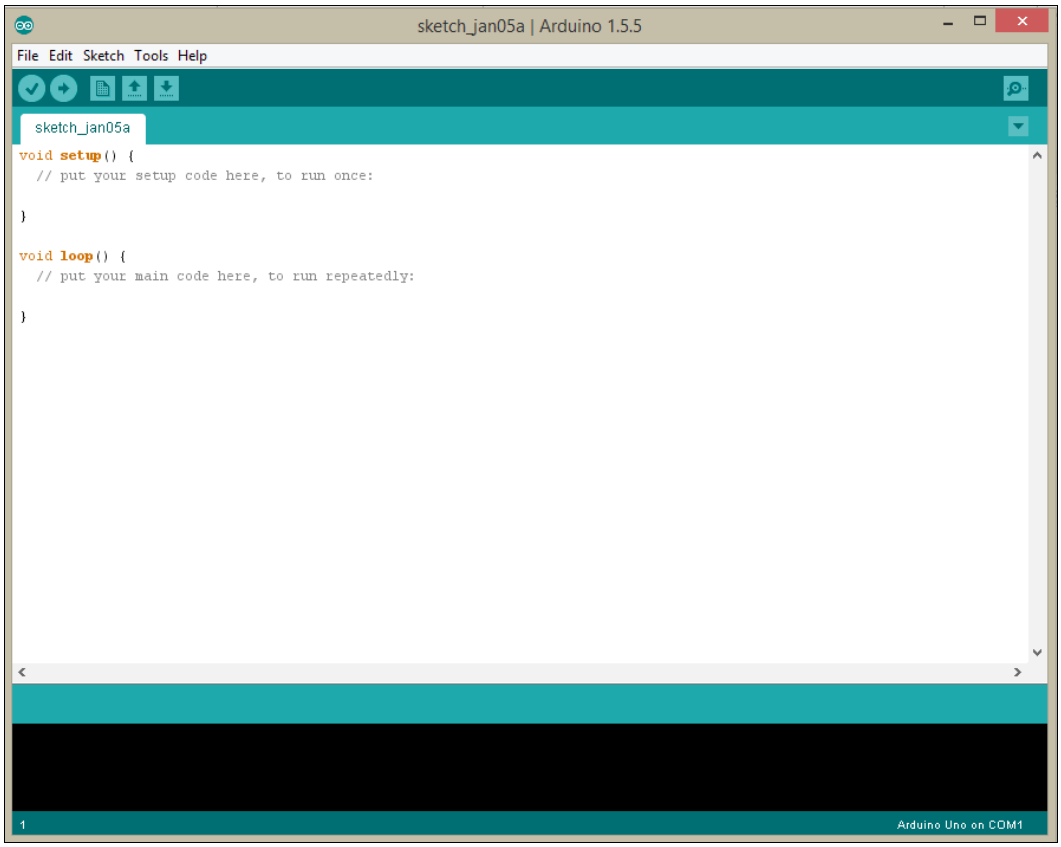
Click on **Windows Installer** to download the Windows version of the Arduino IDE.

At the time of writing this book, the current version of Arduino IDE is 1.5.8.

To install the product, click on all the default options.

Once the Arduino IDE is installed, click on the shortcut shown here to launch the application:

The Arduino IDE will launch with the following screen:



Now that the default settings for each of the applications are set up and launched, we are ready to start programming in each application.

Summary

In this chapter, you learned more about LabVIEW and Arduino. We also installed all the software that we need to get LabVIEW and the Arduino IDE up and running. In the next chapter, we will get the Arduino package for LabVIEW installed and upload a basic sketch to the Arduino board.

2
Getting Started with the LabVIEW Interface for Arduino

In this second chapter of the book, we will see how to hook up LabVIEW and Arduino. We will connect an Arduino board to our computer, install a special package for LabVIEW, and then control the Arduino board directly from LabVIEW. As an example, we will simply light up the on-board LED of the Arduino Uno board from the LabVIEW interface.

This chapter will really be the foundation for all the projects found in this book, so make sure you follow all the instructions carefully.

Hardware and software requirements

On the hardware side, you will not need a lot for this first project of the book. The only thing you will need is an Arduino Uno board (`https://www.adafruit.com/products/50`). This is the same board that we will use in the rest of the book. You can use other boards as well, such as the Arduino Due or the Arduino Pro. However, I recommend that you stick with the Uno board for the whole book.

On the software side, you will need LabVIEW installed on your computer. For this book, I used LabVIEW 2014 for Windows. Of course, you can use LabVIEW on other platforms such as OS X or Linux. You can also use older versions, as the Arduino package that we will use is compatible with LabVIEW 2011 and above. If you don't have LabVIEW yet, you can find all the information at the following link:

`http://www.ni.com/labview/`

After that, you will need the VIPM. This is free software that interfaces nicely with LabVIEW and allows you to automatically install new packages for LabVIEW.

You can download it from the following link:

`http://jki.net/vipm/download`

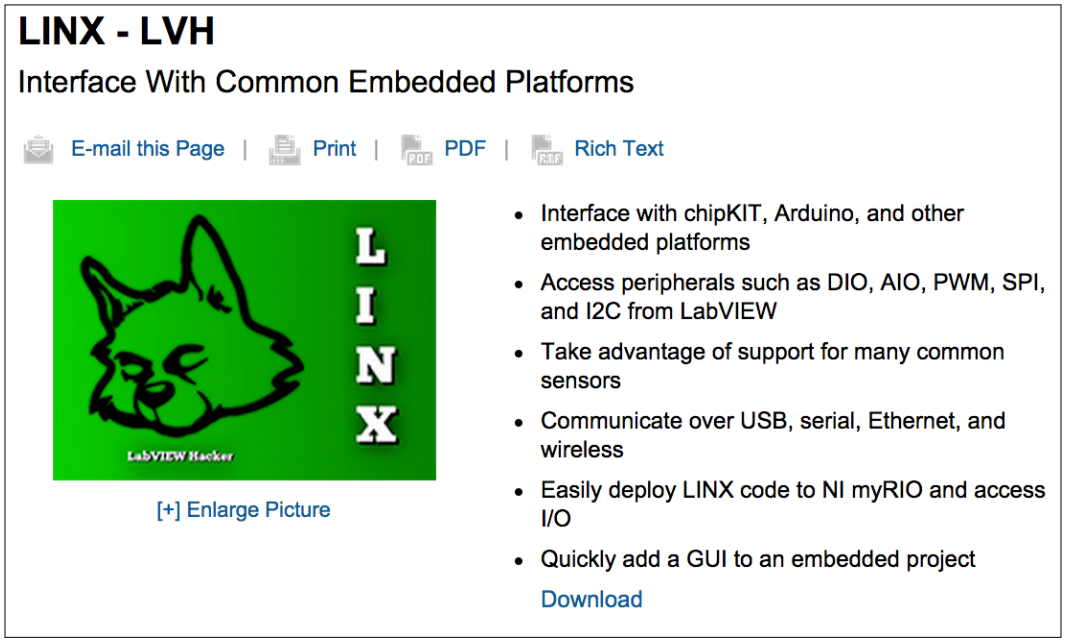
If you encounter an error during the installation that says a version of the software is already installed, make sure that you uninstall the old version first and then retry.

Finally, you will need to install the LINX package, which is a new package replacing the old **LabVIEW Interface for Arduino (LIFA)**.

You can get it at the following URL:

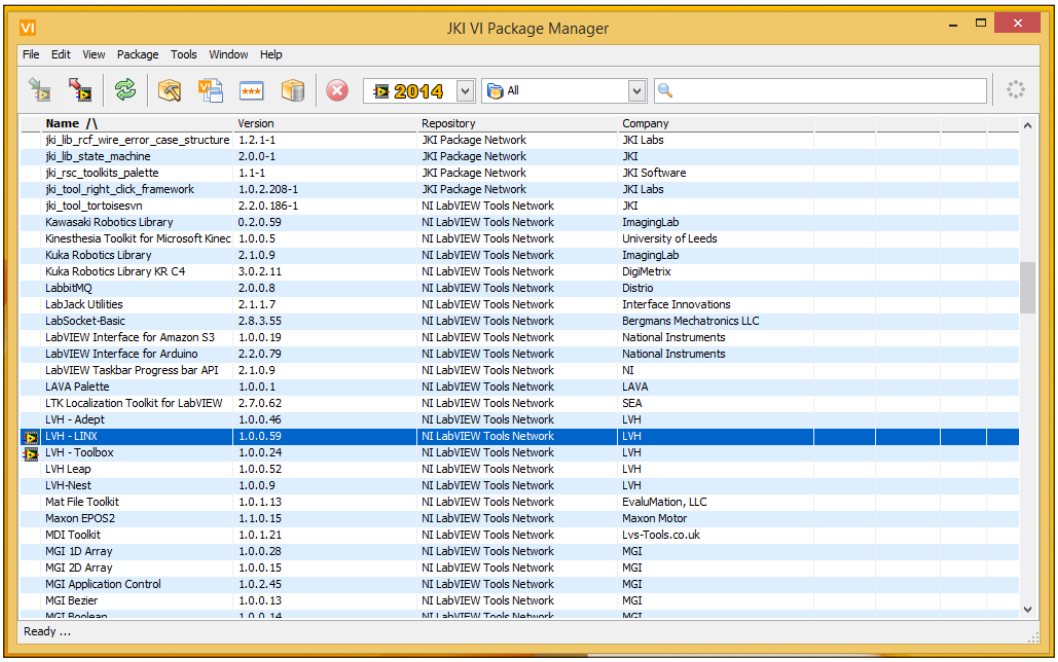
`http://sine.ni.com/nips/cds/view/p/lang/en/nid/212478`

On this page, you will find a link to download the package.



Follow this link, and you will be taken to another page with the direct link for the VI package manager. Click on the **Download Toolkit** button to start the installation process:

The VI package manager should open automatically and install the LINX package.

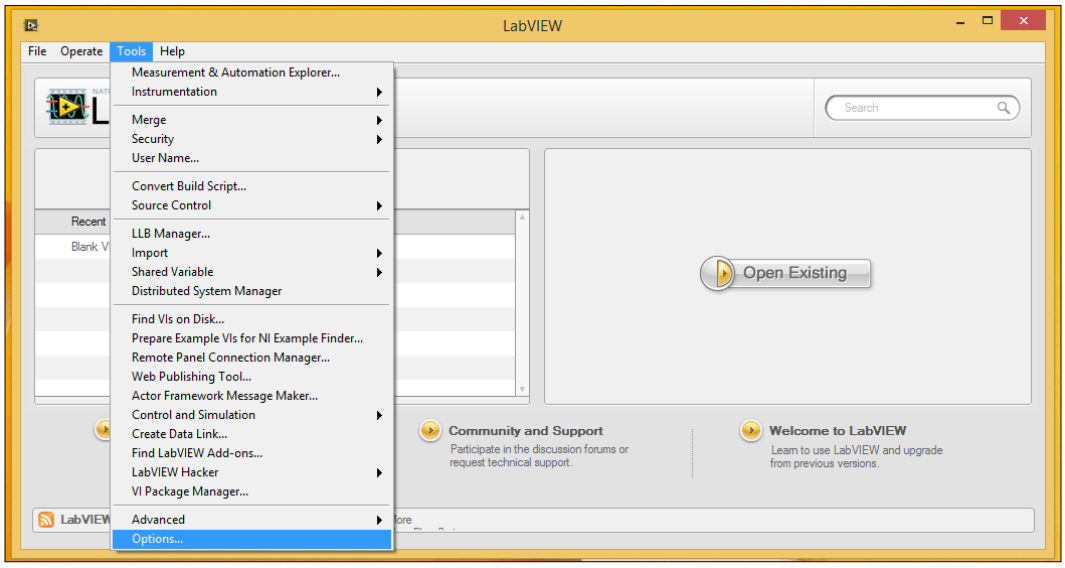


If this does not work and you get an error, it may be linked to the download servers, which may have an issue. In this case, simply retry the procedure, and it should work.

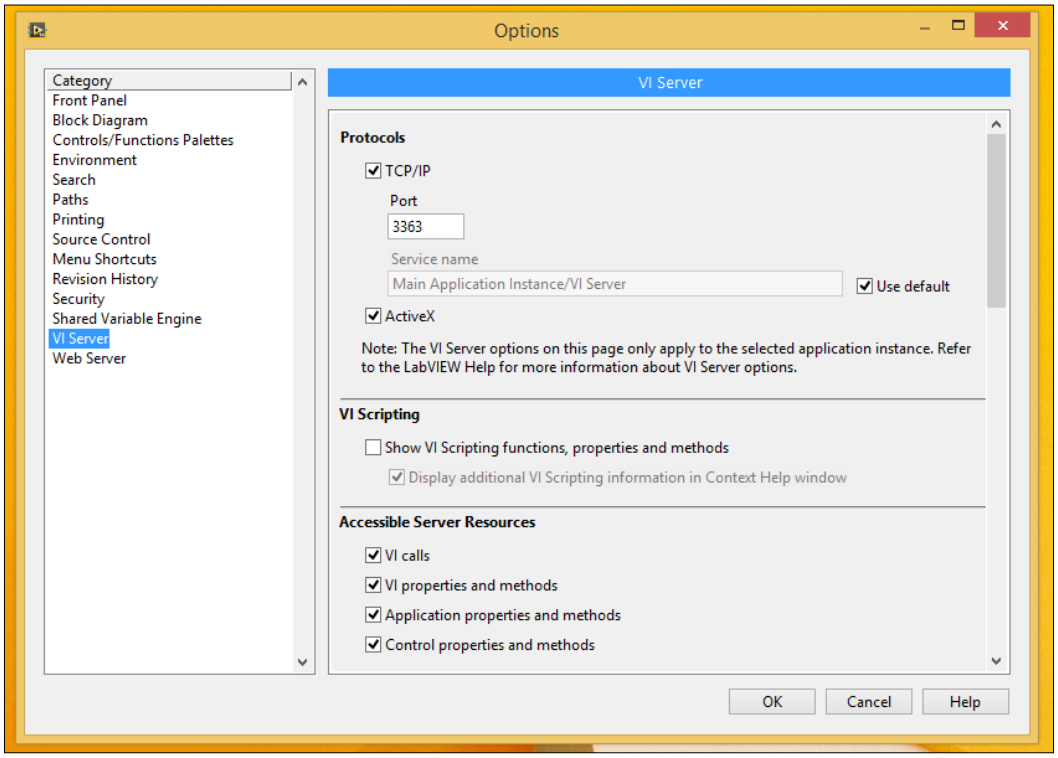
Setting up LabVIEW and LINX

We will now set up LabVIEW and the LINX package so that all the projects of this book can work correctly. Perform the following steps:

1. First, start LabVIEW. Don't create any project, but click on **Tools** and then on **Options**.

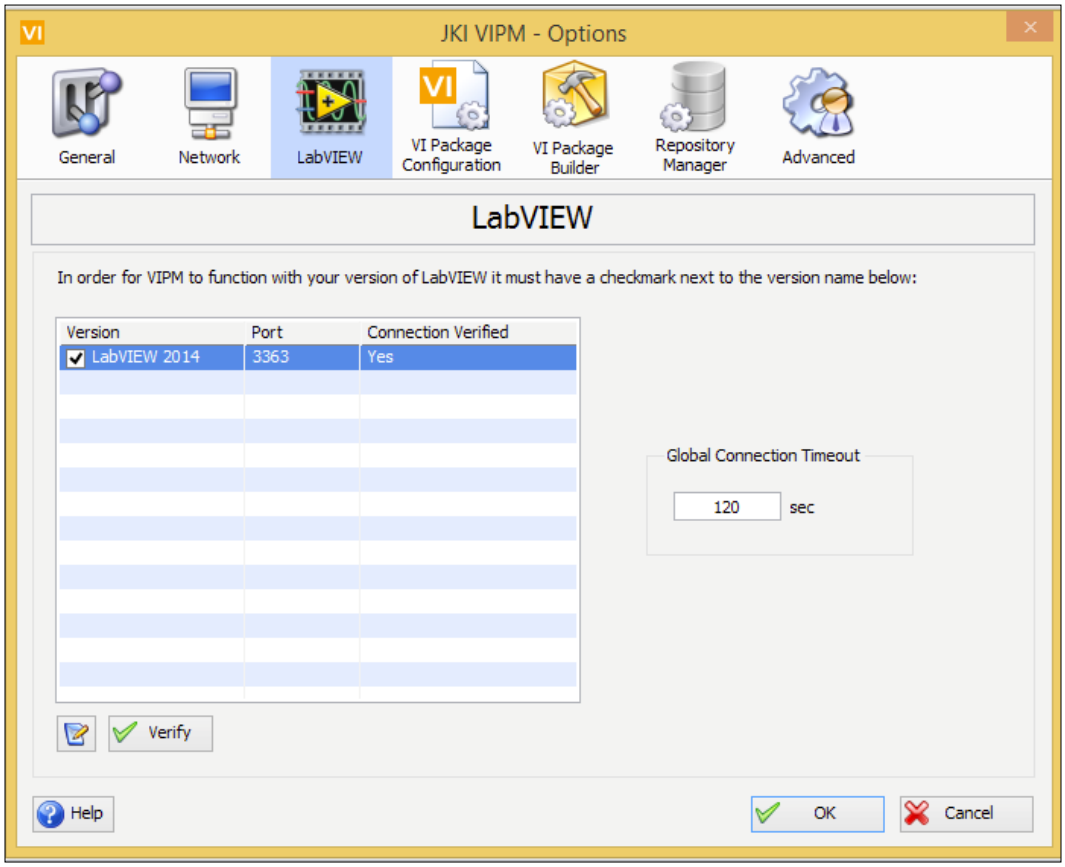


2. You will be taken to the **Options** window of LabVIEW, where you can set all your preferences. Right now, we have to go to the **VI server** menu.



3. You can see that there are some options that you can change here. Change all the options so that they match the options shown in the preceding screenshot.

4. After that, we have to do the same on the VI Package Manager so that both LabVIEW and the Package Manager can talk to each other. On systems like Windows, it was automatically done, but it was not the case on OS X, for example. To do so, simply open the Package Manager, go to the **Tools | Options** menu, and then click on the **LabVIEW** icon.

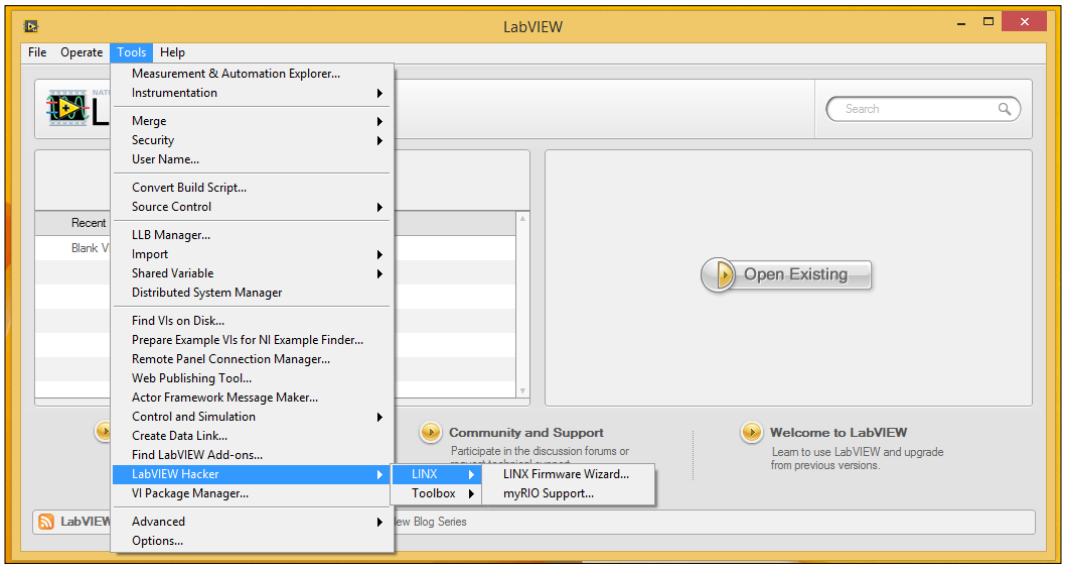


5. In this menu, make sure that the **Port** value next to your LabVIEW installation is the same as the one you defined inside LabVIEW. Correct it here if it is not the case, and confirm.

Testing the installation

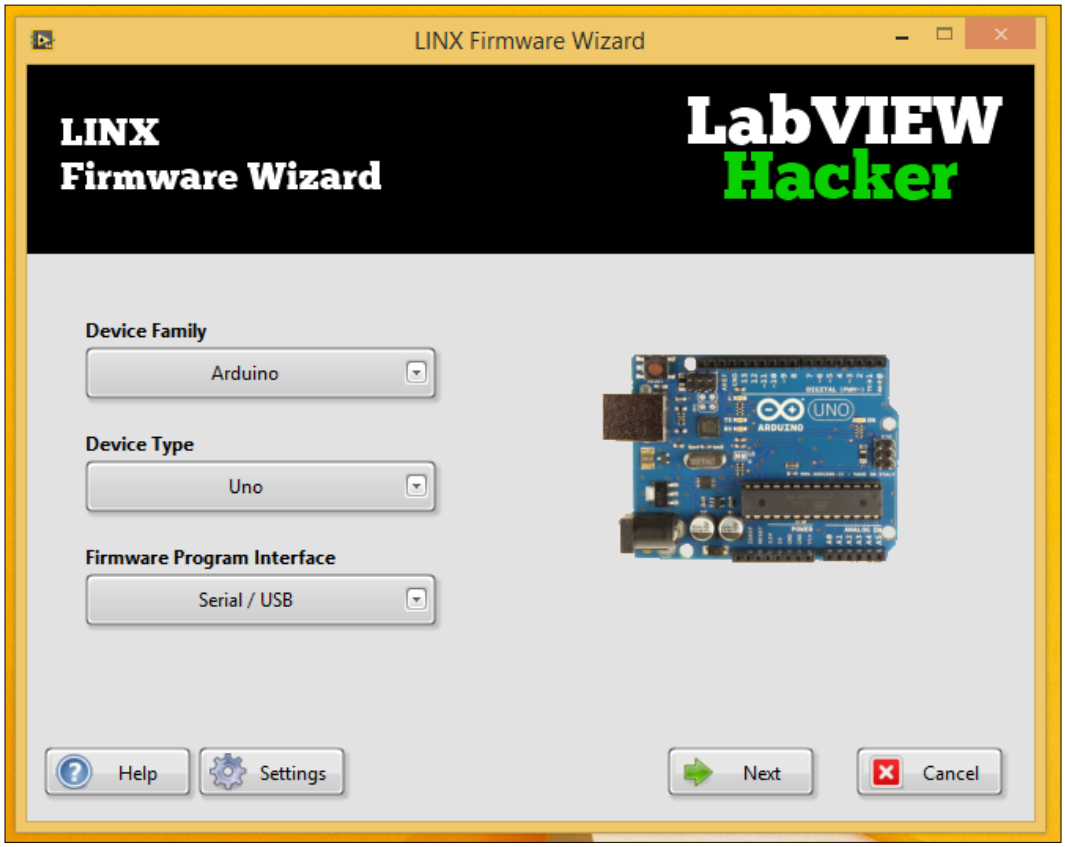
We are now ready to test our LabVIEW/LINX installation and start testing our LabVIEW interface for Arduino.

The first thing that you need to do is go to the main LabVIEW window; then, click on **Tools** and then on **LabVIEW Hacker**, which is the link to access the LINX interface. Then, click on **LINX**, and finally, click on **LINX Firmware Wizard**.



This will take you to the LINX graphical interface that we will use to configure our Arduino board for the project. Note that this step has to be done only one time; once the right software is loaded into the Arduino board, you won't have to touch it again.

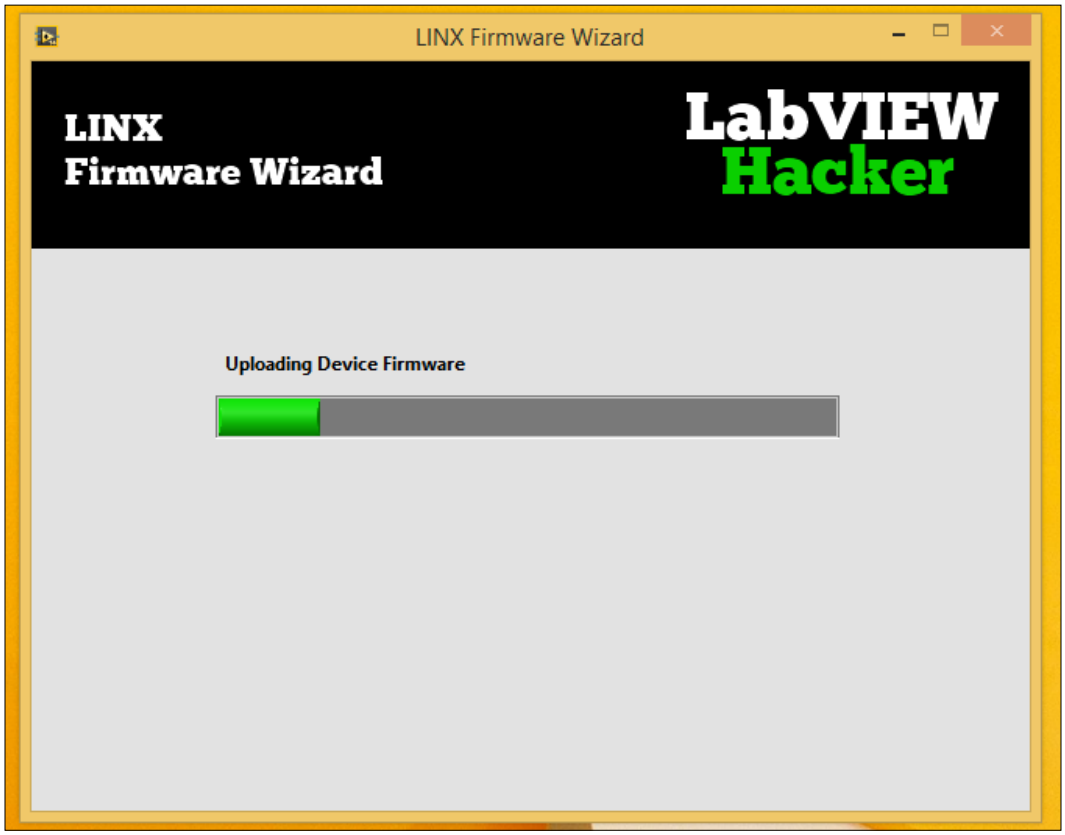
The wizard starts by asking us which board we are going to use. Configure this first page by selecting the same settings as shown in the following screenshot:



After that, you will be prompted to select the Serial Port on which you want the interface to communicate. As I only had one Arduino board connected at that time, I could only select the port that Windows calls COM4. Of course, this will entirely depend on your operating system.

A very simple way to find the COM or Serial Port that corresponds to your Arduino board is to look at the list of proposed Serial ports. Then, disconnect your board and see which Serial Port disappeared; this is the one that corresponds to your board.

Finally, confirm your choice of Serial Port, and start uploading the firmware on the Arduino board.

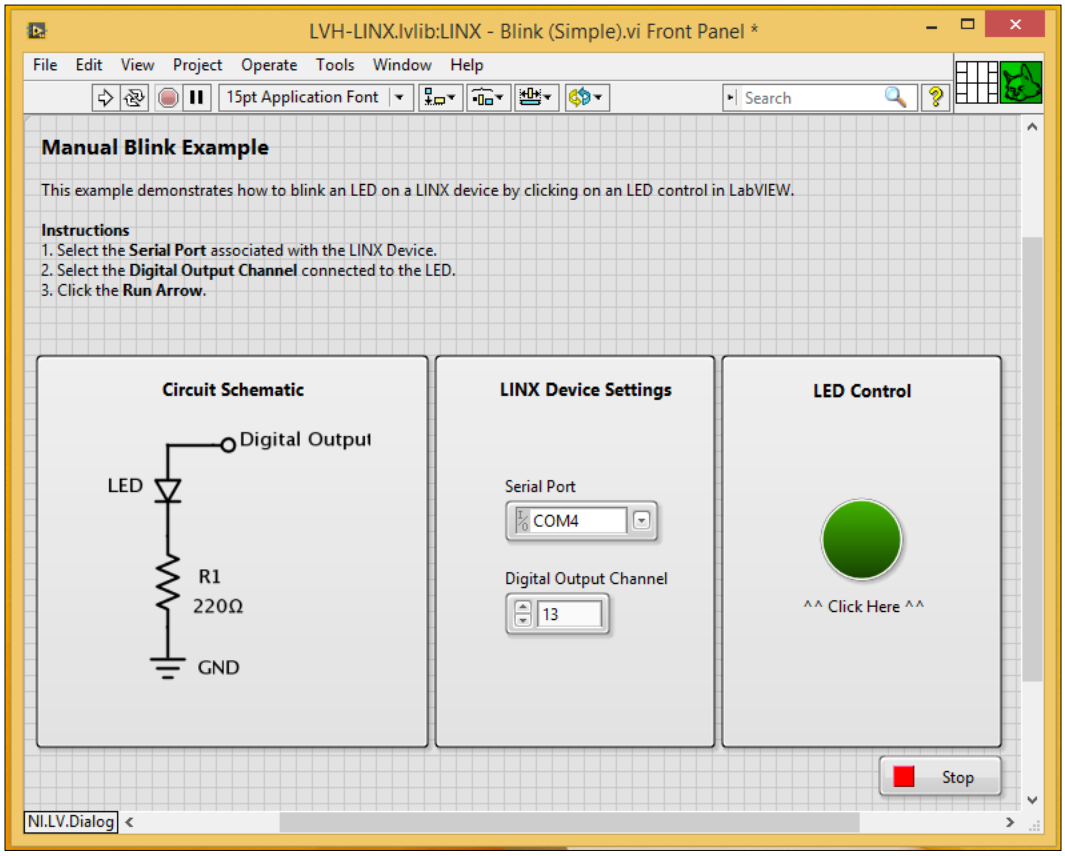


Congratulations! You are now ready to use the LINX interface to control your Arduino board.

If you had an issue at this step, you might have to install the NI-VISA package, which you can download from this link:

`http://www.ni.com/download/ni-visa-4.3/988/en/`

At the end of this setup, LINX will offer to open an example program. Accept this offer, and you will be taken to a new VI.



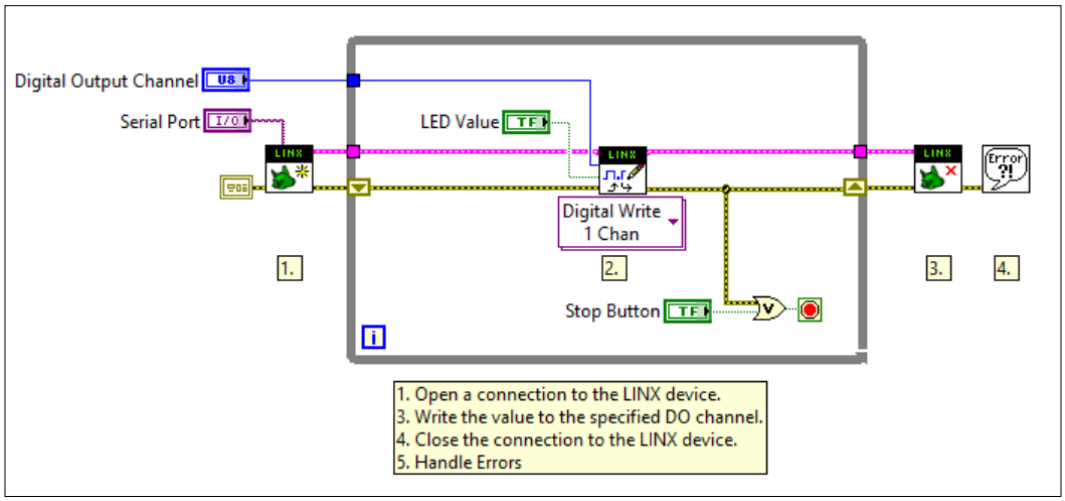
This is called the **Front Panel** of this example project from which you can control the project. As you can see, this VI is really simple, as you can just control the value of a digital pin of the Arduino by clicking on the green button on the right-hand side.

There are two things you need to modify here before you can start the VI. First, you need to set the correct Serial Port in the **Serial Port** box. Just start by typing the name of your port, and it will autocomplete what you are writing.

Then, you need to set which pin you want to control. I simply used pin number **13** here, as it is already connected to the on-board LED on the Arduino Uno board. If you choose any other pin, you will be able to build a simple circuit on your board, as shown in the illustration on the left-hand side of the preceding screenshot.

Let's now use the VI. To do so, simply click on the small arrow on the toolbar. Then, wait for a while. Indeed, the VI will now try to initialize the communication with the Arduino board. If you click on something immediately, it can produce an error. You will know that the initialization process is complete when the Arduino board Serial Port LEDs (TX & RX) are both turned on. Then, click on the green button; you will see that the on-board LED on the Arduino board is immediately turning on or off.

Let's go a bit further and see what is behind that sketch. The details are beyond the scope of this chapter, but it can be interesting to see what is going on at this stage. To do so, go to **Window** and then click on **Show Block Diagram**. Note that you can also use the *Ctrl* + *E* shortcut to switch between **Front Panel** and **Block Diagram**. This will open the following window:



This is the **Block Diagram** window for this project, which is basically what is going on behind the scenes. Some of the components are linked to elements of **Front Panel**, such as the **Serial Port** value. You can see that the core of the project is this **Digital Write** module that we use to send commands to the Arduino board.

For now, we really just wanted to have an overview of what is done in this diagram. In the following chapters of the book, you will see how to build such block diagrams from scratch to build your own projects.

Summary

Let's summarize what we saw in this chapter. You learned how to install the software components that are required for the whole book, such as the VI package manager and the LINX interface for Arduino. This way, you will be able to control Arduino boards from LabVIEW.

We also saw a basic example of a VI used to control an Arduino board, and as an application, we controlled the on-board LED on the Arduino Uno board.

At this stage, it is really important that you perform every step of this chapter correctly, as we will build all the projects in the book based on these steps. If you want to go a little further, you can play with the Block Diagram window of this chapter and modify it a bit. You can also play with the examples that come with the LINX package, which are located in the examples folder of your LabVIEW installation folder.

In *Chapter 3*, *Controlling a Motor from LabVIEW*, you will use what you have learned so far to create your first useful application using LabVIEW and Arduino.

3
Controlling a Motor from LabVIEW

In this chapter, we will write our first VI (LabVIEW program) from scratch. As an example, we will control a DC motor that is connected to the Arduino board. We will build the VI from scratch and then control the direction and speed directly from the LabVIEW graphical interface.

Hardware and software requirements

On the hardware side, you will first need an Arduino Uno board.

For the motor, I chose a small 5V DC motor from Amazon. You can choose any brand that you want for the motor; the important thing is that it has to be rated to work at 5V so that it can be powered directly from Arduino. You can also get a motor that uses higher voltages or currents, but you will need to modify the hardware configuration slightly.

You will also need the L293D motor driver to control the motor from Arduino. This is a dedicated chip that we will use to easily control the motor from LabVIEW. You can also use an alternative to this chip; for example, you can use an Arduino shield that already integrates similar chips on the board. This is, for example, the case of the official Arduino motor shield, which integrates the L298D chip. However, you would need to modify the code slightly if you are using a shield instead of the chip alone.

Finally, you will need a breadboard and jumper wires to make all the connections.

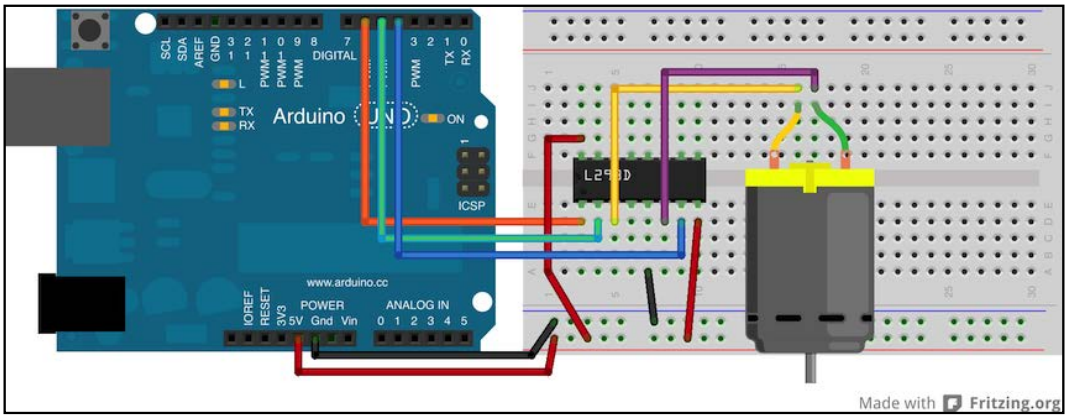
This is a list of all the components required for this chapter, along with the links to find them on the Web:

- Arduino Uno (`https://www.adafruit.com/products/50`)
- L293D (`https://www.adafruit.com/product/807`)
- DC motor (`http://www.amazon.com/Motor-5V-80mA-200mA-torque/dp/B001DAYVA6`)
- Jumper wires (`https://www.adafruit.com/products/1957`)
- Breadboard (`https://www.adafruit.com/products/64`)

On the software side, you will need to have LabVIEW and the LINX package installed. If this is not done yet, refer to *Chapter 2*, *Getting Started with the LabVIEW Interface for Arduino*, to follow all the required steps.

Hardware configuration

Let's now see how to assemble the different components of the project. This schematic will help you visualize the connections between the different components:



To assemble the components follow the steps:

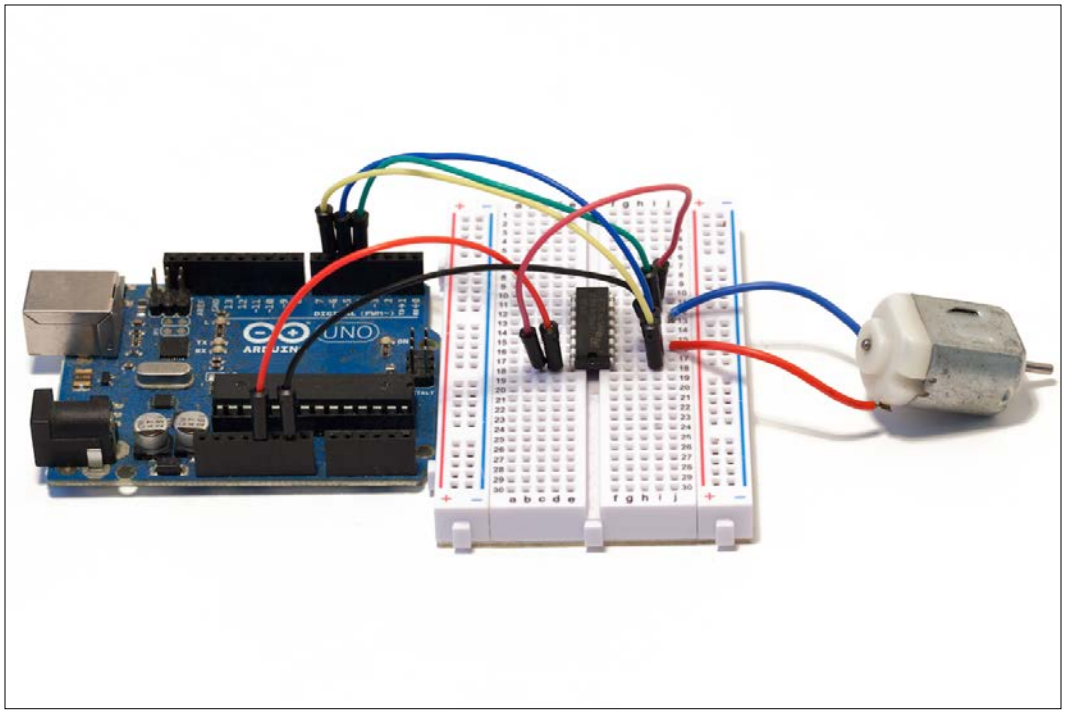
1. First, put the L293D chip in the middle of the breadboard.
2. Then, take care of the power supply; connect the upper-left pin and the lower-right pin of the L293D chip to the Arduino 5V pin.
3. Then, connect one of the pins at the lower center of the chip to the Arduino GND pin.

4. After that, connect the command signals coming from the Arduino, which will be on pins 4, 5, and 6, and the Arduino Uno board.
5. Finally, connect the DC motor to the L293D chip, as shown in the schematic.

To help you out, here is a link to the pins' configuration of the L293D chip:

`http://users.ece.utexas.edu/~valvano/Datasheets/L293d.pdf`

This is what it should look like at the end:



When this is done, you can move to the next step; building the VI in LabVIEW to control the DC motor.

Writing the LabVIEW program

We will now write a new LabVIEW program from scratch so that you can see how the LINX interface for Arduino is working. To start the process, open LabVIEW and create a new blank VI.

We already saw in the previous chapter that there are two main views in LabVIEW: **Front Panel** and **Block Diagram**. In your new blank VI, these two views will be empty. We will first take care of **Block Diagram**, where we will add the elements to control the Arduino board.

Note that we will directly learn about LabVIEW and Arduino by building our first project.

If you want to learn more about the LabVIEW software first, you can visit this link:

`http://www.ni.com/getting-started/labview-basics/`

To learn the basics of Arduino first, the best option is to explore the official Arduino website:

`http://arduino.cc`

The first thing we will place on the blank VI is a While Loop that you can just drag-and-drop from the **Functions** menu (which you can call at any moment with a right-click). The While Loop can be found in the **Structures** submenu. This loop is required for any Arduino board you want to control via LINX, and all the Arduino commands will need to be placed inside this loop.

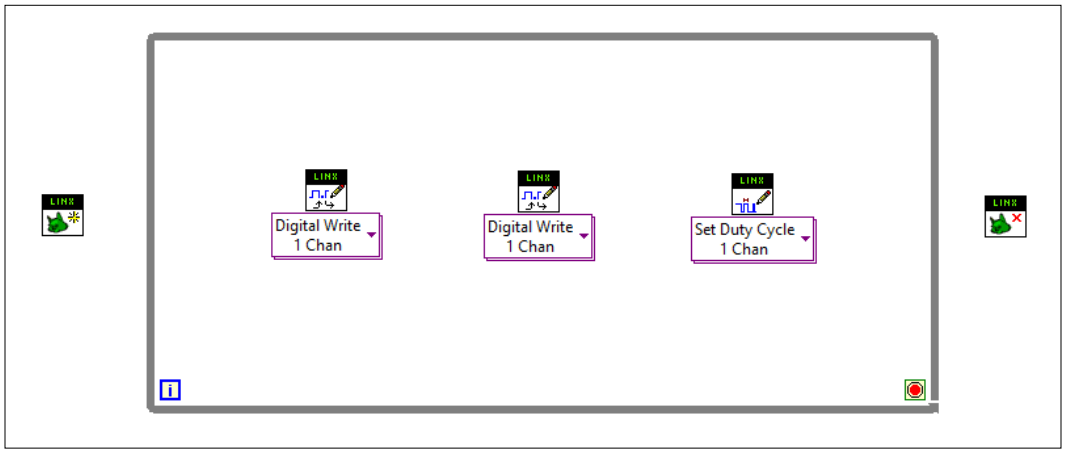
This is how it will look on your VI:

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

After that, we will place our first elements from the LINX package. The first elements we need to place are the LINX initialize and stop elements, which are necessary to tell the software where to start and where to stop. You can find both boxes in the functions panel by going to the **LabVIEW Hacker** submenu.

From the same submenu, place two **Digital Write** blocks (which will be used to control the motor direction) and one **PWM** block (which will be used to control the motor speed). Note that you can find these blocks under the **Peripherals** menu. This is the result:



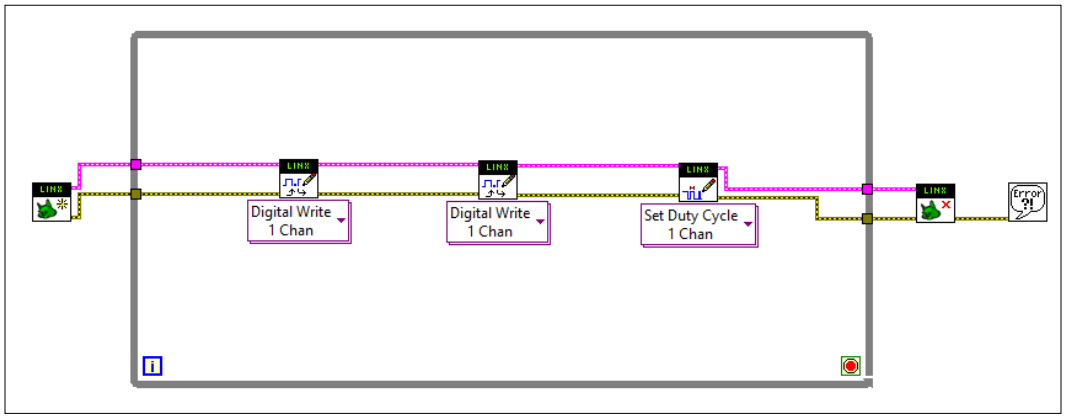
We need a **PWM** block here to control the speed of the motor. PWM stands for Pulse Width Modulation and is used to control the motor's speed or to fade LEDs, for example. On the Arduino board, it is an output of the board that can be set from 0 to 255 on some pins of the Uno board.

To learn more about PWM, you can visit the following link:

`http://en.wikipedia.org/wiki/Pulse-width_modulation`

Now, we need some way to tell LabVIEW in which order we want the sketch to be executed. This is where the error and LINX resource come into play. Simply start from the initialize block on the left-hand side and find the error pin on the block.

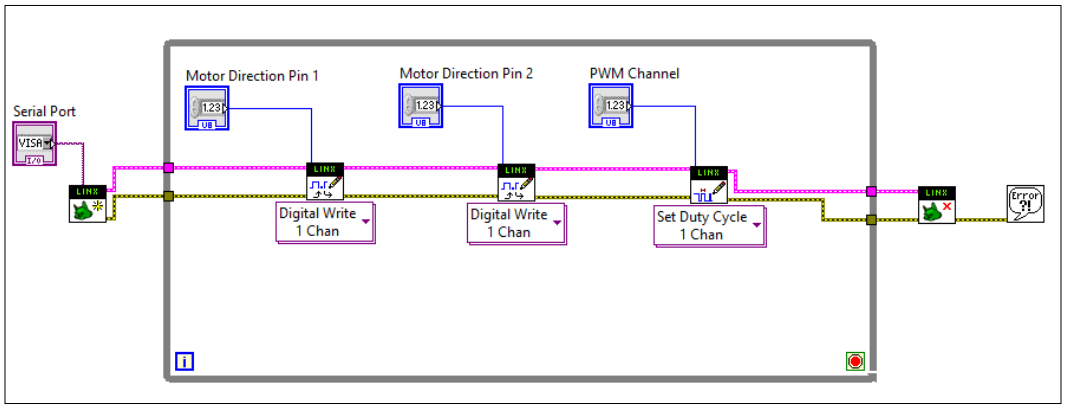
Then, connect the error-out pin of this block to the error-in pin of the first digital block and so on till the end block. After that, do the same with the LINX resource pins. I also added a simple error handler at the end of the VI, just after the stop block. This handler can be found under the **Dialog & User Interface** menu.



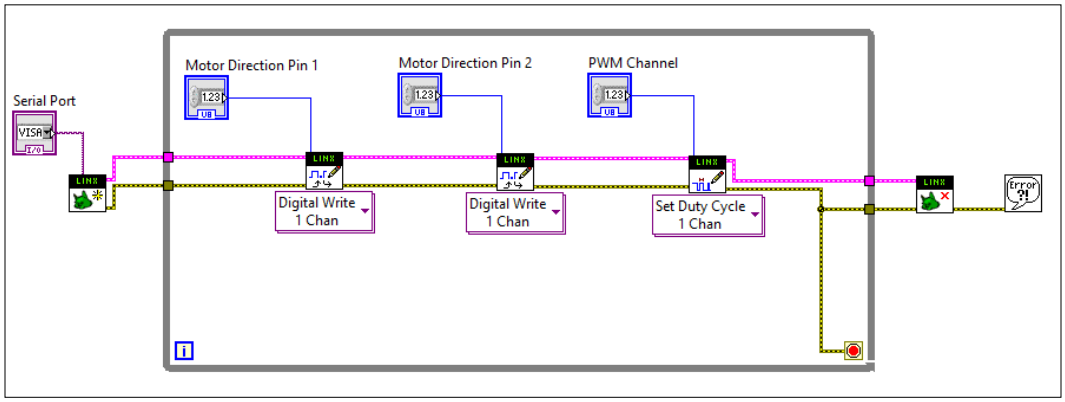
Now that we have the backbone of our project, we will feed the blocks with some inputs. First, add a serial port to the initialize block by going to the serial port pin of the block and right-clicking on it.

Then, go to **Create | Control** to automatically add a serial port input. You will note that the corresponding control is automatically added to **Front Panel** as well. Rename this control to `Serial Port` so that we can identify it in **Front Panel**.

We will also create the same kind of controls for the pins of the blocks we placed earlier. For each block, simply add inputs by right-clicking on the pin's input and then going to **Create | Control**. Also, rename all of these controls so that we know what they mean later in **Front Panel**.



We also need to add an end condition for the While Loop. To do so, we need to connect the little red circle that is located in the bottom-right corner of the While Loop. In this chapter, we will simply connect the error wire directly to this red circle. To do so, just select the input pin of the red circle and connect it to the bottom error wire inside the VI.

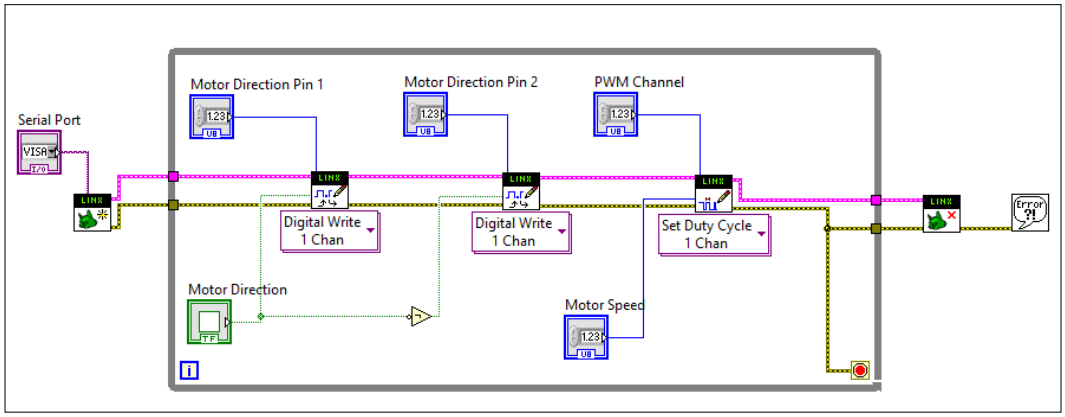


We will now feed the values of the different blocks that we will change from **Front Panel** to control the motor. At this stage, we will keep it simple: we will have some on/off control for the direction and a simple text box for the speed of the motor.

First, let's set the direction that we need to feed on the two first LINX blocks in our VI. The L293D chip requires to be fed with opposite signals on the two direction pins for the motor to rotate in a given direction. For example, when the first **Digital Write** block is on, we want the second one to be off and vice versa.

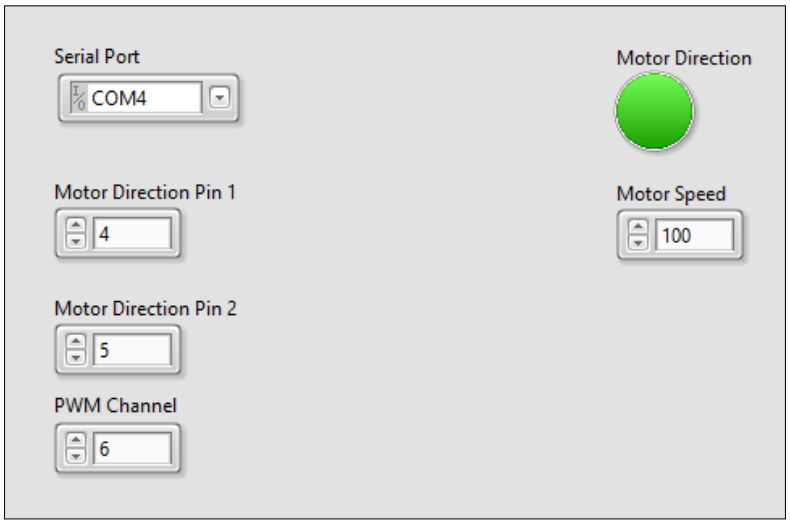
To do so, we will first create a control block on the first **Digital Write** block, again by right-clicking on the input pin and then going to **Create | Control**. Then, we will go to the Functions menu, in Booleans, choose a Not element, and use it to connect our control to the second **Digital Write** channel. This way, we are sure that these two will always be in opposite states.

Finally, also do the same for the **PWM** block by creating a control for the PWM value. This one will simply be displayed as a text input inside **Front Panel**. We will also rename this pin as `Motor Speed` so that we know what it means in **Front Panel**.



You can now go back to **Front Panel** and have a look at all the elements that were automatically added for you. Organize them a little bit so that it is easier to control the motor.

I simply arranged the **Front Panel** so that all the static controls, such as the serial port and pins, are on the left-hand side (we will modify them only once) and the dynamic controls for the motor are on the right-hand side:



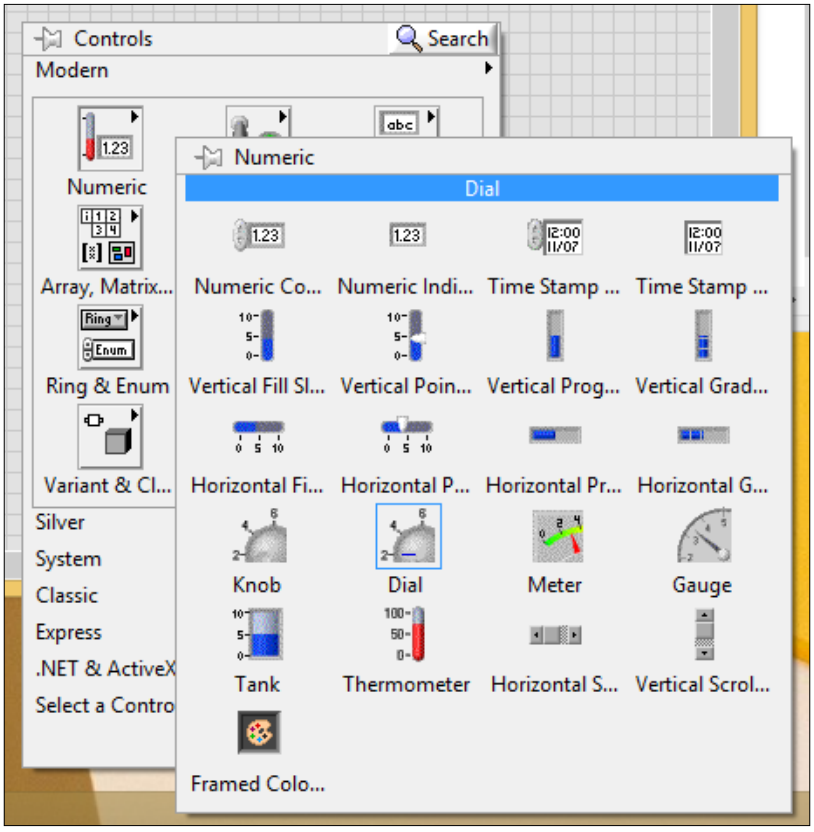
It's now time to test the VI. First, set all the correct pins and your Serial Port, as shown in the preceding image. Then, click on the little arrow in the toolbar to start the VI.

You can now enter a value between 0 and 255 in the Motor Speed input; you will see that the motor starts to rotate immediately. Note that we have to use a value between 0 and 255, as the Arduino Uno PWM output value is coded in 8 bits, so it has 256 values. You can also use the green button to change the direction of the motor.

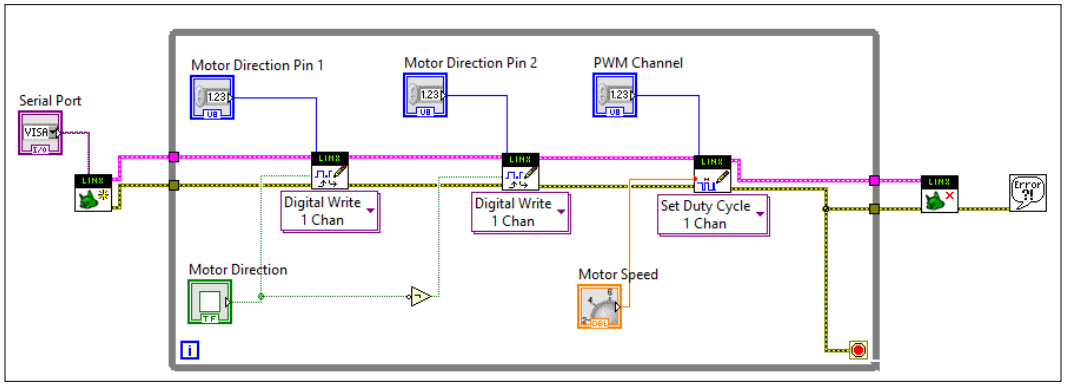
Upgrading the interface

We now have a basic control for our DC motor, but we can do better. Indeed, it is not so convenient to type in the speed of the motor into **Front Panel** every time you want to modify something. This is why we will introduce another kind of control called a **Knob** control.

To add such a control, start from **Front Panel** and right-click to open the **Controls** panel. Then, go to **Numeric** and select the **Knob** control from the menu.

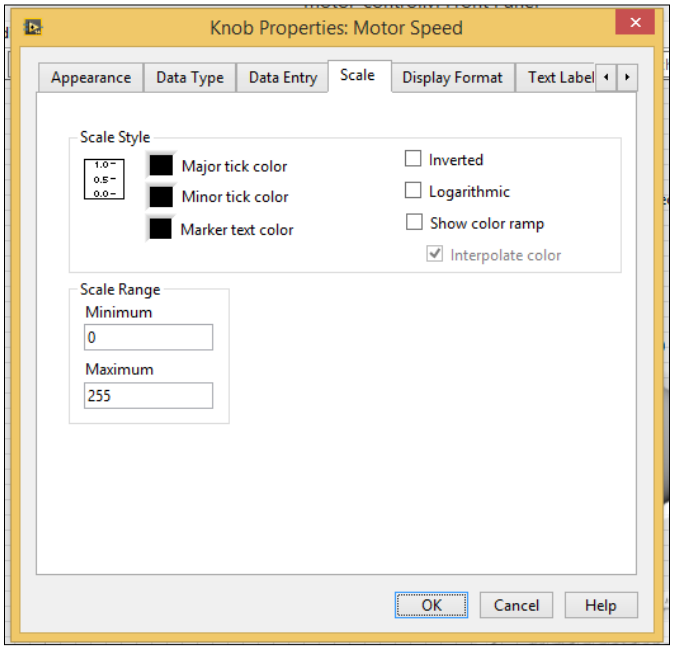


Now, the knob is inserted in **Front Panel**; you can go back to **Block Diagram** where you can remove the old text control from the **PWM** block and connect the new one instead. You can rename it to `Motor Speed` as well.

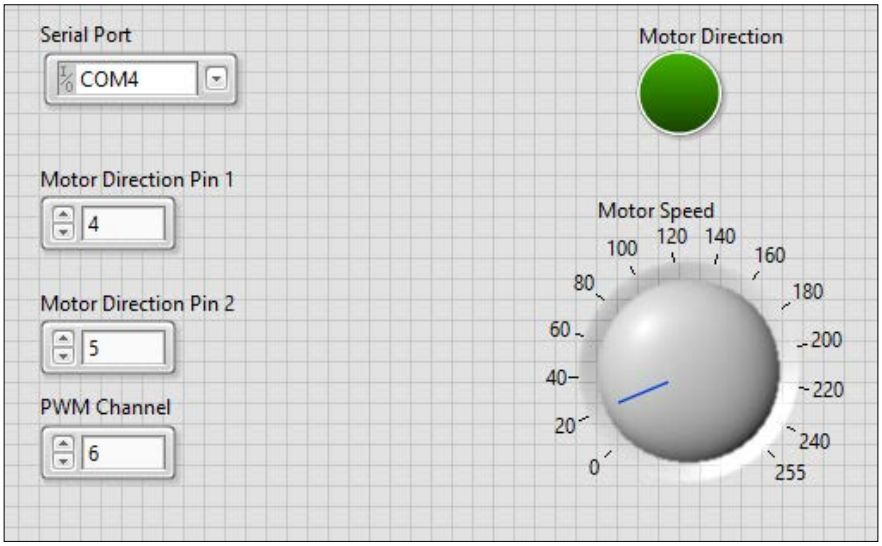


Now, we also need to set the knob so that its output value matches the accepted input of the **PWM** block. Remember that the **PWM** block of LINX accepts values between 0 and 255.

To do so, simply right-click on the **Knob** block and click on **Properties**. In this menu, click on **Scale** and change the minimum and maximum values, as shown in the following screenshot:



You can now go back to **Front Panel**. You will see that the knob is now displaying the correct values, going from 0 to 255. You can also resize the knob at this point so that it is easier to use.



It is now time to test the modified interface. As you did earlier, click on the little arrow inside the toolbar. You can now simply turn the knob to instantly change the rotation speed of the motor.

Summary

Let's summarize what we did in this chapter. We connected a DC motor to Arduino via a dedicated chip to control DC motors. Then, we built an interface in LabVIEW so that we could easily control the direction and speed of this motor. This will be very useful in *Chapter 7*, *A Remotely Controlled Mobile Robot*, of this book, especially when we will build a robot controlled via LabVIEW.

To go further with what you learned in this chapter, there are some things you can do. You can add more motors to the projects and command them all from a single VI in LabVIEW. You can also use what you learned in this chapter to control simpler components such as LEDs.

This chapter was all about controlling outputs. In the next chapter, we will see how to get data from the inputs of the Arduino board and automate measurements using LabVIEW.