Extracted from:

Arduino: A Quick-Start Guide, Second Edition

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Arduino

A Quick-Start Guide

Second Edition

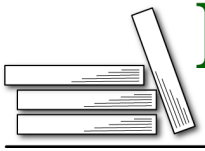
Maik Schmidt

Edited by Susannah Davidson Pfalzer

Arduino: A Quick-Start Guide, Second Edition

Maik Schmidt

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina



Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *https://pragprog.com*.

All circuit diagrams were created with Fritzing (*http://fritzing.org*).

The team that produced this book includes:

Susannah Davidson Pfalzer (editor)
Potomac Indexing, LLC (indexer)
Cathleen Small (copyeditor)
Dave Thomas (typesetter)
Janet Furlow (producer)
Ellie Callahan (support)

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-94122-224-9
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—January 2015

One of the most entertaining electronic activities is simply tinkering: taking an existing product and turning it into something different or using it for an unintended purpose. Sometimes you have to open the product and void its warranty; other times you can safely make it part of your own project.

In this chapter, you'll learn how to hijack a Nintendo Nunchuk controller. It's a perfect candidate for tinkering: it comes with a three-axis accelerometer, an analog joystick, and two buttons, and it is very cheap (less than $20 at the time of this writing). Even better: because of its good design and its easy-to-access connectors, you can easily integrate it into your own projects.

You'll use an ordinary Nunchuk controller and transfer the data it emits to our computer using an Arduino. You'll learn how to wire it to the Arduino, how to write software that reads the controller's current state, and how to build your own video game console. You don't even need a Nintendo Wii to do all of this—you need only a Nunchuk controller (shown in Figure 26, *A Nintendo Nunchuk controller*, on page 6).

What You Need

- An Arduino board, such as the Uno, Duemilanove, or Diecimila
- A USB cable to connect the Arduino to your computer
- A Nintendo Nunchuk controller
- Four wires
- The modified RCA cable you built in Chapter 8, *Generating Video Signals with an Arduino*, on page ?

Wiring a Wii Nunchuk

Wiring a Nunchuk to an Arduino really is a piece of cake. You don't have to open the Nunchuk or modify it in any way. You only have to put four wires into its connector and then connect the wires to the Arduino:

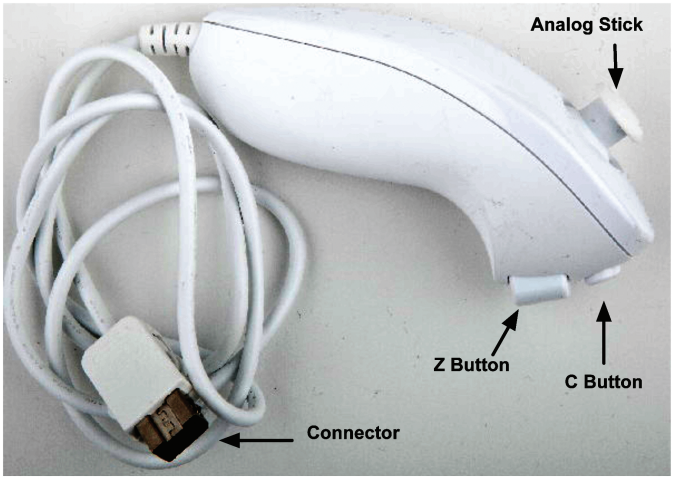
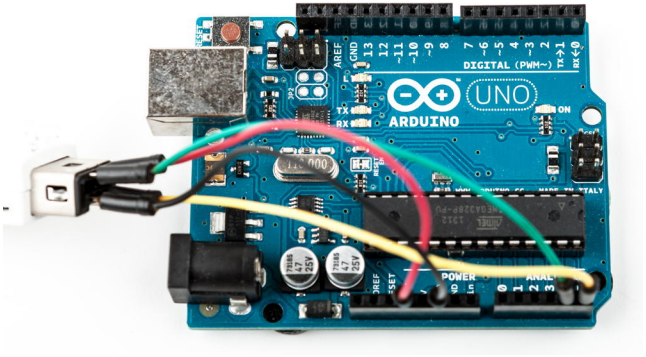
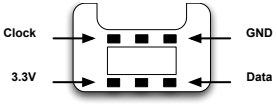


Figure 26—A Nintendo Nunchuk controller





It has six connectors, but only four of them are active: GND, 3.3V, Data, and Clock. Here's the pinout of a Nunchuk plug:

Put a wire into each connector and then connect the wires to the Arduino. Connect the data wire to analog pin 4 and the clock wire to analog pin 5. The GND wire has to be connected to the Arduino's ground pin, and the 3.3V wire belongs to the Arduino's 3.3V pin.

That's really all you have to do to connect a Nunchuk controller to an Arduino. In the next section, you'll see that the two wires connected to analog pins 4 and 5 are all we need to interface with the controller.

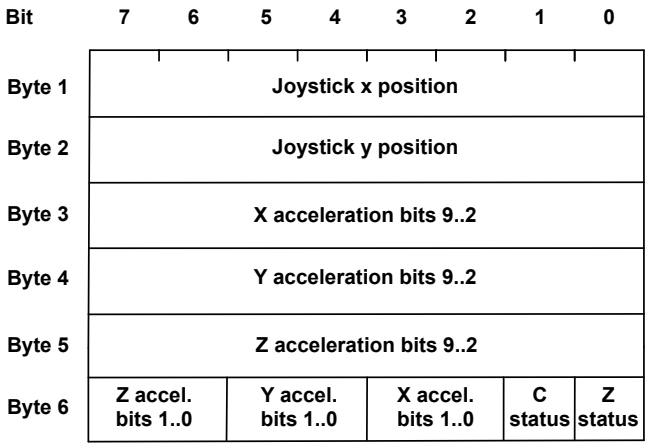
Talking to a Nunchuk

No official documentation shows how a Nunchuk works internally or how you can use it in a non-Wii environment. But some smart hackers and makers on the Internet invested a lot of time into reverse-engineering what's happening inside the controller.

All in all, it's really simple, because the Nunchuk uses the Two-Wire Interface (TWI), also known as I^2C (Inter-Integrated Circuit) protocol.[1] It enables devices to communicate via a master/slave data bus using only two wires. You transmit data on one wire (Data), while the other synchronizes the communication (Clock).

The Arduino IDE comes with a library named Wire that implements the I^2C protocol. It expects the data line to be connected to analog pin 4 and the clock line to analog pin 5. We'll use it shortly to communicate with the Nunchuk, but before that, we'll have a look at the commands the controller understands.[2]

To be honest, the Nunchuk understands only a single command: "Give me all your data." Whenever it receives this command, it returns 6 bytes that have the following meanings:



- Byte 1 contains the analog stick's x-axis value, and in byte 2 you'll find the stick's y-axis value. Both are 8-bit numbers and range from about 29 to 225.

1. http://en.wikipedia.org/wiki/I2c
2. At http://todbot.com/blog/2010/09/25/softi2cmaster-add-i2c-to-any-arduino-pins/, you can find a library that allows you to use any pair of pins for I^2C communication.

- Acceleration values for the x-, y-, and z-axes are three 10-bit numbers. Bytes 3, 4, and 5 contain their eight most significant bits. You can find the missing two bits for each of them in byte 6.
- Byte 6 has to be interpreted bit-wise. Bit 0 (the least significant bit) contains the status of the Z-button. It's 0 if the button was pressed; otherwise, it is 1. Bit 1 contains the C-button's status.

The remaining six bits contain the missing least significant bits of the acceleration values. Bits 2 and 3 belong to the x-axis, bits 4 and 5 belong to Y, and bits 6 and 7 belong to Z.

Now that you know how to interpret the data you get from the Nunchuk, you can start to build a `Nunchuk` class to control it.

Improve People's Lives with Tinkering

Because of its popularity, peripheral equipment for modern game consoles often is unbelievably cheap. Also, it's no longer limited to classic controllers; you can buy things such as snowboard simulators or cameras. So, it comes as no surprise that creative people have built many interesting projects using hardware that was originally built for playing games.

An impressive and useful tinkering project is the EyeWriter.[a] It uses the PlayStation Eye (a camera for Sony's PlayStation 3) to track the movement of human eyes.

A team of hackers built it to enable their paralyzed friend to draw graffiti using his eyes. Because of a disease, this friend, an artist, is almost completely physically paralyzed and can move only his eyes. With the EyeWriter, he can create amazing artwork again.

It's not an Arduino project, but it's definitely worth a look.

a. http://www.eyewriter.org/

Building a Nunchuk Class

The interface of our `Nunchuk` class (and the main part of its implementation) looks as follows:

Tinkering/NunchukDemo/nunchuk.h

```
Line 1 #ifndef __NUNCHUK_H__
#define __NUNCHUK_H__
#define NUNCHUK_BUFFER_SIZE 6

5 class Nunchuk {
public:
  void initialize();
```

```
  bool update();

  int joystick_x() const { return _buffer[0]; }
  int joystick_y() const { return _buffer[1]; }

  int x_acceleration() const {
    return ((int)(_buffer[2]) << 2) | ((_buffer[5] >> 2) & 0x03);
  }

  int y_acceleration() const {
    return ((int)(_buffer[3]) << 2) | ((_buffer[5] >> 4) & 0x03);
  }

  int z_acceleration() const {
    return ((int)(_buffer[4]) << 2) | ((_buffer[5] >> 6) & 0x03);
  }
  bool z_button() const { return !(_buffer[5] & 0x01); }
  bool c_button() const { return !(_buffer[5] & 0x02); }

private:
  void request_data();
  char decode_byte(const char);

  unsigned char _buffer[NUNCHUK_BUFFER_SIZE];
};

#endif
```

This small C++ class is all you need to use a Nunchuk controller with your Arduino. It starts with a double-include prevention mechanism: it checks whether a preprocessor macro named `_NUNCHUK_H_` has been defined already using `#ifndef`. If it hasn't been defined, we define it and continue with the declaration of the `Nunchuk` class. Otherwise, the preprocessor skips the declaration, so you can safely include this header file more than once in your application.

In line 3, we create a constant for the size of the array we need to store the data the Nunchuk returns. We define this array in line 31, and in this case, we define the constant using the preprocessor instead of the `const` keyword, because array constants must be known at compile time in C++.

Then the actual declaration of the `Nunchuk` class begins. To initiate the communication channel between the Arduino and the Nunchuk, you have to invoke the `initialize` method once. Then you call `update` whenever you want the Nunchuk to send new data. You'll see the implementation of these two methods shortly.

We have public methods for getting all of the attributes the Nunchuk returns: the x and y positions of the analog stick, the button states, and the accelera-

tion values of the x-, y-, and z-axes. All of these methods operate on the raw data you can find in the buffer in line 31. Their implementation is mostly trivial, and it requires only a single line of code. Only the assembly of the 10-bit acceleration values needs some tricky bit operations (see *Bit Operations*, on page ?).

At the end of the class declaration, you'll find two private helper methods named request_data and decode_byte. We need them to implement the initialize and update methods:

Tinkering/NunchukDemo/nunchuk.cpp

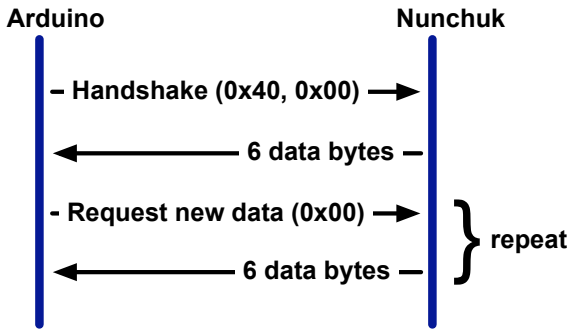
```
Line 1 #include <Arduino.h>
  - #include <Wire.h>
  - #include "nunchuk.h"
  - #define NUNCHUK_DEVICE_ID 0x52
  5
  - void Nunchuk::initialize() {
  -   Wire.begin();
  -   Wire.beginTransmission(NUNCHUK_DEVICE_ID);
  -   Wire.write((byte)0x40);
 10   Wire.write((byte)0x00);
  -   Wire.endTransmission();
  -   update();
  - }
  -
 15 bool Nunchuk::update() {
  -   delay(1);
  -   Wire.requestFrom(NUNCHUK_DEVICE_ID, NUNCHUK_BUFFER_SIZE);
  -   int byte_counter = 0;
  -   while (Wire.available() && byte_counter < NUNCHUK_BUFFER_SIZE)
 20     _buffer[byte_counter++] = decode_byte(Wire.read());
  -   request_data();
  -   return byte_counter == NUNCHUK_BUFFER_SIZE;
  - }
  -
 25 void Nunchuk::request_data() {
  -   Wire.beginTransmission(NUNCHUK_DEVICE_ID);
  -   Wire.write((byte)0x00);
  -   Wire.endTransmission();
  - }
 30
  - char Nunchuk::decode_byte(const char b) {
  -   return (b ^ 0x17) + 0x17;
  - }
```

After including all of the libraries we need, we define the NUNCHUK_DEVICE_ID constant. I^2C is a master/slave protocol; in our case, the Arduino will be the master, and the Nunchuk will be the slave. The Nunchuk registers itself at

the data bus using a certain ID (0x52), so we can address it when we need something.

In initialize, we establish the connection between the Arduino and the Nunchuk by sending a handshake. In line 7, we call Wire's begin method, so the Arduino joins the I^2C bus as a master. (If you pass begin an ID, it joins the bus as a slave having this ID.) Then we'll begin a new transmission to the device identified by NUNCHUCK_DEVICE_ID: our Nunchuk.

We send two bytes (0x40 and 0x00) to the Nunchuk, and then we end the transmission. This is the whole handshake procedure, and now we can ask the Nunchuk for its current status by calling update. In the following figure, we see the message flow between an Arduino and a Nunchuk.



update first pauses for a millisecond to let things settle. Then we request six bytes from the Nunchuk, calling Wire.requestFrom. This doesn't actually return the bytes, but we have to read them in a loop and fill our buffer. Wire.available returns the number of bytes available on the data bus, and Wire.read returns the current byte. We cannot use the bytes we get from the Nunchuk directly, because the controller obfuscates them. "Decrypting" them is easy, as you can see in decode_byte.

Finally, we call request_data to tell the Nunchuk to prepare new data. It transmits a single zero byte to the Nunchuk, which means "prepare the next six bytes."

Before we actually use our Nunchuk class in the next section, take a look at the documentation of the Wire library. In the Arduino IDE's menu, choose Help > Reference and click the Libraries link.

Scientific Applications Using Wii Equipment

Because of the Wii's accuracy and low price, many scientists use the Wii for things other than gaming. Some hydrologists use it for measuring evaporation from a body of water.[a] Usually, you'd need equipment costing more than $500 to do that.

Some doctors at the University of Melbourne had a closer look at the Wii Balance Board, because they were looking for a cheap device to help stroke victims recover.[b] They published a scientific paper verifying that the board's data is clinically comparable to that of a lab-grade "force platform" for a fraction of the cost.

---

a. http://www.wired.com/wiredscience/2009/12/wiimote-science/

b. http://www.newscientist.com/article/mg20527435.300-wii-board-helps-physios-strike-a-balance-after-strokes.html