The Arduino is an open source microcontroller built on a single circuit board that is capable of receiving sensory input from its environment and controlling interactive physical objects. It is also a development environment that allows you to write software to the board, and is programmed in the Arduino programming language. The Arduino has become the most popular microcontroller platform and thus hundreds of projects are being developed using it, from basic to advanced levels.

This book will first introduce you to the most important board models of the Arduino family. You will then learn to set up the Arduino software environment. Next, you will work with digital and analog inputs and outputs, manage the time precisely, establish serial communications with other devices in your projects, and even control interrupts to make your project more responsive. Finally, you will be presented with a complete real-world example by utilizing all the concepts learned so far in the book. This will enable you to develop your own microcontroller projects.

Who this book is written for

If you are a hobbyist who wants to develop projects based on Arduino as the main microcontroller platform or an engineer interested in finding out what the Arduino platform offers, then this book is ideal for you. Some prior knowledge of the C programming language is required.

What you will learn from this book

- Select the correct Arduino board to meet the prerequisites of your project
- Set up a working Arduino development environment and get to grips with all of its inner workings
- Work with digital and analog outputs to act on external devices
- Sense your environment in a digital or analog way by using different types of sensors such as potentiometers, photocells, or thermistors
- Develop time-based projects that will manage the time in a precise manner
- Make your project interact with other devices via serial communications
- Use interrupts to deal with unexpected events
- Get familiar with a complete real-world project that summarizes all the concepts learned throughout the book
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 5 "Sensing the Real World through Digital Inputs"
- A synopsis of the book’s content
- More information on Arduino Essentials

About the Author

Francis Perea is a vocational education professor at Consejería de Educación de la Junta de Andalucía in Spain with more than 14 years of experience.

He has specialized in system administration, web development, and content management systems. In his spare time, he works as a freelancer and collaborates, among others, with ñ multimedia, a little design studio in Córdoba, working as a system administrator and the main web developer.

He has also collaborated as a technical reviewer on SketchUp 2014 for Architectural Visualization Second Edition, Arduino Home Automation Projects, and Internet of Things with the Arduino Yún, all by Packt Publishing.

When not sitting in front of a computer or tinkering in his workshop, he can be found mountain biking, kite surfing, or working as a beekeeper, taking care of his hives in Axarquia County, where he lives.

To Salomé: I owe it all to you.
Arduino Essentials

The Arduino platform has become de facto standard when talking about microcontrollers. With a wide range of different board models, it can cover a wide spectrum of projects, and its ease of use has made it the preferred platform for those starting out in the microcontroller world.

If you are a hobbyist wanting to develop projects based on Arduino as its main microcontroller platform or an engineer interested in knowing what the Arduino platform offers, then this book is ideal for you.

If you have little or no previous experience in these kinds of tools, this book will help you get a complete view of the platform and the wide peripherals it has to offer by following a carefully designed set of project examples that cover the most important platform features.

Whether you have never written a line of code or you already know how to program in C, you will learn how to work with Arduino from the point of view of both hardware and software thanks to the easily understandable code that accompanies every project that has been developed exclusively with that premise in mind. This will be easy for those who don't have previous experience in programming.

This book was written with the aim to present the Arduino platform to all those wanting to work with Arduino but without any great knowledge of the microcontrollers scene. It will gradually develop a wide set of projects that have been designed to cover the most important aspects of the Arduino platform, from the use of digital and analog inputs and outputs to harnessing the power of interrupts.

What This Book Covers

*Chapter 1, Meeting the Arduino Family,* introduces you to the Arduino platform, and the different board models that integrate the Arduino family are presented, noting their common aspects.

*Chapter 2, The Arduino Development Environment,* shows you how to download, install, and set up a working Arduino integrated development environment and gives a complete explanation of its use and commands.

*Chapter 3, Interacting with the Environment the Digital Way,* covers the connection and use of digital outputs by dealing with simple devices that can be digitally operated, such as LEDs.

*Chapter 4, Controlling Outputs Softly with Analog Outputs,* shows you how to manage analog outputs and the use of transistor drivers to deal with high-current devices, such as motors.
Chapter 5, *Sensing the Real World through Digital Inputs*, explains the use of digital inputs by giving examples of typical applications, such as buttons and switches, and proposes an optical coin detection device that uses an optocoupler.

Chapter 6, *Analog Inputs to Feel between All and Nothing*, presents analog inputs and their use and offers two new projects: an ambient light meter with a photocell and a motor speed controller by using a potentiometer as an input device.

Chapter 7, *Managing the Time Domain*, introduces you to the different tools the Arduino library offers to deal precisely with time by building two more projects: a simple timer and a visual and acoustic metronome.

Chapter 8, *Communicating with Others*, shows you how to connect your Arduino projects to other platforms via serial communication and how to use the Serial Monitor to read from and send data to Arduino. A computer-controlled motor speed driver and a dial thermometer will be built.

Chapter 9, *Dealing with Interrupts*, shows you how to use interrupts to deal with unexpected events and to understand the difference between having to wait for something to occur and be called when it occurs. We will use a tachograph as a good example to show you all these concepts.

Chapter 10, *Arduino in a Real Case – Greenhouse Control*, gives you a complete real example of a project that summarizes all the concepts learned throughout the book.
Dealing with outputs is really just the half of the pie. Inputs are the other half, and if I may, I'd say they are the most interesting part of any project you will work on. In this chapter, I'll introduce you to the use of digital inputs as a way to make your project sense their environment through the assembly and programming of two different examples that I'm sure you will like.

Sensing by using inputs

Inputs, whether they are digital or analog, are the way through which Arduino can sense what is happening around it. In some cases, they are used as an interface with the user, such as when we connect buttons and switches. In other cases, we use inputs to measure a physical variable that will make our project react in some way or other.

Nowadays, we can find almost a different sensor for every physical variable we want to measure and an endless number of devices that allow humans to interact with electronic devices. So, the first thing we will have to consider is the type of sensor we will need for our project and, most importantly, from the point of view of connection and programming, the type of signal it generates, digital or analog. This is because it will affect the way in which we have to connect it to our Arduino board, by way of a digital or analog input, and program our sketch accordingly.
Digital sensors usually give us a 0V or 5V voltage or any way to obtain it, which can be easily traduced into HIGH or LOW values in the Arduino code. In other cases, they provide a different value that we will have, by means of additional circuitry, to adapt to the Arduino digital input allowed voltage, that is, 0V to 5V.

Regarding analog sensors, their main characteristic is that they provide a continuous value between a range, and we will also have to adapt them to our Arduino allowed range, but we will talk deeply about analog inputs and sensors in Chapter 6, Analog Inputs to Feel between All and Nothing.

At the moment, let's see the simplest digital input circuit, a switch, and how to program it.

**Connecting a button as a digital input**

The simplest circuit that we can prepare to be used as a digital input is that consisting of a switch connected to a digital input of the Arduino board.

A typical momentary push button usually has two states, open or closed, and acts like a switch that, when not pressed, keeps the circuit open, preventing the connection between the two parts of the circuit it connects, and when pressed, makes these two parts connected.

Keeping in mind that an Arduino digital input can sense values of 0V and 5V, we have to prepare our circuitry in such a way that it provides a 0V signal to the Arduino digital input when we want to consider it LOW and a 5V signal when we want to consider it HIGH.

To be precise, the Arduino microcontroller can sense anything between 3V and 5V as a HIGH value and something between 1.5V and 0V as a LOW value, the range between 1.5V and 3V undetermined and, thus, not valid.

For the purposes of our first digital input example, we will use a very simple circuit that you can see in the next schematic:
Although it is a simple circuit, it may deserve some explanation. It basically exposes two different states: the button or switch pressed or released:

- **Released**: When the button is not pressed, the two parts of the circuit remain unconnected, thus not flowing any current through that branch of the circuit. In this case, the Arduino board only senses a GND or 0V value through the digital input by which we connect this circuit to it due to the fact that it is physically connected to the ground through the resistor.

- **Pressed**: When we press the button or close the switch, what we are internally doing is connecting both parts of the circuit, that is, the 5V upper side to the GND lower side, allowing current to flow from one to the other through the 10K resistor. This is precisely why we use the 10K resistor: to provide some load to this branch of the circuit, preventing the short circuiting of the power source. From the point of view of the Arduino board, it senses a 5V signal in the pin we use to connect this circuit to the board.
The momentary push button

In the next screenshot, you can see a bunch of different momentary push buttons of two kinds:

- **Printed circuit board (PCB) soldering**: These used to be smaller, really tiny in some cases, and they come with little legs that allow them to be soldered through holes in a PCB. In the screenshot, they are the three on the left-hand side.

- **Panel mounting**: These are the three on the right-hand side in the picture. Usually, they come in a much bigger size than the PCB kind and come with bigger connections so that wires can be soldered to them. They usually come with some kind of nut so that they can be fixed to a panel.

![Different momentary push buttons](image1.jpg)

Physical differences apart and looking at them from the connection point of view, we have to notice an important difference between both groups.

If you take a closer look at the PCB group or, even better, if you perform a simple search on the Internet, you will notice that these kinds of momentary buttons usually come with a total of four legs instead of simply two. Usually, the reason is to give a stronger union to the PCB group by allowing up to four soldering points, but also because they have their connections duplicated to provide a more reliable connection between both points of the switch.
This means that usually, momentary push buttons with four legs have them internally connected two by two, which looks like what is shown in the next diagram:

![An internal wiring connection of a typical momentary push button](image)

This simple feature has to be taken into serious consideration when connecting one of these buttons through a breadboard, because if we connect them in the wrong way, we could easily short-circuit the Arduino board, with subsequent bad consequences.

The first thing you should do when using one of these momentary push buttons is to identify every leg so that you know which of them are internally connected. They usually come with some kind of indication or diagram, but if they don’t, you could always use a voltmeter or continuity tester to find out.

Once you have determined which legs are connected, you should always connect the momentary push button just above the centerline of the breadboard, which divides the two groups of holes in such a way that you leave two of the replicated connections out of use and ensure that the other two corresponding to the two contact points of the switch are always connected in different columns of the breadboard connections block. This way, they prevent a short circuit when you connect them to the positive rail (5V) and ground.

Also remember that this kind of circuit needs a current limiting resistor in a part of the branch that gets closed between V+ and the ground to prevent a short circuit again. A value of 10K is more than sufficient.
Complete circuit schematic

Here, you have the complete circuit schematic for this example. As you can see, the connection point between the switch and the resistor is connected to the Arduino pin number 7 in this example, and we will use pin number 9 as an output to connect the LED that we will use to blink or fade.

Given that we will deal with the LED as a digital or analog device up to the pressing of the button, we have to connect it to the Arduino board through any of the PWM pins that will allow us to use it with the `digitalWrite()` or `analogWrite()` functions.

The only consideration to be made about this circuit is regarding the right connection of the momentary button we will use, and we have talked about this in the previous paragraph.
Breadboard connections diagram

All that said, I’m sure you will see it clearly in the breadboard connections diagram, where you can notice how the two upper legs of the momentary push button are left on the upper-side connections block of the breadboard without connecting anything to them and separating them from the other two legs, thanks to the central space of the breadboard.

You can also see how once we push the button, the current flows from the positive rail to the ground through the switch and the current limiting resistor, and it is in just this point, after the switch and before the resistor, where we take a wire to the Arduino pin that we will use as a digital input.
Writing code to react to a press

Before we go into the details of the programming of such a circuit, we'll have to decide what behavior we will unchain in our project once the event of a press is detected. For the sake of simplicity, we will make an LED blink or fade depending on the press of the button.

This way, we can test all that we have already learned in a simple example and this will also lead us to meet another very important control structure of any programming language, called a conditional bifurcation, which some say is the basic structure that makes any program a logical structure and not a simple sequence of instructions.

Once the circuit is assembled, let's take a look at the code we are going to write to detect and react to the button press as a digital input. Here, you have the complete code for this example:

```cpp
/*
   Chapter 05 - Sensing the real world through digital inputs
   Sensing a switch
   By Francis Perea for Packt Publishing
*/

// Global variables we will use
int led = 9;
int button = 7;
int pressed = 0;
int intensity = 0;
int increment = 10;

// Configuration of the board: one output and one input
void setup() {
    pinMode(led, OUTPUT);
    pinMode(button, INPUT);
}

// Sketch execution loop
void loop() {
    // We read the button pin
    pressed = digitalRead(button);
    // if it is not pressed lets blink digitally
    if (pressed == LOW) {
        digitalWrite(led, LOW);
        delay(50);
    }
}
digitalWrite(led, HIGH);
delay(50);
}
// otherwise lets fade analogly
else {
    for(intensity=0; intensity<=255;
        intensity=intensity+increment){
        analogWrite(led, intensity);
        delay(30);
    }
    for(intensity=255; intensity>0; intensity=intensity-
        increment){
        analogWrite(led, intensity);
        delay(30);
    }
}
}

**Configuring and reading a digital input**

From the point of view of programming, there are basically two main new concepts in this example regarding the use of a digital input:

- **pinMode(button, INPUT):** In the setup() function, we have to set the pin we are going to sense as an input so that Arduino can read from it. The function is just the same pinMode() function we have been using so far, but instead of using the OUTPUT constant, this time, we use INPUT to set the pin accordingly.

- **digitalRead(button):** When we deal with an input whether it is a digital or analog one, we are going to receive the read value instead of setting a value like how we have been using outputs. This means that we only pass a parameter indicating the pin to be read to the reading function, analogRead() in this case, but it also means that this function, as opposed to writing functions, is going to return us a value that we have, in some cases, to store in a variable for later use.

Being realistic, in this example, it isn't really necessary to store the read value in a variable because we are not going to use it anymore in the rest of the code, and we could simply have used the digitalWrite() function inside the if parentheses like if(digitalRead(button) == LOW), but for the sake of clarity in this first example, I preferred to use a variable this time.
Taking decisions with conditional bifurcations

Beside the differences between the use of inputs and outputs, there is an even more important concept in this example, and it is the use of the if control structure to decide what has to be done depending on the state of a previous event.

The if control structure is the basis of any programming language, giving them the power to decide and act on consequences.

The simplest syntax of the if sentence is as follows:

```java
if (condition) {
    Block of instructions to be executed
}
```

Being a condition, any logic expression is one that evaluates as true or false and that can use logical operators such as `==` (is equal to), `>` (is bigger than), `<` (is less than), or `!=` (is different).

In a more complex format, the if control structure can even include a block of instructions to be executed in case the condition evaluates as false, in which case, its syntax is as follows:

```java
if (condition) {
    instructions to be executed in case the condition evaluates as true
} else {
    instructions to be executed in case the condition evaluates as false
}
```

As you can see in the code for our example, this format is the one we have used to get two different reactions on the pressing of the button:

- If the `pressed == LOW` condition is true, being pressed the value read from the digital input and meaning no press on the button, we execute a simple blink
- In the case of the condition being false, which means that there has been a press on the button, we execute the block of instructions contained in the else branch of the code

Timing and debouncing

If you play a little with the previous circuit once programmed, you can easily conclude that the reaction to the button press is not immediate, and it is due to the fact that in the code, the button press is only read once the reaction, blinking or fading, has concluded and both of them take their time, even if short.

In Chapter 9, Dealing with Interrupts, we will see another way to react immediately to these kinds of events through the use of a very powerful procedure called interrupts.

There is another final consideration with respect to this example that is called debouncing, and that is due to the physical way in which electrical contacts are made when pressing a button.

Although it’s hard to imagine, when you press a button, the state oscillates between on and off before the contact settles down, which can lead to incorrect readings of a sensor in the code.

You can learn more about switches and debouncing them on the Wikipedia page for switches at http://en.wikipedia.org/wiki/Switch.

The Arduino site also has a very interesting page exclusively dedicated to buttons debouncing at http://arduino.cc/en/pmwiki.php?n=Tutorial/Debounce, which deserves a reading.

Other types of digital sensors

Switches are perhaps the most used devices as digital input sensors but obviously not the only ones. Any device that can open or close a circuit branch can be easily configured and connected to Arduino to act like a digital input sensor, from a reed relay (http://en.wikipedia.org/wiki/Reed_relay) to a PIR motion detector (http://en.wikipedia.org/wiki/Passive_infrared_sensor).

One of those devices that are very cheap and easy to acquire are optocouplers or, as they are sometimes referred to, optical switches.

An optocoupler is a very simple device that has two parts:

- **An infrared light emitter**: This is a simple infrared LED in a housing that directs its light emission in a precise direction.

- **A phototransistor**: This is a special kind of transistor whose base is activated by the reception of light, infrared light in this case, and also in a special case that ensures that the phototransistor gets excited only with light coming from a concrete direction—that of the infrared LED on the other part of the housing.
In the following couple of images, you can see the schematic of the internal structure of a typical optocoupler and a real picture of the one I used for my own assembly of the next project:

![Optocoupler](image)

The internal structure of an optocoupler and a real one

If you take a closer look at the picture, you may even notice the indications on the plastic housing of the emitter part with its diode sign to reference the anode and cathode and the collector and emitter markings on the phototransistor part.

If we connect the phototransistor part to our Arduino board, we could easily detect whether it is being excited or not, thus indicating whether there is something placed just in the middle of the plastic housing or not.

This was the same principle that used the first mechanical computer mice to detect and count the X and Y movement, and for that, they used a very recognizable small slotted wheel that allowed for its internal microcontroller to account for every step the mouse took in every axis.

[If you are too young, perhaps you have never seen one of this kind of mice, but you could take a look at the Wikipedia entry for mouse under the Mechanical mice section at http://en.wikipedia.org/wiki/Mouse_(computing).]

In the next example, we will use an optocoupler as a light barrier that when interrupted, triggers a digital input.
Using an optocoupler as a coin detector

If we take advantage of the fact that there is a small space between the emitter and receptor part of a typical optocoupler that allows a coin to pass through, we could easily use one of them to create a simple coin detector that could be used for a bigger project—who knows, perhaps a candy vending machine or your next arcade game cabinet.

From the point of view of the circuit, we will have to provide little additional circuitry—only a pair of resistors: one to limit the current through the infrared LED and another to act as a load for the phototransistor between its emitter and collector.

Regarding the reaction we will get once the coin has been detected, I’ll simply turn on an LED for the sake of simplicity in this example project and to allow you to focus on the digital input side of the circuit and corresponding code.

The schematic of the coin detector

In the following diagram, you can see the complete schematic for the circuit we will use in this example:

![The complete circuit for an optical coin detector](image-url)
As you can see, from the optocoupler side, I have only added the two resistors I mentioned previously: a 10K to act as a load for the phototransistor and a 470 Ohms one to limit the current through the infrared LED.

There is just an additional consideration. If you take a closer look, you could notice that in this case, I have connected the LED that is going to be used as the output side of our project in a different way than in other examples in the book.

In this case, I haven't connected the LED from the Arduino pin to ground but just the opposite way, that is, from $5V$ to the Arduino pin, just to show you that you can also operate the LED in a negative way. This means that setting the Arduino pin to HIGH won't allow any current to flow through the LED, and setting the pin to LOW will allow approximately 10 mA to flow through the LED and into the Arduino board.

I've done it this way just to show you that it is not always necessary to provide current to output devices, but they can also be powered from the V+ voltage source and use the Arduino pin as a virtual ground as long as the total current doesn't exceed the 20 mA limit of the total current capacity of any Arduino pin.

**The breadboard connections diagram**

Regarding the breadboard connections, they are a little more complex this time due to the fact that the optocoupler housing of the Fritzing part I used was a little too big for the breadboard, so I've had to use some extra wires.

That apart, please keep in mind that the exact optocoupler you may find might not have the same pin distribution as the one used in this diagram, and you should refer to your specific component datasheet to adapt the wiring connections to your optocoupler or you could easily damage it.
In the optocoupler used in the diagram, the upper connections correspond to the phototransistor and the lower ones to the infrared LED emitter.
The complete example code

There is nothing really new in the code that we will use for this example. We will, just like in the previous one, simply read the digital input corresponding to the phototransistor and only if it is a HIGH value, which means it is not receiving any light and thus not conducting, will we make the output LED blink.

Once again, just for the sake of simplicity, and given that now we know that we don’t really need to store the value read with digitalRead() and we can simply call the functions inside the if condition, I have opted this time to save up this variable and simplify even a little more of the code:

```c
/*
   Chapter 05 - Sensing the real world through digital inputs
   Optical coin detector
   By Francis Perea for Packt Publishing
*/

// Global variables we will use
int led = 13;
int phototransistor = 8;

// Configuration of the board: one output and one input
void setup() {
    pinMode(led, OUTPUT);
    pinMode(phototransistor, INPUT);
}

// Sketch execution loop
void loop() {
    // We read the optocoupler pin
    // and if the phototransistor doesn't receive light keep blinking
    if (digitalRead(phototransistor) == HIGH) {
        // simply blink
        digitalWrite(led, LOW);
        delay(50);
        digitalWrite(led, HIGH);
        delay(50);
    }
}
```
A real working project

Finally, here you have a real picture of the complete circuit assembled with the optocoupler I had and a coin being detected:

![A complete assembly detecting a coin](image-url)
Summary

In this chapter, we had an introduction to the use of digital inputs by working with two different examples, each one showing different devices that can be used as digital sensors.

We saw what a momentary push button is and its particularities and also learned the right way to connect one to Arduino to be used as a digital input.

We also saw that there are lots of other devices that can be used as digital sensors and, in a second example, we used an optocoupler as an optical switch to build a coin detector.

From the point of view of programming, we met the if control structure in its two versions, simple and with a negative else branch, and learned that it can be the basis of any logical decision in our code.

In the next chapter, we are going to dive into the amazing world of analog sensors and meet two of them that will allow us to measure physical variables, which will enrich the field of application of our projects.

We will, of course, also learn how to program them and learn how to constrain their values to valid ranges in an easy way and multiply the possible execution branches of our code.

So, roll up your sleeves because this is going to get interesting.
Get more information Arduino Essentials

Where to buy this book
You can buy Arduino Essentials from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.