



ARDUINO

∞

THE ULTIMATE BEGINNER'S
GUIDE TO LEARN ARDUINO.

ARDUINO

∞

TIPS AND TRICKS TO
LEARN ARDUINO QUICKLY
AND EFFICIENTLY

ARDUINO

∞

SIMPLE AND EFFECTIVE
STRATEGIES TO LEARN
ARDUINO PROGRAMMING

ARDUINO

∞

BEST PRACTICES TO
LEARN AND EXECUTE
ARDUINO PROGRAMMING

ARDUINO

∞

ADVANCED STRATEGIES TO
LEARN AND EXECUTE
ARDUINO PROGRAMMING

DANIEL JONES

DANIEL JONES

DANIEL JONES

DANIEL JONES

DANIEL JONES

ARDUINO

← Copyright 2018 by Daniel Jones All rights reserved.

The follow eBook is reproduced below with the goal of providing information that is as accurate and reliable as possible. Regardless, purchasing this eBook can be seen as consent to the fact that both the publisher and the author of this book are in no way experts on the topics discussed within and that any recommendations or suggestions that are made herein are for entertainment purposes only. Professionals should be consulted as needed prior to undertaking any of the action endorsed herein.

This declaration is deemed fair and valid by both the American Bar Association and the Committee of Publishers Association and is legally binding throughout the United States.

Furthermore, the transmission, duplication or reproduction of any of the following work including specific information will be considered an illegal act irrespective of if it is done electronically or in print. This extends to creating a secondary or tertiary copy of the work or a recorded copy and is only allowed with express written consent from the Publisher. All additional right reserved.

The information in the following pages is broadly considered to be a truthful and accurate account of facts and as such any inattention, use or misuse of the information in question by the reader will render any resulting actions solely under their purview. There are no scenarios in which the publisher or the original author of this work can be in any fashion deemed liable for any hardship or damages that may befall them after undertaking information described herein.

Additionally, the information in the following pages is intended only for informational purposes and should thus be thought of as universal. As befitting its nature, it is presented without assurance regarding its prolonged validity or interim quality. Trademarks that are mentioned are done without written consent and can in no way be considered an endorsement from the trademark holder.

TABLE OF CONTENTS

ARDUINO

The Ultimate Beginner's Guide to Learn Arduino

Introduction

Chapter One : What is Arduino?

Chapter Two : Hardware, Software, and Applications

Chapter Three : Data Types Found in Arduino

Chapter Four : Local and Global Variables

Chapter Five : Operators Used in Arduino

Chapter Six : Arduino Control Statements

Chapter Seven : Arduino Loops

Chapter Eight: Functions in Arduino and How They Work

Chapter Nine : Strings in Arduino

Chapter Ten : String Objects

Chapter Eleven : How to Use Time in Arduino

Chapter Twelve : Arduino Arrays

Chapter Thirteen : Input and Output Functions Found in Arduino

Chapter Fourteen : How PowerShell is Different from Other Shells

Conclusion

ARDUINO

Tips and Tricks to Learn Arduino Quickly and Efficiently

Introduction

Chapter 1 : Arduino UNO

Chapter 2 : How to Install Libraries

Chapter 3 : Tricks for the Bootloader

Chapter 4 : Upgrading Arduino to the Latest Version of Atmega328P Chip

Chapter 5 : Conversion to 3.3V Devices

[Chapter 6 : Tricks for Maximizing Arduino](#)

[Chapter 7 : Arduino Module: Tricks and Tips](#)

[Chapter 8 : ArduinoISP](#)

[Summary](#)

[Conclusion](#)

[ARDUINO](#)

[*Simple and Effective Strategies to Learn Arduino Programming*](#)

[Introduction](#)

[Chapter 1 : Introduction to Arduino](#)

[Chapter 2 : Light the World](#)

[Chapter 3 : Programming](#)

[Chapter 4 : Graphics](#)

[Chapter 5 : References in Arduino](#)

[Conclusion](#)

[ARDUINO](#)

[*Best Practices to Learn and Execute Arduino Programming*](#)

[Introduction](#)

[Chapter 1 : Learning What Arduino Is](#)

[Chapter 2 : Hardware, Software, and Applications](#)

[Chapter 3 : The Data Types You Will Find in Arduino](#)

[Chapter 4 : Constants – Local and Global](#)

[Chapter 5 : Operators](#)

[Chapter 6 : Control Declarations](#)

[Chapter 7 : Loops in Arduino](#)

[Chapter 8 : Arduino Functions](#)

[Chapter 9 : Arduino Strings](#)

[Chapter 10 : String Objects for Arduino](#)
[Chapter 11 : Time with Arduino](#)
[Chapter 12 : Arrays with Arduino](#)
[Chapter 13 : Sensors](#)
[Chapter 14 : The Best Practices for Arduino](#)
[Conclusion](#)

[ARDUINO](#)

[*Advanced Strategies to Learn and Execute Arduino Programming*](#)

[Introduction](#)

[Chapter One : The Arduino Due and the Arduino Zero](#)

[Chapter Two : The Pulse Width Modulation](#)

[Chapter Three : Calculated Digital Representations](#)

[Chapter Four : Interrupts](#)

[Chapter Five : Communication](#)

[Chapter Six : Inter-Integrated Circuit](#)

[Chapter Seven : Serial Peripheral Interface](#)

[Chapter Eight : Advanced Strategies for Arduino](#)

[Chapter Nine : Humidity Sensor](#)

[Chapter Ten : Temperature Sensor](#)

[Chapter Eleven : Water Detector and Sensor](#)

[Chapter Twelve : PIR Sensor](#)

[Chapter Thirteen : Ultrasonic Sensor](#)

[Chapter Fourteen : Connecting Switch](#)

[Chapter Fifteen : Inputs and Outputs](#)

[Conclusion](#)

ARDUINO

***The Ultimate Beginner's
Guide to Learn Arduino***

DANIEL JONES

Introduction

The following chapters will discuss what you need to understand Arduino and everything that has to go with it. You are going to learn what you need to learn in order to be able to use Arduino in an efficient manner that is going to make it to where when you need to use Arduino on a daily basis, you are going to know how to use it.

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, and please enjoy!

Chapter One

What is Arduino?

Arduino is a computer company that works in both the software and hardware that a computer uses. The community that uses Arduino is going to be the ones that design and manufacture the microcontrollers and microcontroller kits that are located on a single board.

The Arduino boards are used to build digital devices as well as other interactive objects that are going to be able to control things in the physical world around us. These products are going to be using an open source hardware as well as an open sourced software that has been licensed under the LGPL (GNU Lesser General Public License) as well as the GPL (GNU General Public License) so that the boards can be manufactured and distributed to the public. You can find an Arduino board either preassembled or in a kit that will allow you to build it yourself.

The design of the board is going to vary based on what sort of controllers or microprocessors it will contain. The board is going to be equipped with both digital and analog input and output pins that are going to allow the Arduino board to interact with expansion boards and even other circuits. There is a feature on the Arduino board that allows the communication interfaces to be used. This includes the use of USBs on a few of the Arduino models which will allow for programs to be loaded off of a personal computer.

Arduino's microcontrollers are going to be programmed to use a dialect that is going to use the programming language of C as well as C++. It also uses a compiler toolchain that is more traditional in order to provide an integrated development environment (IDE) that is going to be based on the programming language that it works with.

The Arduino project began in 2003 in Italy initially as a program for the students that were attending the Interaction Design Institute. The purpose of the project was to try and provide an easy and cost-effective way for a novice as well as a professional to be able to create a device that would allow them to interact with the environment around them through the use of actuators and sensors.

Some of the most typical examples of which the devices interact are things such as a thermostat, a motion detector, and even robots.

Arduinos name came from one of the bars that can be found in Ivrea, Italy in which the founders would meet to discuss their ideas. This bar was named after the March of Ivrea and the King of Italy back in 1002 to 1014.

Chapter Two

Hardware, Software, and Applications

Being that Arduino is an open source hardware, there are going to reference designs to the hardware that are going to be sent out through the Creative Commons Attribution-Share Alike License which also means that they are available on the Arduino website. The layout and even the production files are going to be available depending on which version you are working with.

Even the source code has been released to the public because of the GNU General Public License. However, an official Bill of Materials for the boards has not and probably will not be published by the staff that works with Arduino.

Despite the fact that the designs for the hardware and software are free, the name Arduino is strictly exclusive to the boards in order to make sure that it cannot be used for any other boards without permission from the company. In fact, the documentation that is in place for the Arduino name specifically says that the project is open to working with others to incorporate them into the official product.

Several products are compatible with Arduino that were released commercially so that they could avoid having other products out on the market with the suffix -duino. A vast majority of the boards are going to have an Atmel 8-bit AVR microcontroller. Which consists of different amounts of pins, flash memory, and various other features. If you are using the 32-bit board, then it will work with an Atmel SAM3X8E which was introduced back in 2012. All of the boards are going to use a single or double row of pins that are going to facilitate the connections needed so programs and other circuits can work with it properly.

The boards that you are able to purchase are:

1. Arduino RS232
2. Arduino Diecimila
3. Arduino Duemilanove
4. Arduino Uno R2

5. Arduino Uno SMD R3
6. Arduino Leonardo
7. Arduino Pro
8. Arduino Mega
9. Arduino Nano
10. Arduino LilyPad 00
11. Arduino Robot
12. Arduino Esplora
13. Arduino Ethernet
14. Arduino Yun
15. Arduino Due

Shields

Any Arduino or devices that are compatible will use a circuit expansion board known as a shield that will be able to be plugged into the pin heads. A shield is going to be able to provide the controls that are needed for 3D printing and other applications such as GPS or liquid crystal displays. You even have the option of making your shield.

Software

Any program that is written for Arduino is going to be written in any programming language that you want it to be written in as long as it is able to be produced by a binary machine code for the target processor. The Atmel is going to give you a development environment that will work with the microcontrollers.

There is an integrated development environment that is going to be provided by the Arduino project in order to cross-platform how applications are written inside of Java. This started with the IDE for languages that will do the processing and the wiring. It also has a code editor that will allow you to cut text, paste it, search and replace it. There is also syntax highlighting, automatic indenting, and brace matching that will be there to provide one click mechanisms in order to compile and upload programs to the board. There is a message area as well as a text console, toolbars, and other buttons that are going to aid in improving the functionality of the board.

There are two primary functions that Arduino is going to use.

1. Loop (): this is done after the setup function is called. This function is going to keep being carried out in the main program. This will be what controls the board until you turn the board off or restart it.
2. Setup (): as soon as the sketch has started this function is going to be called. You are going to use it to initialize variables and other libraries that are needed for the sketch to work.

Applications

These are just several of the apps that are going to work with the Arduino boards.

1. A sensor system that will be used to detect bovine milk adulteration
2. Xoscillo – this is an open sourced oscilloscope.
3. A low-cost data glove that will work with virtual reality applications
4. Arduinome – this is a device that is going to mimic the Monome through a MIDI controller.
5. An automatic titration system that is going to work with Arduino as well as a stepper motor.
6. OBDuino – a computer that will be used for on-board diagnostics that you do to most of the modern-day cars.
7. Testing the quality of water
8. Ardupilot – this software is for a drone that works with Arduino
9. The Arduino phone which is a phone that you can build yourself.
10. Gameduino – this shield will allow for you to recreate the 2D retro video games

Other applications are going to work with Arduino, and you are going to be able to build your own applications with the board if you so desire to. You do not necessarily have to sell them to the Arduino name, you can use them just for yourself, but by working with the Arduino company, you are going to be getting your applications out to more people.

Chapter Three

Data Types Found in Arduino

The data types for Arduino are going to be written in C and will refer to an extensive system that you are going to use in order to declare functions or variables. The variable types are going to be what determines how much space that variable is going to occupy on the storage as well as how the bit pattern is going to be interpreted.

The data types that you will use with Arduino are:

1. String-char array
2. String- object
3. Word
4. Unsigned int
5. Int
6. Byte
7. Unsigned char
8. Char
9. Boolean
10. Void
11. Long
12. Unsigned long
13. Short
14. Float
15. Double
16. Array

Void

This keyword will be used when you are working with function declarations. As the word indicates, the function is going to be returned with no information whenever it is called.

Example

```
Void loop () {
```

```
# this is where all of the rest of your code is going to be  
}
```

Boolean

The Boolean is going to consist of two different values. It is either going to be true, or it is going to be false. For every Boolean variable that is found, one byte of memory is going to be used.

Example

```
Boolean var = false; # your variable declaration will be here and then  
initialized as false
```

```
Boolean state = state; # your variable declaration will be here and then  
initialized as true.
```

Char

This data type is going to take up at least one byte of memory for every character type that needs to be stored. The character literals are going to use single quotes while the double characters are going to use double quotes. Even though the numbers are going to be stored as numbers, you will see how the encoding is particular when it comes to the ASCII chart. What this means is that you can do arithmetic operations with the characters that are found in the ASCII value.

Example

```
Char chr_z = 'z' # the variable will be declared with the char type, and then it  
will be initialized with the z character.
```

```
Char chr_M = 77 # the variable is declared here and then initialized with the  
character 77 because 77 is the value that is assigned to M on the ASCII chart.
```

Unsigned char

The unsigned char is going to be a data type that is unsigned and is going to occupy one byte of memory. The data types for the unsigned char are going to fall between the values of 0 and 255.

Example

```
Unsigned char chr_m = 24 #the variable is going to be declared with this  
unsigned char before initializing it with your m character
```

Byte

The byte is going to store up to 8-bits of unsigned numbers which will be from the value of 0 to 255.

Example

Byte l = 4 # the variable is going to be declared for the byte type and then initialize it with the number 4.

Int

An integer is going to be primary data type for the numbers that are stored in on the memory. Integers are going to be stored in 2-byte values up to 16- bit. The range for an integer is -32,768 to 32, 767 which means that the minimum value is -2^{15} while the maximum value is $(2^{15}) - 1$.

The integer size is going to vary based on which board you are going to be using. The Arduino Due is going to allow for integers to be stored on 4-byte values. The range will be in the range of -2, 147, 483, 648 all the way to 2, 147, 483, 647. The minimum value is going to be -2^{31} while the maximum value is $(2^{31}) - 1$.

Example

Int counter = 43 # the variable of in will be declared and then initialized with the number 43.

Unsigned int

The unsigned integers are going to work the same way that a regular integer is going to, but they are going to store a 2-byte value. Rather than storing negative numbers, they are only going to store positive values. The range is going to be 0 to 65, 535 $(2^{16}) - 1$. The Arduino Due is going to store 4 bytes which will range from 0 to 4, 294, 967, 295 $(2^{32} - 1)$

Example

Unsigned int counter = 109 # the variable of the unsigned int will be declared and then initialized with 109.

Word

If you are using ATMEGA boards, the word is going to be stored to a 16- bit number that is unsigned. When using the Due and Zero, it will be 32-bits of an unsigned number.

Example

Word a = 293 # the declaration for the word type will be here before it is initialized with 293.

Long

The long variables are going to be the variables that have been extended in size so that they are able to store more in the number storage. They will save 4 bytes and be inside the range of -2,147,483,648 to 2,147,483,647.

Example

Long = 103843 # the variable will be declared with the long variable type and then initialize it with 103843.

Unsigned long

The long variables that are unsigned are going to be variables that are extended in size for the number storage which will store it in 32 bits. However, unlike a standard long variable, the unsigned ones are not going to be storing the negative numbers, they will only work with positive numbers which will range from 0 to 4,294,967,295 ($2^{32} - 1$).

Example

Unsigned long = 1093773 # the variable will be declared with the unsigned long type and then initialized with the number 1093773.

Short

The short will be a data type that is 16 bits. On all of the ATmega and ARMbased boards are going to use a 16-bit value for the storage. This causes a range of -32,768 to 32,767. The minimum value will be -2^{15} and a maximum value of $(2^{15}) - 1$.

Example

Short var = 32 # the short variable will be declared before initializing with the value of 32.

Float

The floating-point number data type is going to be the decimal point numbers. The floating-point numbers are usually used in order to approximate the analog as well as the constant values due to the fact that they have a greater resolution than integers.

The floating-point number has the ability to be as big as 3.4028235×10^{38} while it can be as low as $3.4028235 \times 10^{-38}$. A float will be stored as 4 bytes of information in the memory bank.

Example

Float number = 3.234 #the variable of a float is going to be declared before being initialized with the number 3.234

Double

While using the ATMEGA boards, the double precision floats number is going to take up 8 bytes. Double implementation is going to be the exact same as a float, but there is not going to be any gain in the precision. When you use the Arduino Due, the double is going to have an 8-byte precision.

Chapter Four

Local and Global Variables

Variable scope

When using the C programming language, the variables are going to have a scope. This scope will be the region for the Arduino program as well as the three places that a variable will be declared. Variables are going to be declared

1. Outside of any function with global variables
2. Inside of a block that will be called local variables
3. Inside of the definition of the function's parameters which will be the formal parameters.

Local variables

The variables that are inside of a function will be declared of local variables. They are only going to be the statements that are found in the function or the code block. Local variables are not going to be able to work outside of the function.

Example

```
Void setup ()  
{  
  Void loop () {  
    Int e, a;  
    Int r; local variable declared here  
    E = 0;  
    A = 0; # this is where the initialization is going to actually start  
    R = 39;  
  }  
}
```

Global variables

The global variables are the variables that are defined outside of the function and are going to be located at the top of the program typically. The global variable is going to be able to hold the value that is assigned to it the entire life of your program.

Global variables are going to be accessed via any function and are going to be available throughout the whole program once it has been declared.

Example

```
Int L, E;
```

```
Float A = 0; the global variable has been declared here
```

```
Void setup () {
```

```
}
```

```
Void loop () {
```

```
Int e, a;
```

```
Int r; local variable declared here
```

```
E = 0;
```

```
A = 0; # this is where the initialization is going to actually start
```

```
R = 39;
```

```
}
```


Chapter Five

Operators Used in Arduino

Operators are going to be the symbols that are going to inform the compiler which mathematical or logical function needs to be carried out. The C programming language is going to be rich in operators that are built into the program. The operators that you will see are:

1. Compound operators
2. Arithmetic operators
3. Bitwise operators
4. Comparison operators
5. Boolean operators

Arithmetic operators

Arithmetic operators are the operators that are going to give you the math symbols that will be used when you are doing math in Arduino.

- Division (/): the numerator will be divided by the denominator example: Z / B
- The assignment operator (=): the value is going to cause the value on the right to be equal to the variable that is located on the left side of the equals sign. Example: $Z = B$
- Multiplication (*): both operands will be multiplied example: $Z * B$
- Addition (+): two operands are going to be added together example $Z + B$
- Subtraction (-): the second number in the expression is going to be subtracted from the first example $Z - B$
- Modulo (%): the modulus operator will give you the remainder after you have done division example $Z \% B$

Comparison operators

- Greater than or equal to (\geq): the value on the left will be checked to see if it is greater than or equal to the value that is found on the right. Should this be accurate, the condition will be marked as true example $Z \geq B$
- Equal to ($=$): the two numbers will be checked to see if they are equal.

If they are, then the condition is going to be true example: $Z == B$

- Less than or equal to (\leq): the value that is on the left will be checked to see if it is less than or equal to the value that is on the right. Should it be, then the condition will be true. Example: $Z \leq B$
- Not equal to (\neq): the two values are going to be checked to see if they are equal. If they are not then the condition is going to be considered true. Example: $Z \neq B$
- Greater than ($>$): the value on the left will be checked to see if it is greater than the value on the right. If it is, then the condition is true. Example: $Z > B$
- Less than ($<$): the value on the left will be checked to make sure it is less than the value on the right. If it is, then the condition will be marked as true. Example: $Z < B$

Boolean operators

- Not (!): this is the logical NOT operator. The logical state will be reversed. If the position is determined to be true, then the logical NOT will make it false. Example! ($Z \& \& B$)
- And (& &): the logical AND operator will check to see if the operands are non-zero. If this is found to be true, the condition is going to be true example: ($Z \& \& B$)
- Or (||): the logical OR operator will see if there are any non-zero operands. If there is one or more, then the condition will be considered as true.

Bitwise operators

- Shift left ($<<$): a binary left shift. The left value is going to be shifted left by the number of bits that the right operand specifies. Example $Z << 3$
- And (&): binary AND will copy the bit to the result that is found in both operands if any. Example $Z \& B$
- Not (~): binary one's complement operator will flip the bits. ($\sim Z$)
- Or (|): binary OR is going to copy the bit that exists in one or both operators. Example ($Z | B$)
- XOr (^): the binary XOR will copy the bit only if it is in one of the

operands, it cannot be found in both. Example $(Z \wedge B)$

- Shift right ($>>$): the left operand will be moved to the right by the number of bits that are specified by the number on the right. Example $Z >> 4$

Compound operators

- Compound division ($/=$): divide AND assign operator. The left operand will be divided by the right and then assigned to the left operand. Example $Z /= A$ which is equivalent to $Z = Z / B$
- Increment ($++$): the integer will be increased by one value each time. Example $Z ++$
- Compound multiplication ($*=$): the multiple AND assignment operator will multiply the right number with the left one and then assign it to the left operand. Example $Z* = B$ which is equivalent to $Z = Z * B$
- Decrement ($--$): the operand will decrease the integer by one each time example $Z --$
- Compound subtraction ($-=$): The Subtract AND assignment operator is going to subtract the right number from the left and then assign it to the left. Example $Z - = B$
- Compound addition ($+=$): the right operand will be added to the left and then assigned to the left operand example $Z += B$
- Compound bitwise and ($\&=$): bitwise AND assignment example $Z \& = 4$
- Compound modulo ($\%=$): the modulus will be taken with the two operands and then assigning it to the left operand example $Z \% = A$
- Compound bitwise or ($|=$): bitwise inclusive OR and assignment example $Z |= 3$

Chapter Six

Arduino Control Statements

The structure that is going to be used for making decisions is going to mean that you are going to have to specify one or more conditions that have the ability to be evaluated by the program. This has to be along with any statements that you put into the program that have to be executed in order to see if the condition is true. Any other statements can be executed in the event that the condition turns out to be false.

Control statements

1. 'If' statement: the expression that is located in the parenthesis or a block of declarations. Should the expression be evaluated as true, then the statement is going to be executed; otherwise, the statement will be skipped.

Syntax:

Form A:

```
If (expression)
    Statement;
```

Form B:

```
If (expression) {
    Block of statements
}
```

2. 'If...else' statement: the 'if' statement has the ability to be followed by an else statement that will be executed whenever the expression is found to be false.

Syntax:

```
If (expression) {
    Block of statements;
}
Else {
    Block of statements;
```

}

3. 'If... else if... else' statement: just like the 'if' statement can be followed by an 'else' statement, it can also be followed by an 'else if... else' statement which will be used whenever you are testing a variety of conditions through the utilization of a single 'if... else if' statement

Syntax:

```
If (expression_1) {  
    Block of statements;  
}  
Else if (expression_2) {  
    Block of statements;  
}  
Else {  
    Block of statements;  
}
```

4. Switch case statement: this is similar to the 'if' statement, it is going to control the flow of the programs that will allow you the programmer to specify the various codes that have to be executed inside of the conditions.

Syntax:

```
Switch (variable) {  
    Case label:  
        //statements  
    Break;  
}  
Case label: {  
    // statements  
    Break ;  
}  
Default: {
```

```
    //statements  
    Break;  
}
```

5. Conditional operator: the provisional operator is the only ternary operator that you are going to find in the C program.

Syntax:

Expression 1 ? expression 2: expression 3

Chapter Seven

Arduino Loops

Many programming languages are going to give you controlled structures that are going to enable a more complicated execution path to be carried out. Loop statements are the statements that enable the group of statements to be executed multiple times and is going to follow the general set up that you are going to see for a loop statement in most programming languages.

Several different loop types are going to be processed with C programming.

1. 'while' loop: the while loop is going to consistently run the expression that is placed inside of the parentheses until the condition is found to be false. Something is going to have to change such as the tested variable or else the loop is never going to end.

Syntax:

```
While (expression) {  
    Block of statements;  
}
```

2. 'do...while' loop: 'do...while' loops are going to operate like a while loop. The while loop is going to continually run until the condition that is tested at the top of the loop is met before the body of the loop is executed.

Syntax:

```
Do {  
    Block of statements;  
}  
While (expression) ;
```

3. 'for' loop: for loops are going to take the statements and carry them out a predetermined number of times. Control expressions for the loop will be initialized, tested, and even manipulated within the parentheses for the loop.

Syntax:

```
For (initialize; control; increment or decrement) {  
    // statement block  
}
```

Example

```
For (number = 3; control <= 32; number++) {  
    // your block of statements is going to be carried out 33 times.  
}
```

4. 'nested' loop: C programming enables more than one loop to be performed. This is considered to be nested loops because you are going to be placing one loop inside of another loop.

Syntax:

```
For (initialize; control; increment or decrement) {  
    // statement block  
For (initialize; control; increment or decrement) {  
    // statement block  
    }  
}
```

Example

```
For (start = 3; control <= 2; increment++) {  
    // the statements here are going to be executed 3 times  
For (s = 9 s <= 5; s++) {  
    //these comments will be carried out 6 times  
    }  
}
```

5. Infinite loop: this loop will have no condition that stops it from being run therefore the loop will run forever.

Syntax:

```
For (initialize; control; increment or decrement) {  
    // statement block  
For (initialize; control; increment or decrement) {  
    // statement block
```

}

}

Chapter Eight

Functions in Arduino and How They Work

A function is going to allow for you to structure the segments of your code in order to perform individual tasks. Most of the time you are going to create a function whenever you need one in order to perform the same action several times over in the program.

Some of the advantages to standardizing code fragments in your functions:

1. Your function is going to assist the programmer in staying organized which will help with the conceptualization of the program.
2. Functions are going to be able to code a single action in one place in order for the function to be evaluated and debugged once.
3. The chances of an error coming up are going to be reduced in the event that the code has to be modified.
4. The functions will make your sketch smaller and more compact due to the fact that the blocks of code are going to be reused multiple times over.
5. Code is going to be easier to be reused in a different program which makes it modular and by using functions more often will make your code more readable.

The Arduino sketch is going to have two functions that it requires which we discussed earlier as the setup and the loop. Any other function is going to be created outside of the brackets of these two primary functions.

Syntax:

```
Return type function name e (argument1, argument 2, ...)  
{  
    Statements  
}
```

Function declaration

Functions are going to be declared outside of the other functions it will either

be placed above or below the loop function.

Functions can be declared in two ways. One is to write the function out known as the function prototype which will be placed above the loop function. The loop function is going to consist of these parts:

1. Function argument type. You will not have to write the argument name out.
2. Function return type
3. Function name

The prototype has to be followed by a semicolon!

Example:

```
Int add_func (int a, int e) //the function will be declared here {  
    Int u = 0;  
    U = a + e ;  
    Return u; //the value will be returned here.  
Void setup ( ) {  
    Statement block  
}  
Void loop ( ) {  
    Int outcome = 0 ;  
    Result = add_func ( 2, 4) ; //the function call  
}
```

The second method that you can do is to define the function. In order to declare your function, you have to declare your loops. Ensure that you have all of these parts for this method.

1. Function body: your statements are going to be listed inside of the function when it is being executed after it has been called.
2. Function return type
3. Function argument type: you must have the name of the argument
4. Function name

Example

```
Int add_func (int, int) ; //your function prototype
```

```
Void setup ( ) {  
    Group of statements  
}  
Void loop ( ) {  
    Int outcome = 0 ;  
    Result = add_func (3, 5) // the function call  
}  
Int ad_func (int a, int w) //the declaration of the function  
    Int o = 0;  
    O = a + w;  
    Return o //the value that needs to be answered.  
}
```


Chapter Nine

Strings in Arduino

The text that you enter in Arduino is typically going to be stored inside of a string. These strings can be displayed on an LCD or inside of the IDE serial monitor window. The strings are going to be useful when it comes to storing any input that comes from the user as well. An instance of this will be if a user is using a keypad that is connected to the Arduino to input answers to your questions.

Arduino is going to work off of two different string types.

1. The Arduino string that will allow the user to create a string object while in the sketch.
2. The character arrays which will be the same as any other string that you use in the C programming language.

String character arrays

This first string type is going to be used with a series of characters known as the char type. Arrays are consecutive series of the same variable type that is going to be stored in the memory. Strings are going to be arrays of char variables.

Strings are known as special arrays that are going to include an extra element which will be found at the end of the string, which is going always to be a zero value. This will be known as 'null terminated string.'

Example

```
Void setup () {  
  Char a_str [ 7 ]; // the array is going to have six characters in the string  
  Sequence. Start (8483);  
  A_str [0] = a  
  A_str [1] = e  
  A_str [2] = i  
  A_str [ 3 ] = o  
  A_str [4] = u  
  A_str [5] = y
```

```

A_str [ 6] = 0 // this is going to be the null terminator
Sequence. Println (a_str);
}
Void loop ( ) {
}

```

The example that you are about to see is going to show you what a string is actually made up of. The character array is going to contain printable characters as well as the zero that needs to be left in the last place of the array in order for the program to know where the string ends. The string can be printed out in the IDE serial monitor window through the use of the `serial.println ()` function before it passes through the name of the string.

Example

```

Void setup ( ) {
    Char a_str [ ] = "a,e,i,o,u";
    Sequence.start (3494);
    Sequence. Println (a_str);
}
Void loop ( ) {
}

```

The compiler is going to calculate the sketch on the size of the string array. It is also going to terminate the string with the zero automatically. Arrays that contain six elements are going to consist of five characters followed by the zero.

Manipulating string arrays

String arrays can be altered inside of the sketch

Example

```

Void setup ( ) {
    Char similar [ ] = "I am a fan of vanilla ice cream"; //the string is going
    to be created here.
    Sequence. Start (4838);
    // (1) the string will be printed
    Sequence. Println (similar) ;
}

```

```

        // (2) now part of the string has to be deleted.
Similar [4] = 0
Sequence. Println (similar)
        // (3) a new word needs to be substituted into the string
Similar [5] = ' ' ; // the null terminator will be replaced with a space
Similar [ 6] = 'c' // this is where your new word is going to start
Similar [ 7] = 'o'
Similar [ 8 ] = 'k'
Similar [ 9 ] = 'e'
Similar [10] = 0 // your string is now going to be terminated here
Sequence. Println (similar);
    }
    Void loop ( ) {
    }

```

‘I like vanilla coke’

Your outcome is now going to contain your new word and your old word is not going to be in the string.

Creating and printing a string

With the sketch example you just saw, a new string is going to be created and printed into your serial monitor window.

Shortening your string

The string can be shortened by taking your fourteenth character in the string and place a zero in it. The element number is going to be the previous number if you start counting from zero.

Whenever your string is printed, every character is going to be printed out until the new null is reached. Any other characters are not going to disappear, they still have their place in the memory and inside of your string, they are just not seen. You are only going to see the string as it is up to the first null terminator.

Changing a word inside of the string

As you saw in the example, the word ice cream was replaced with the word coke. The first thing that had to happen was that the null terminator had to be

replaced with a space in order to create a new string from the original string.

Your new characters are going to write over the word that you are wanting to replace. The program will write over each character individually and the last space will be the new null terminator. It does not matter when your string is printed due to the fact that the function is going to print your new string until it meets the first terminator.

Functions used to manipulate string arrays

With the previous example, the string was manipulated, but it was manipulated manually. There is an easier way to manipulate a string! You will create your own function in order to complete this task. Or, there are functions you can use that are provided in the C library.

Example

```
Void setup() {
  Char str [ ] = "I have created a string"; //my string is created
  Char in_str [ 3] // this is the output from your string function
  Int num ; // your integer for general purpose
  Sequence. Start (8384) ;
  // (1) your string has to be printed
  Sequence.println(str) ;
  // (2) the length of the string needs to be obtained without the null
  terminator
  Num = strlen (str) ;
  Sequence. Print (" your string lenght is: " );
  Sequenc.println (num);
  // (3) the length of the array including the null terminator
  Num = sizeof (str) ; //the sizeof () is not going to be a string
  function for C programming language
  Sequence. Print ("the size of the array is: ");
  Sequence. Println(num) ;
  // (4) get a copy of your string
  Strcpy (in_str, str);
  Sequence. Println (in_str);
```

```

        // (5) add your string to the end of the other string
        Strcat (in_str, "sketch");
        Sequence. Println (in_str);
        Num = strlen (in_str);
        Sequence. Print ("the length of the string is: ");
        Sequence. Println (num) ;
        Num (sizeof (in_str);
        Sequence. Print ("the size of the array in_str [ ]: ");
        Sequence. Println (num) ;
    }
    Void loop ( ) {
    }

```

Print the string

Your new strings are always going to be printed into the serial monitor window as you have seen in the previous examples.

Getting the length of the string

You are going to use the `strlen ()` function to get the length of any array that you are working with. The length is going to be the printable characters only and it is not going to include your null terminator.

So, if a string consists of 3 characters, then there are going to be 3 characters printed into the serial monitor window.

Get length of array

Your `sizeof ()` operator is going to be used in getting the length of the array that is inside of the string. The length is going to include the null terminator which means that there is going to be one more length to the string.

`Sizeof ()` is going to appear to look like a function, however, it is actually an operator and it is not going to be considered part of the string library for the C language. However, being used in a sketch is going to show the difference between what the size of your array is and the size of the string is.

Copying strings

Using the `strcpy ()` function is going to be used when you are copying your string. This function will copy the second string as it passed into the first

string. This copy is now going to be inside of the array and will only take up as many elements as there are in the array. This is going to still leave you some char elements in your array. The free elements are going to be found in your string's memory.

When your string is copied, then there is actually going to be extra space in your array that you can use in the next part of the sketch which is going to end up adding the string to the end of a string.

Appending a string to string (concatenate)

Whenever the sketch joins a string to another string, this is going to be known as concatenation. The `strcat()` function will be used to complete this action. The `strcat()` function is going to put the second string and pass it to the end of the first one.

After you have completed concatenation, the length is going to be printed in order to show how long the new string is. The length of your array is going to be printed in order to show that you have a character string of x inside of an element that is x elements.

Array bounds

As you work with the strings and arrays, you need to also make sure you are working with the bounds for those strings and arrays. As you saw in the example above, there was an array created with 40 characters, so in order to allocate the memory, it is going to be used when you are manipulating the string.

Should the array not be made big enough, you are going to attempt to copy a string that is longer than the array it is tied to, once that string is copied over to the end of the array it should be longer. Any memory that goes beyond the end of your array is going to hold the other data that is going to be used by the sketch. However, it is going to be overwritten by the string. In the event that the memory is overrun, then the sketch is going to crash or give you unexpected results.

Chapter Ten

String Objects

An object is going to be constructed of both functions and data types. The string object can be created as if it was a variable and it is assigned to a value or to a string. String objects will contain functions called methods when you are working with object oriented programming which will have the ability to operate in the string data that is inside of the object.

Example

```
Void setup ( ) {  
    String a_str = "I created this string "  
    Sequence. Start (3848);  
    // (1) the string needs to be printed  
    Sequence.println (a_str);  
    // (2) the string will need to be changed to upper case  
    A_string.toUpperCase ( );  
    Sequence.println (a_str)  
    // (3) now overwrite your string  
    A_str = "This is the new string"  
    Sequence. Println (a_str)  
    // (5) obtain the length of your string  
    Sequence.print ("the length of the string: ");  
    Sequence. Println (a_str. Length ( ) );  
}  
Void loop ( ) {  
}
```

String objects are going to be created and will be able to be assigned to the strings or to values when they are placed at the top of the sketch.

Example

```
String a_str = "I created a string" ;
```

In this example, you are going to see that a string object has been created by using the a_str and the value is going to be "I created a string."

You can compare this to when you are creating variables and assigning them

to things such as integers.

Printing string

Like with any string that is printed, it is going to be printed in the serial monitor window.

Convert the string to uppercase

String objects that have a number of various functions are going to invoke the use of objects by their names and they will be followed by a dot before the name of the function is used.

Example

```
A_str.toUpperCase ( );
```

The function toUpperCase is going to take the string that you are working with and convert the data that is in the object to where it is using upper case characters. The list for the function is going to be the string class and this is going to be able to be found in the string reference. In all reality, the string is going to be a class and is going to be used in the creation of string objects.

Overwriting strings

You are going to use the assignment operator when you are assigning a new string to the object that needs to be replaced. However, assignment operators cannot be used with character array strings, therefore, it is going to work on a string object only.

Replacing words in a string

You will use the replace () function when you are replacing something in your string so that it is passed to the second string. Replace is going to be one of those functions that is built into the string class and can be used on a string object.

Getting the length

When you want to get the length of the string, you will use the length () function. As you saw in the example, your result is going to be returned in the same function after it has been passed to the serial.print (). And, it is not going to need to use an intermediate variable.

When you should use a string object

String objects are easier to use than trying to decide when you should use a

string character array. Objects are built into the function that is going to allow you to perform a vast number of operations with your string.

There is a disadvantage to using the string object just like there are disadvantages to anything and that is that string objects are going to use a lot of memory and fill it up quickly. The RAM memory is going to cause the program to hang up, behave unexpectedly, and even crash. Should a sketch that is on Arduino be small and have limited use of objects, then there should not be any problems. However, you may experience some.

A character array string as we stated is going to be harder to use and you may find that you are having to write out your own functions in order to operate with these sorts of strings. But, the biggest advantage is going to be that you can control the size of the string that you are creating, that way you can keep your arrays short and save on the memory.

Ensure that you are not writing beyond the end of your array. String objects are not going to give you this issue, because it is going to take care of the bounds for the string as long as there is enough memory that will allow it to operate. String objects are going to attempt to write on memory even if it does not exist. However, it is not going to be able to write over the end of the string that it is working with.

Chapter Eleven

How to Use Time in Arduino

There are four different manipulations that you are going to be using with Arduino.

1. Delay function: this function is going to accept single number arguments. The number is going to show the time which is going to be measured in milliseconds. Arduino should wait to go on to the next section of code whenever this function is found. But, the issue you are going to have with delay is that you should not make your program to wait because it is going to block your function.

Syntax:

Delay (ms) ; //ms is going to stand for milliseconds in order to create the pause. This is known as an unsigned long.

2. Delay microseconds function: the delay microseconds function is only going to accept single number arguments as well. This number is going to represent the time that is going to be measured in microseconds. If you do not already know, there are a thousand microseconds in one millisecond and a million microseconds in a second.

Syntax

Delaymicroseconds (us) ; // us is going to be the number of microseconds for the pause. This is an unsigned int.

3. Millis (): function: you are going to use this function whenever you want to return the number of milliseconds that are going to go back to when the board began running the program that you are currently working with. This is going to be an overflow number.

Syntax:

Millis () : //the function will give you the milliseconds from when the program first started.

4. Micros () function: the micro function is going to give you the number

of microseconds from the time that the program began. This is another overflow number. There is a resolution of eight microseconds on your 8 MHz boards.

Syntax:

Micros () : //this is going to be an unsigned long.

Chapter Twelve

Arduino Arrays

Arrays are going to be a group of memory locations that are found consecutively and are going to be the same type. In order to refer to a specific location or element that is found in the array, you are going to name the array and the position of that element.

Subscripts are going to have to be integers or expressions of an integer. Should a program use an expression in the subscript, then the Arduino program is going to evaluate the expression in order to determine if it is a subscript.

Example

The variable for z is going to be equal to seven while the variable of y is equal to one. Therefore the statement is going to have two different arrays with elements in two different places.

When an array's name is subscripted, then it is going to have a value that will be used on the left side of your assignment. This is similar to the non-array variable names.

When you are wanting to print out the sum of your values that you have in the first three elements of your array, then you are going to write it out with the variable on the outside of a closed set of brackets that will contain each of the numbers that need to be added together.

Declaring arrays

An array is obviously going to occupy space in the memory of the program. So, when you are wanting to specify what type of elements that are going to be in your array; you are also going to need to know how many elements you are putting in your array to make the declaration.

Syntax:

```
Type arrayName [ arraySize ] ;
```

Your compiler will set aside the proper amount of memory so that it can ensure you can save your array. The size of the array has to be an integer that is greater than zero.

Example

The compiler needs to know that you are placing five elements on your array, therefore, it will save enough room for five elements!

An array has the ability to be declared so that it can contain the values for the non-reference data types. This means that an array can be used to store character strings.

Important concepts

1. Passing arrays to functions: in an effort to pass the array's argument to the function, you will need to state the name of the array without using any brackets. So, if your array is named `babyfeedingtimes` then it will be the function that is declared, and the call is going to pass through the array. Its size is going to be to modify your array.

You need to keep these important points in mind when you are passing arrays to functions.

- When you are passing the array to your function, in normal passes, the array size is going to be passed on as well, this is done so the function has the ability to process the number of elements that are in the array. If it does not pass the size, then a function has to be called on and built, or the array size will need to be placed in a global variable.
- If you are using C++ the array will be passed to the function by a reference.
- The name of the array is going to be used in order for it to be saved on the memory of the computer. Being that the address that starts the array will be passed, the function that is called will know exactly where the array is on the memory and be able to modify the array and the elements in the array. All this will be done without moving the array from its original memory location.
- Entire arrays are going to be passed through references, however individual elements in the array are going to be passed by the value.
- When you want to pass elements of the array to the function, you will use the subscripted name for the array's element as your argument in your call.
- In order for the function to get your array, you will need to modify the

parameter list for the function so that it specifies that the function has the ability to accept the array.

The function prototype for modifying the array is going to look a little different than anything else that you are used to.

Syntax

```
Void modifyarray ( int [ ], int) ;
```

However, your prototype will write it like this just for documentation purposes

Syntax

```
Void modifyarray (int anyarrayname [ ] , int)
```

The compilers in C++ will ignore the name of the variables that are found in prototypes. Keep in mind that the prototype is going to be what tells the compiler how many arguments there are and what type of arguments there are so that the arguments are shown in the order they are supposed to be shown.

Chapter Thirteen

Input and Output Functions Found in Arduino

If you look at the pins on the Arduino board, you will see that they can be configured as an input or an output. You should remember that a lot of the analog pins have the possibility of needing to be configured and used in the same way that a digital pin is going to be used.

Input pin configuration

The pins are going to be set to input by default, therefore, they do not need to be declared as an input by using the `pinmode ()` function whenever you are wanting to use them as an input. The pins that are configured this way will be in a state of high impedance due to the fact that the input pins are only going to be made to make small demands on the circuit that they sample, which is going to be equal to the series resistor of a hundred megaohm for the front pin.

In other words, it is not going to take much current to switch the input from one state to the next which makes it useful for things like when you need to implement a touch sensor or when you are reading a LED as a photodiode.

The pins that are configured as `pinmode` will have nothing that is connected to them, if wires are connected to them, then they cannot be connected to another circuit. It has been reported that there are changes in the pins state, environmental noise picked up through the pins or the capacitively coupling in the state of a pin that is near the first pin.

Pull up resistors

The pull up resistor is going to be useful when you are needing to steer the input of a pin to a known state but you do not have any input present. This is going to be best when there is no input. All you need to do is add a pull up resistor that goes up to 5 volts or you can choose a pull down resistor for the input. It is best that you use a 10k resistor that is going to be good for pulling up or pulling down the resistor.

Using the built in pull up resistor with the pins configured to input

In the Atmega chip, there are around 20,000 pull up resistors that are built

into it that you are going to have access to in the software. These resistors are going to be accessed through your `pinmode ()` setting by inputting `input_pullup`. Now you have inverted the behavior of your input mode so that high will turn the sensor off and low will turn the sensor on.

The various values for the pull up are going to depend on what kind of microcontroller you are using. For many AVR boards, the value is going to be between 20 and 50 k ohms. For the Arduino Due, you will find that it is between 50k and 150 k ohms. In order to figure out what the exact value is, you will need to look at the datasheet for the microcontroller that is installed on your board.

Whenever you connect sensors to the pins that are configured for input, you need to ensure that the other end is grounded. This is done so that if the pin is reading high, the switch is going to be opened and low means that the switch is pressed. With pull up resistors, you are going to be able to provide enough current to light up the LED that is connected to the pin.

There are some registers that are going to tell the pin if it is on high or low while controlling the pull up resistor. There are also pins that can be configured in order to have the pull up resistor turned on whenever the pin is in input mode, which will mean that the pin is turned on to high. Should the pin be switched over to output by use of the `pinmode ()` function, then it is going to work the opposite direction. So, if the pin is on output mode, the high state is going to have the resistor set up to where if switched it will go into input mode.

Example

```
pinmode (4, input) ; // the pin is set to input
pinmode (6, input_pullup) ;
```

Pins configured to output

A pin that is configured to output with `pinmode ()` is going to be the low impedance state. With that being said, they are going to be able to provide a large amount of current to other circuits that are hooked up to it. The Atmega pins are going to give you the positive current or the negative current depending on how many milliamps of current the other devices are going to need. As long as it is 40 mA or under, you will be able to have enough current to light up a LED brightly, but it is not going to be enough to run any

motors.

Whenever you attempt to run a device that is going to require a lot of currents, the pins can become damaged or even destroyed. This can end up destroying the entire Atmega chip which is going to result in a dead pin in your microcontroller. However, the other pins are going to still work, but they may work at a lesser power than they were before. That is why you should hook your output pins up to another device that is either 470 ohms or is a 1k resistor. The only time that you should not is if the current draw that is coming from the pins is required to run a certain application.

Pinmode () function

Pin mode is going to be used whenever you are configuring a specific pin so that it is going to behave as an input or an output pin. There is the possibility that you can enable the internal pull up resistor through the input_pullup mode. It also makes it to where the input mode is going to disable any internal pull-ups.

Syntax:

```
Void setup ( ) {  
    Pinmode (pin, mode) ;  
}
```

1. Pin is going to be the number of pins that you are wanting to set the mode for.
2. Mode will be either input, output, or input_pullup.

Digitalwrite () function

This function is going to be used when you need to write the high or low value for your digital pin. Should the pin be configured for output, then the voltage needs to be set to the value of 5 volts. There cannot be any volts for low because it will need to be grounded.

If the pin is on input, then the high setting is going to be disabled while the low is going to be the internal pullup for the pin. It is highly recommended that you set your pinmode () function so that input_pullup will enable a pull up resistor inside of the board.

In the event that you do not set the pin mode for output and then proceed to connect a LED tot hat pin as you call on the high setting, then the LED is

going to appear dimmer than it should be. If you do not explicitly set the pin mode and digital write functions to enable the internal pullup, it is going to do so automatically which is going to act like a large current limiting resistor.

Syntax

```
Void loop ( ) {  
    digitalWrite (pin, value) ;  
}
```

1. Pin is going to be the number of pins that you are wanting to set to input or output.
2. Value is the high or low setting.

Analogread () function

The Arduino program is going to have the ability to detect if there is voltage being applied to one of the pins before reporting it through the digitalread () function. You have to know that there is a difference between the on and off sensor so that the analog sensor is constantly charging. To read this type of sensor, you are going to require a different type of pin.

When you look at the lower right of your board, there are going to be six pins that are marked as analog in. These pins are not just going to tell if there is any voltage being applied to them, but also how much is flowing through it. When you use the analogread () function, you will be able to read the voltage that is applied to just one of the pins.

For this function, a number is going to be returned between 0 and 1023 in order to represent the voltage between 0 and 5 volts. For example, if you have a voltage of 2.5 V that is being applied to the pin number 3, you will get a return of 512.

Syntax

```
Analogread ( pin) ;
```

1. Pin: the number of which pin to be read from 0 to 5 on a great majority of boards. 0 to 7 if you are using the mini and Nano boards. And then 0 to 15 on the Mega boards.

Chapter Fourteen

How PowerShell is Different from Other Shells

The sensors are going to interact with the world around you.

Humidity sensor (DHT22)

The DHT22 sensor is going to have the digital out that will sense the humidity and the temperature around it. It is going to be used when capturing the humidity sensor and a thermistor in order to measure the air that is around the sensor which will then send a signal to the data pin. The connections on the humidity sensor is going to be simple to understand.

The sensors are going to interact with the world around you.

Humidity sensor (DHT22)

The DHT22 sensor is going to have the digital out that will sense the humidity and the temperature around it. It is going to be used when capturing the humidity sensor and a thermistor in order to measure the air that is around the sensor which will then send a signal to the data pin. The connections on the humidity sensor is going to be simple to understand.

The first pin on the left is going to be 3 to 5 volts of power while the second is the data input pin and the pin that is on the right is going to be the ground.

Technical details

- Temperature: 40 to 80 C
- Power: 3 – 5 Volts
- Humidity: 0 – 100 %
- Max current: 2.5mA

Components required

- 10 K ohm resistor
- Breadboard
- DHT22
- Arduino Uno R3

Note

There are four terminals for the humidity sensor and they are going to be connected to the board as follows:

- The data pin is pin 2
- The V_{cc} pin which is going to have 5 volts of energy
- The GND pin or the ground
- The resistor that will be connected to the data and V_{cc} pins

After everything is hooked up, the DHT22 can be added to your library files as code that you have already used.

Temperature sensor

This sensor is in the LM 35 series and it is going to use an output of voltage that is going to be proportionate to the Centigrade temperature. This device is going to be better than a linear temperature sensor that is calibrated to Kelvin because you do not have to subtract voltage from your output in order to get the centigrade scaling you need. Also, this sensor is not going to have to require any extra calibration or trimming in order to provide the proper accuracies.

Technical specification

- The sensor is suitable for the more remote applications
- Calibrated directly for the centigrade
- Rated to read between -55 C to 150 C
- Linear scale factor
- 0.5 C to ensure accuracy

Components required

- LM35 sensor
- Breadboard
- Arduino Uno R3 board

Notes:

There are three terminals for the temperature sensor

- The +Vs is going to be connected to the Arduino board with 5 V
- The Vout will be connected to A0 on the board

- The GND will be connected to the GND on the Arduino board.

The temperature is going to be displayed on a port monitor which is going to be updated every second.

Water detector/sensor

This water sensor is going to be designed specifically for water detection in any body of water, and it does not even have to be a body of water, it can be liquid leakage. When the water sensor is connected to the Arduino, then you will be able to detect leaks, spills, floods, anything that has to do with water. It is going to be used to not only measure the volume of the water, but the absence of water. You can use it in order to remind yourself to water your plants. It is going to work better than most other sensors since it is going to be able to expose any traces of water and read when the water is detected and what level it is at.

Components

- 330 Ohm resistor
- Breadboard
- Led
- Arduino Uno R3
- Water sensor

Notes

There are three terminals for the water sensor.

- The Vs has to be connected with 5 volts
- The S digital pin will be connected to pin number eight.
- The GND will be connected with the GND
- The LED is going to be connected to digital pin number nine.

Whenever the sensor detects the presence of water then pin 8 will be switched to low and the LED will be turned on.

PIR sensor

This sensor will detect motion as long as something moves inside or outside of the range of the sensor. They are usually found in appliances or gadgets used for security. PIR is known as Passive Infrared, Pyroelectric or IR

motion.

The advantages of a PIR sensor is that they are

- Not going to wear out
- They are small
- Easy to use
- There is a wide lens range
- Low power
- Easy to program
- Inexpensive

Components needed

- Breadboard
- PIR Sensor preferably MQ3
- Arduino Uno R3

Notes

This sensor also has three terminals

- The Vcc will be connected to 5 volts on the board
- The out will be connected to the digital pin number 2
- The GND will be connected to the GND

After you have set it up, the sensor will be able to be adjusted for sensitivity and delay time. These two resistors are going to be located at the bottom of your board. After the sensor has detected motion, a message will be sent to you so that you know that motion has been detected which is going to make it to where you will know if you need to do something about it, or if you are okay. The motion sensor is great to see if your kids are home safe or to see if anyone is around your house when you are away on vacation.

Conclusion

Thank you for making it through to the end of *Arduino*, let's hope it was informative and able to provide you with all of the tools you need to achieve your goals whatever it may be.

The next step is using Arduino. If you know other programming languages, you are going to be able to add to your programming language. Not only that, but you are also going to be able to program the Arduino board so that it is able to interact with the world around you.

The world is limitless when it comes to Arduino because you are going to be able to program the Arduino board to interact with things like your thermostat. You may even be able to find something new to program to your Arduino board thanks to the ever expanding world of technology!

With Arduino, you will find that your life may become easier thanks to the fact that you will be able to keep your plants alive and even change the temperature of your house without ever going to a thermostat. While there are other applications out there that make these things possible, Arduino is going to be superior to the others.

Finally, if you found this book useful in any way, a review on Amazon is always appreciated!

ARDUINO

***Tips and Tricks to Learn Arduino
Quickly and Efficiently***

DANIEL JONES

Introduction

I would like to express my gratitude in downloading this eBook, “*Arduino: Tips and Tricks to Learn Arduino Quickly and efficiently*” and a pat on the back for selecting this eBook.

Coding is something that has been in the world for a long time, from sending secret messages during the war to current coding that involves programming. This book is a guide on some of the technology that is used in coding, which is the Arduino boards.

These boards, as explained in the book in detail was for beginners. Even if you are an advanced user in programming this book can help you determine which boards to move on to next, after using Arduino for quite a while.

This eBook contains steps that have been tested and proven on how

- Move to the latest 3.3V from the previous 5V!
- Combine two coding concepts to create a logic module of your own
- Make the best out of your Arduino
- On android phones (if you have been looking for ways of doing so, look no further!) how to unlock your bootloader
- In your device to install a bootloader
- Install libraries on your devices, be it Mac or Windows or Linux

This eBook contains pictures that will guide you on the steps you will take to install your ArduinoISP, install libraries in your devices, or how to upgrade your Arduino device to Arduino328 chip.

With the great array of books out there, I am beyond grateful that you decided to select this book, once again a huge thumbs up to you for selecting this e Book, I believe it will be of great and help and enjoy it!

Chapter 1

Arduino UNO

Quite frankly, I believe most people do not know what Arduino is, what it does or why it is important to some people. I'm pretty sure there are times that you hear some people mention the word, and you come up blank on what it even means, then get a headache just thinking about it.

In this book, I will explain to you what Arduino is, the various ways in which you can make use of it on your devices and the best upgrades that can help you make the most of your Arduino.

To begin with, I will explain what Arduino is and the types of Arduino that are available in the market. Being that hardware manufacturers constantly upgrade their merchandise, there are some latest versions of Arduino that I am going to clarify for you.

Below is what an Arduino looks like, just to keep you in the loop.



But before all that, HISTORY TIME!

Once upon a time, there was the first series of Arduino that had RS232 that was used by Arduino associated friends and the team, no one else. Then, the first manufactured Arduino that got famous that was named NG (New Generation, you know like Star Trek-if you are a fan you will know about this). The NG was using the Atmega8 chip, which back then was running at 16MHz and had an FT232 chip in place of the USB interface.

Years later, a new version was made going by the name Diecimila that had an updated chip from the previous predecessor Atmega8 to Atmega168. This was phenomenal as it doubled the memory and space from 8K to 16k, though

it still ran at 16MHz. there were two extra headers that were added to this Arduino for the 3.3V, this was changed from what was previously there the FTDI chip a reset pin that when a shield was used to cover the Reset button, was quite useful.

The space that the bootloader (explained further in the book what it is and does) took up to 2KB and was running at 19200 baud. An added asset was auto-resetting to make life much easier and awesome for its users.

The Duemilanove was released in 2009 with an upgraded chip of Atmega328! With doubled memory and space. There was another upgrade of the power is automatically switched between DC-jack and USB which removed the previous jumper that most people I'm sure complained about.

Moving from the programming to standalone became an easier and faster process and it got rid of the unnecessary confusion. The baud increased from 19200 to 57600 baud but the bootloader still took the 2KB space.

Uno! uses the power switch and the 328P chip, was released in 2010. There was more space created for users projects when the bootloader was decreased in size and it was later called OptiBoot and runs at 115K there is an extra flash space of 1.5K this was used by the bootloader.

The FTDI chip was replaced by the atmega8u2 that was there previously and it is the best thing as it allows you, as the user, to have different interfaces for the USB. An extra 3.3V regulator is available (in more complex terminology LP2985) a 3.3V supply is way better, thankfully!

Exciting things in Store

In all the other Arduinos, meaning the older versions (Diecimila, Duemilanove, and NG) used the FTDI chip, which is the FT232RL for converting the serial TTL that is on the Arduino chip. This is quite an achievement as it allows you have printable debugging. It connects to software like Python, PureData/Max etc. etc. you are also able to update the firmware through the use of the serial bootloader.

With free royalty drivers, the FT232RL works quite well which is a good thing. The only way for it to work is if it is a Serial/USB port that is the thumbs down the side of the FT232RL chip. The chip cannot act like a disk drive or mouse.

The pic below shows you where the Atmega8u2 is located:



The FTR232RL chip and atmega8u2 chip were exchanged when the new Uno was released. The new chip allows you to perform a couple of tasks; though as a heads up, it pretty much acts the same way as the FTDI chip worked; it simply is still a USB-serial port!

An improvement that has come with the chip, Mac users previously had to install FTDI drivers but the 8u2 imitates an accepted CDC serial device. Consequently, Mac customers now do no longer need to install a motive force in any respect.

Windows users do need to put in the .INF record, however, no drivers. Which means there are few problems with the Windows new versions. An INF file is required when using a serial USB device if you are using windows, unfortunately ☹

The 8u2 to advanced users, if you are reading this book, how you doing? You can turn your Arduino into any sort of USB device that suits you, literally. An example, you can make your USB port act as a mouse or keyboard, or a MIDI interface and so on. There are ways to do this, get cracking!

There are other extras that came up when the 8u2 reduced the price for the board....your pocket will be thankful.

More 3.3V Power

The older boards sadly had on the FTDI's chip, within its internal generator, was the 3.3V power supply. At most you had 50mA power supply, at best. SD cards have high power. This is similar to ADC that would have made the FTDI chip slow and it would also affect the USB as it would reset its

connections.

When you have the UNO, you are not going to have such a problem because of the added 3.3V regulator, a 150mA is provided quite easily by the LP2985 and it gets ranked as a high-quality regulator. It has a 1% analog reference and it is fantastic to use when powering things up.

New 3.3V regulator
(LP2985) @ 150mA



With all that you have read, I am certain you have several questions. For instance, the Arduino chip can run at 20MHz, yet why does it run at 16MHz? Here's why....running at 16MHz was the original Arduino which worked with Atmega8 when upgraded the boards were to be made speed compatible. Arduino, if you are looking for a board that has fast processing, it is not for you since it is only an 8-bit, therefore the chips will run at 16MHz.

Uno is available under Creative Commons license; the Arduino is still an Open source software and hardware.

UNO R2 & R3

UNO R3 was released by Arduino as their new version in 2011, it was better known as revision 3 will be available. Here is what is known about that version of Arduino:

- a) It was not available to people until the 1st of December that same year.
- b) It has similar uploading and driver and looks as UNO.

The few changes that were made in UNO were:

1. The reset button moved close to the USB connector making it simpler to press in case the shield is on the pinnacle.

2. USB controller chip moved from atmega8u2 this is 8K to and atmega16u2 that is 16K flash. It now does not necessarily imply that there is extra flash for our sketches. The improv is, especially in your USB interface chip. It means that a low-level USB interface will be easily available like MIDI/Keyboard available. These are only theoretical at the moment and might change in the future.

3. Next, to the AREF pin, the PCB has three more breakout pins on it. Two 12C pins (SDA). This is a duplicate of the analog pins 4 and 5 have no extra 12C interface. Next, to the Reset pin, there is no IOREF that informs the shields what I/O voltage pin that the board runs by, an example is UNO which has 5 voltage power. It is a copy of the energy pin and to the UNO's voltage level shifting it adds absolutely nothing!

With the mentioned changes, some things did not change in UNO:

1. Board size and shape is the same
2. Upload speed and techniques
3. Processor speed and size are the same that is the ATmega328P that runs at 16MHz. There is not much of a difference on how the code will run on the R3, but definitely not faster.
4. The driver is the same.
5. The number of pins is similar even with more breakouts!
6. Shield compatibility meaning the lugs working in UNO R1/R2 work in the same manner in R3.

Arduino's Progress

The Creation of Arduino boards

In Ivrea, Interaction Design Institute was where the first boards were made. The boards were made specifically for students who did not have an inkling on any programming and electronics concepts. The boards adapted to new challenges, changing to Internet of Things (IoT) applications from 8-bit boards and other things.

In the years that have passed, Arduino boards have built projects. Designers, programmers, experts have all gotten together and, with their donations, they have been able to add immense knowledge that can aid beginners and specialists learn more about programming and coding as well.

Advantages to having an Arduino board

- Makes working with microcontrollers simple
- It gives some advantages to teachers and beginners over other systems
- It is a cross-platform
- It has an open and extensible hardware
- It is inexpensive
- The programming environment is clear and simple

Different Arduino boards

As mentioned above we have the NG, Diecimila, Duemilanove, UNO R1, R2, and R3, but there are other Arduino boards.

Such as:

Red board- like the name suggests this Arduino board is red in color! It is the best thing for those who want some color in their devices other than blue or black. This works on Windows 8 only which means you won't need to alter your security settings. Using a Mini-B USB cable is the only way it can be programmed on the Arduino IDE. The picture below shows what it looks like.



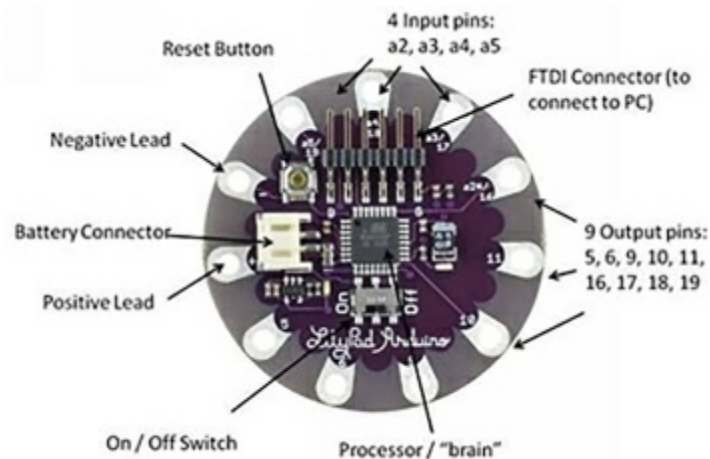
The differences between RedBoard and Uno are as follows:

- The color, first of all, it is SparkFun red
- The price of the RedBoard is significantly lower than Uno
- The RedBoard is offered only in SMD version and the SMD is taken further as it makes every component surface mount with no sharp edges at the bottom of the board. While the Uno board is PTH.
- The RedBoard uses a smaller mini-B connector meaning you will

require a mini-B-to-A USB cable connect to your device

- Arduino Uno uses ATmega16u4 that is loaded with custom firmware so as to convert between serial and USB. The RedBoard uses FTDI FT232RL. When installing drivers, that is where the difference is as they both require different driver file.

Lilypad Arduino Board – is a wearable e-textile era. The boards on this Arduino board were designed with connecting pads and have easy backs in order that they may be sewn into apparel by use of conductive thread. It accommodates of I/O energy and the sensor boards that have been constructed for e-textiles. And the most amazing bit, they are washable!



An Arduino Mega R3 Board- as we've discussed above, they are just like UNO's huge brother. it's far very useful for designing tasks that require several virtual i/PlayStation or o/PlayStation like many buttons.



Arduino Leonardo Board – changed into the first improvement board of an Arduino. It has one microcontroller in conjunction with the USB which means its pocket pleasant (yay!) and simple. With the USB handles without

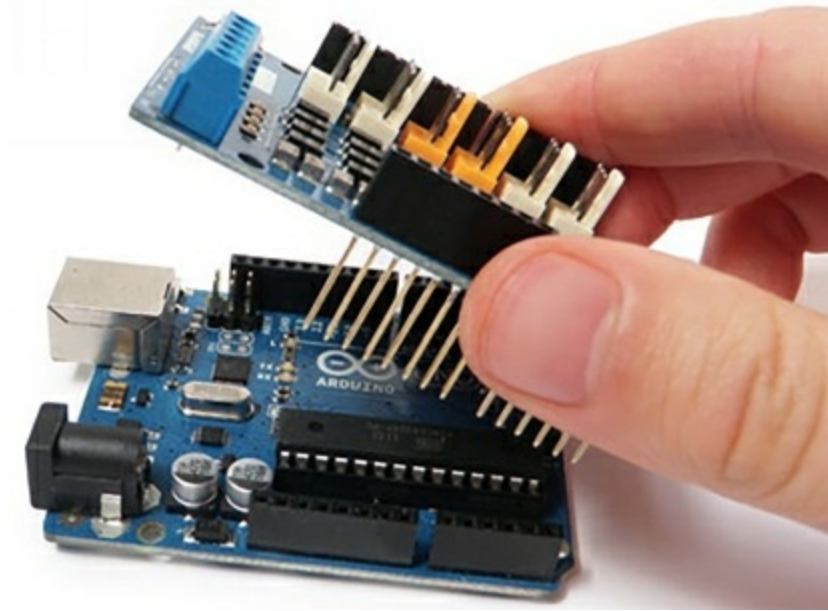
delay via the board, application libraries may be obtained and it permits the board to follow a keyboard or mouse of the laptop.



Arduino Uno (R3) – as mentioned formerly is a large choice on your preliminary Arduino. With 6-pins used as PWM (pulse width modulation) outputs, reset button, strength jack, and USB connection. It has the whole lot that may be used to keep up the microcontroller.



Inside the ebook, there can be mentions of Arduino shields that are pre constructed circuit boards that connect to several Arduino forums. Additional capabilities are added to the Arduino boards by these shields such as motor controlling and LCD screen controlling among other capabilities.



Arduino shields are as follows:

- GSM shield
- Ethernet shield
- Proto shields and
- Wireless shields

Chapter 2

How to Install Libraries

After discussing about Arduino and their background in this chapter we are going to look at libraries, not the one in your house or that huge building by the street that houses enormous amount of books. This library is one that is found in your device and helps the Arduino run better than it was previously.

Within the chapter we will talk the way to install libraries in your Mac, windows and Linux devices, followed with images to manual you and make sure you're doing the installation system successfully.

What's a library?

After discussing Arduino and their background in this chapter we are going to look at libraries, not the one in your house or that huge building by the street that houses an enormous amount of books. This library is one that is found in your device and helps the Arduino run better than it was previously.

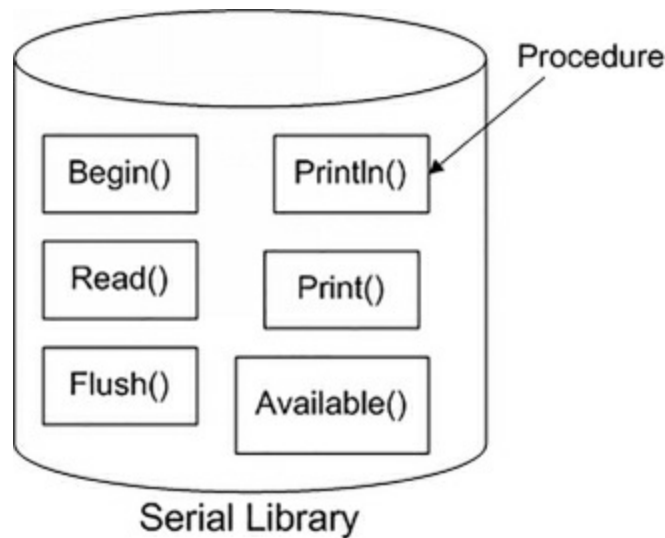
Within the chapter, we will talk the way to install libraries on your Mac, Windows, and Linux devices, followed by images to manual you and make sure you're doing the installation system successfully.

What's a library?

A library is a vast place that hosts a large volume of books all the knowledge that you need to do something, can be found in a library. For example, fix a car, change locks, understand a language, disappear into fairy tales; a library has it all.

It is possible for you to purchase the books and always have it, but the library houses the book for you, less clutter in your house or room if you are the kind that does not like clutter or fancies having books in your house.

Software Libraries work the same way as the physical library. The list of things that need to be done is housed in a software library is the definition of a procedure. This library is made up of a large number of procedures that are all related one way or another. How this works for example; in case you want to exchange automobile oil, you need to discover the car Oil Changer: that's a collection of processes that are already written so that you can use without having a training session on how cars work.



This diagram shows what an Arduino Serial Library. The library permits Arduino to send data to your computer.

Arduino libraries are convenient for sharing code like commonly used utility functions. There are two types of Arduino Libraries:

a) Libraries that are user installed

On the sketchbook folder, this is the place to have them on your device. this is to permit them for use with all the versions of the IDE. It will save you the trouble of reinstalling all your favorite libraries in case a new IDE version is released.

There are plenty of libraries that have device drivers and useful functionality that cater to quite a wide array of hardware. The devices can be found in GitHub, Google Code, and Arduino Playground.

There are over 100 libraries that are provided by Adafruit that do support almost all the Arduino compatible products. These libraries are usually hosted on GitHub.

b) Standard Libraries

These libraries are found in Arduino IDE and are used for commonly used functionality. The IDE and other examples found in it are supported by these libraries.

Fundamental conversation features and help for a number of the most not unusual hardware sorts are covered in the fashionable library capability which includes character liquid crystal display shows and Servo motors.

Those libraries are pre-set up within the Libraries folder of the Arduino

you're installing. in case you do have more than one versions of the IDE already set up on your tool, all the one's variations do have their own set of libraries.

One key is to be aware and think critically; do not install your libraries inside the identical folder neither is it a very good concept to trade the usual Libraries.

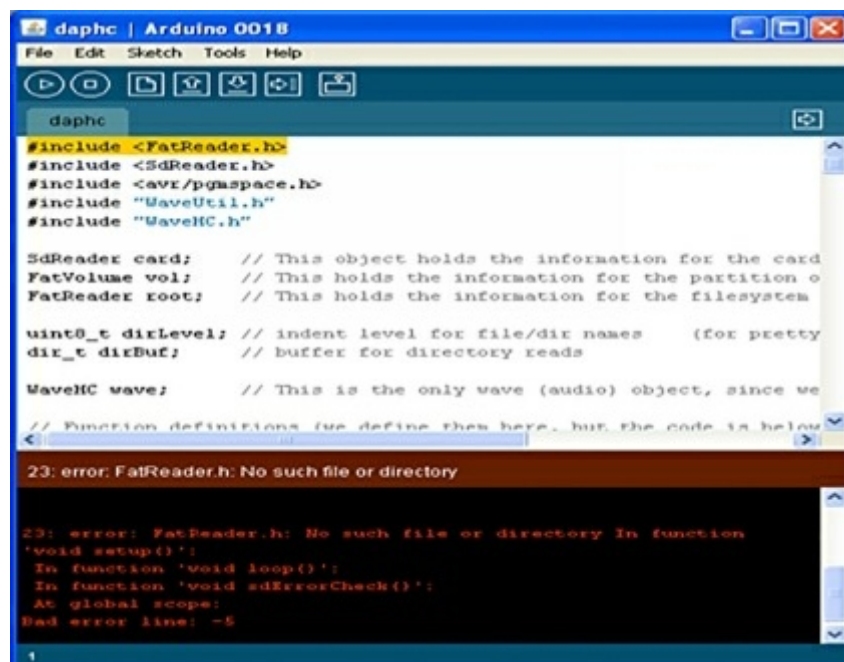
Using Libraries

It's far possible to add on pre-crafted libraries that your device's software and is one of the pleasant capabilities that the Arduino task has. You have the choice of picking among the array available which one to install.

Once the sketches that you are currently working on are in need of libraries, they will be loaded only then. The sketches do depend on the libraries that are available. You check at the top of the sketch in order to know what libraries you are using at the moment. They tend to look like this:

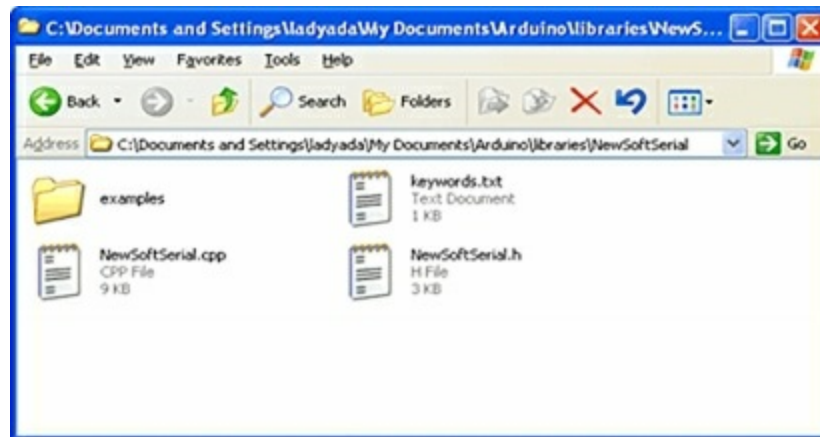
```
#include <FatReader.h> Copy Code
```

The library that you will require is named FatReader or gets one that has a FatReader file. When the library is not installed an error box will appear:



What is found in a library?

The files in a library end with `.cpp` (C++ code file) and `.h` (C++ header file). Some files that are found in a folder is what a library is.



There are some files that conclude with `.o` are C++ compiled Objects. Make sure that you delete the `.o` files should you be working on the library and also make sure to change it as this will make the IDE restructure the changed `.cpp`'s into the fresh `.o`'s.

Two optional files may appear when in your library; **examples** folder that might have some handy test-sketches and **keywords.txt** this gives cues to the IDE the color your sketches should be.

Under File – Examples – Libraries dropdown, all these will show up, like below:



NOTE:

It is of utmost importance that the library folder's structure is on point. The `.c` and `.h` files are at the bottom, showing that they are at the last level. For

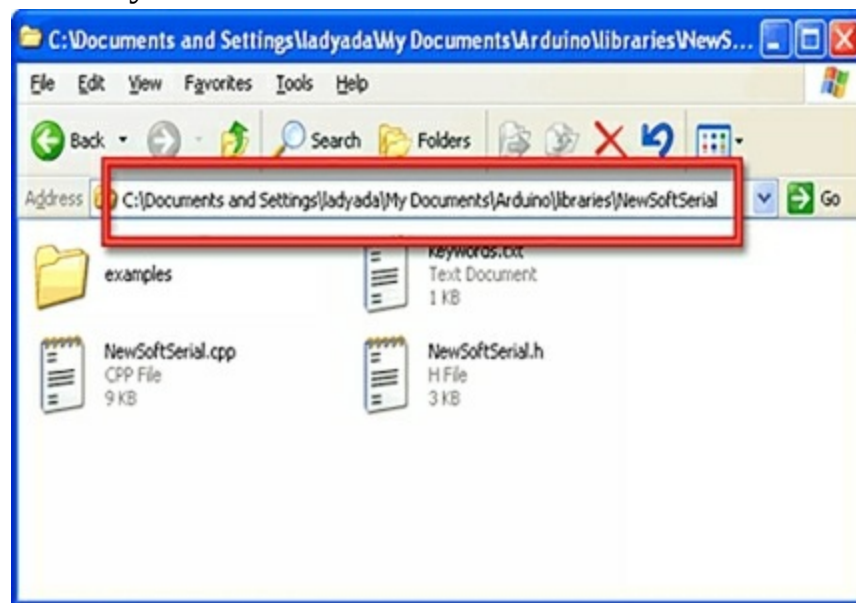
instance, you won't have **Arduino/libraries/MyLibraries/WaveHC/file.c** rather it is supposed to appear as **Arduino/libraries/WaveHC/file.c**.

Setting up Libraries

The sooner versions of Arduino like Arduino V16, their libraries had been saved inside the `ArduinoInstallDirectory/hardware/libraries` folder that housed each and every built-in libraries including Serial and cord.

Those are saved in the user libraries as `ArduinoSketchDirectory/libraries` folder in `Arduinov17` and others above it. The first time, you likely ought to make a libraries folder and you may not be required to move your libraries or re-deploy them each and every occasion you upgrade the software.

Below, I will show you how the NewSoftSerial looks in a Windows device:



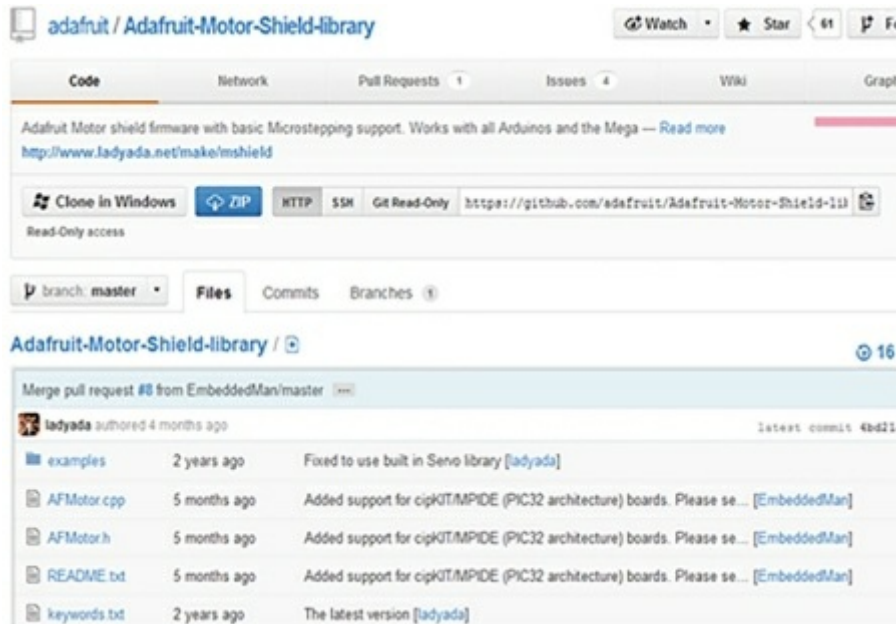
Inside, create a *New Folder* named libraries also the uncompressed library folder as well; place it in this new folder you have created.

When you have done so, make sure the file reads as so `Documents/Arduino/libraries/MyNewLibrary` folder and it has the `.cpp` and `.h` files in it. Once you have completed this, restart the Arduino IDE.

How to Install a Library on Windows

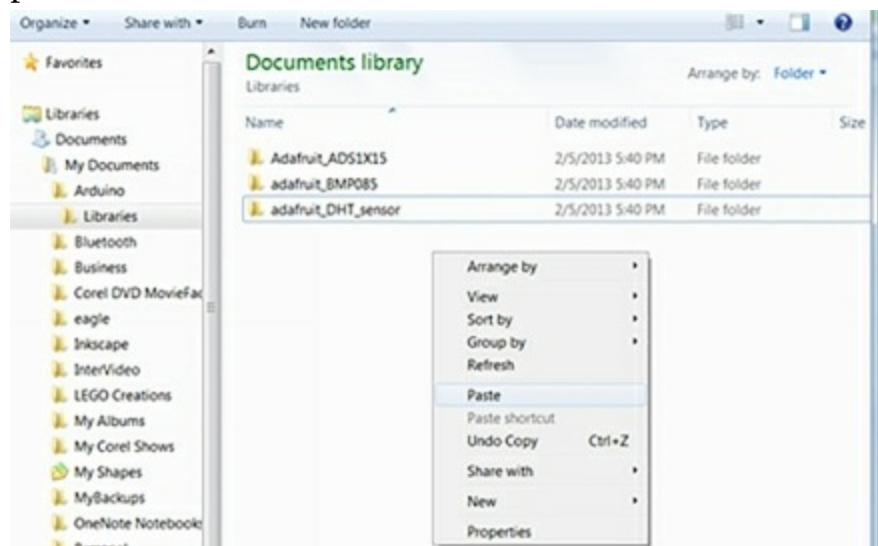
1. First and foremost close the Arduino IDE. The IDE is used to scan libraries at startup and it will be impossible for it to see the new library you are to install or have installed as long as the IDE is open.

2. Download the Zip File by clicking on the *Zip* button.



3. copy the library grasp folder into the Zip record.

4. As soon as you have copied the library master folder, paste your sketchbook libraries folder in the master folder you had duplicated from the .Zip.



5. Since the IDE won't realize the folders that have been made with dashes in the name. Thus, change the Master folder by giving it a legal name. You can use underscores if you want.

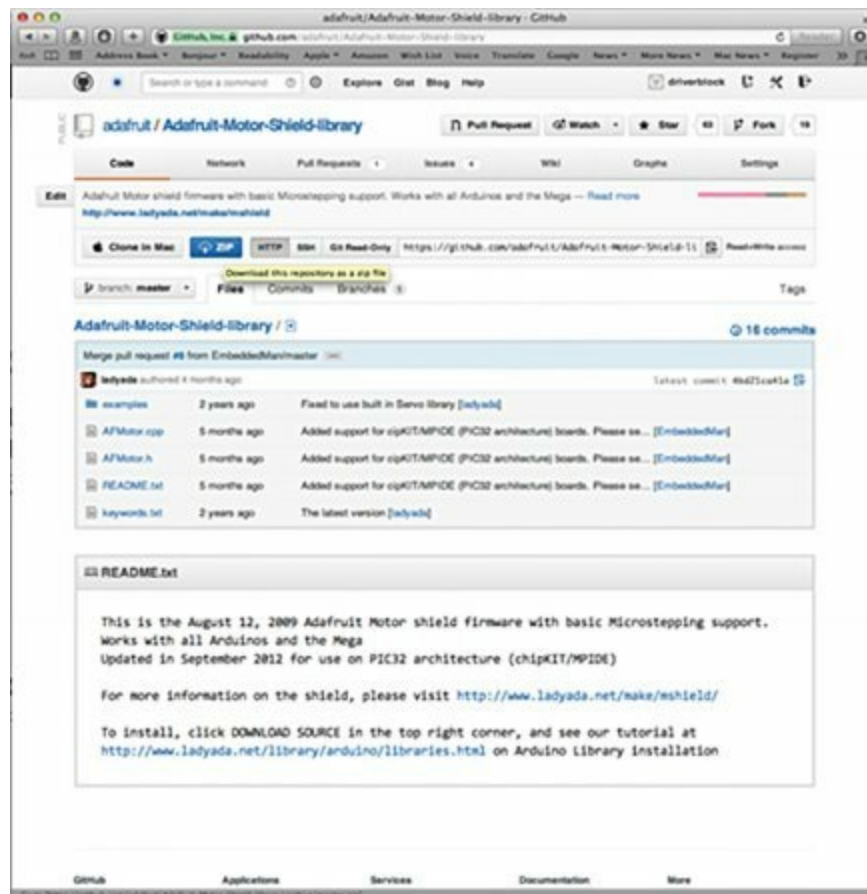
6. After you have renamed the file, restart the Arduino IDE and make sure you confirm it is in the File-Examples menu. If you are not sure

how, test using a library example.

7. Whilst performed, click on the checkmark icon on the top left and confirm that there are no mistakes in the example cartoon compiles.

How to install a Library on Mac OSX

1. Make sure the Arduino IDE is closed.
2. Download the Zip File from the GitHub.



3. In the Downloads Folder, the Zip file will open.



4. You drag the master folder from the Downloads to the sketchbook Libraries folder,

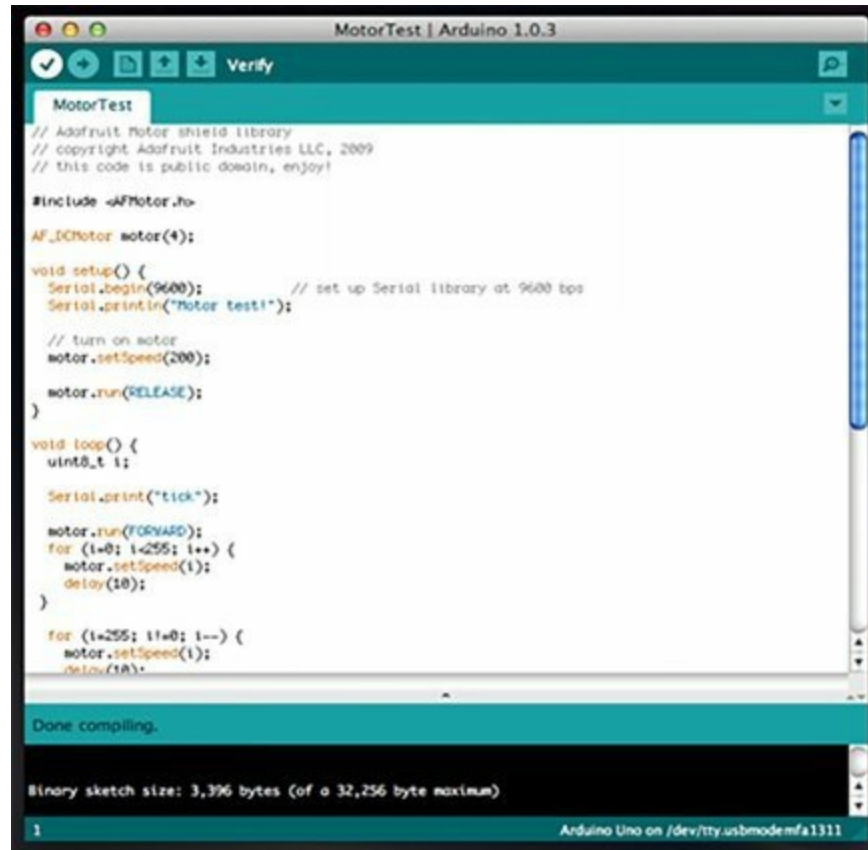


5. Give the file a legitimate name for the IDE does not acknowledge dashes in a file name. Underscores can be incorporated in the name.

6. Restart the Arduino IDE once you have completed it and confirm the library is at the File-Examples menus. Test using the library examples.

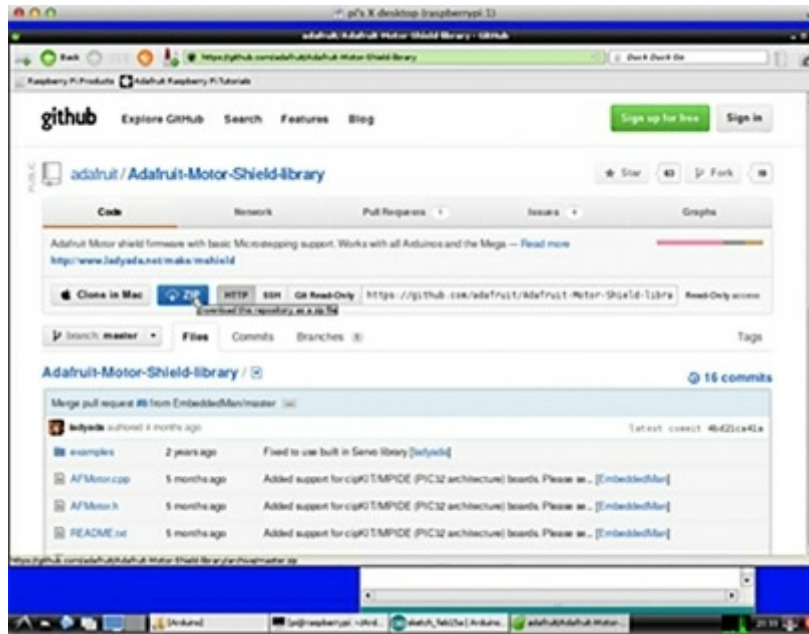


7. Finally, verify that the library complies by clicking on the top left of the checkmark icon. No errors are to be on the example sketch, confirm this.

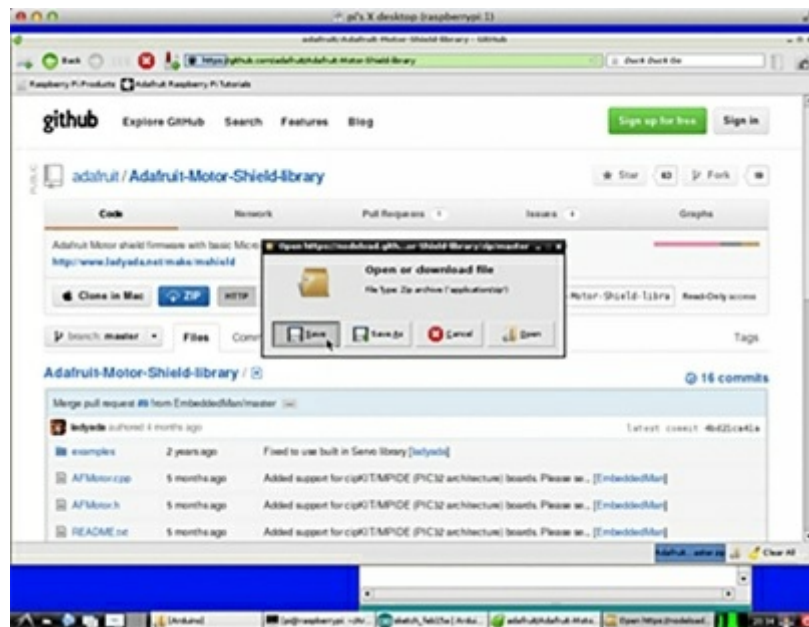


How to Install a Library on Linux

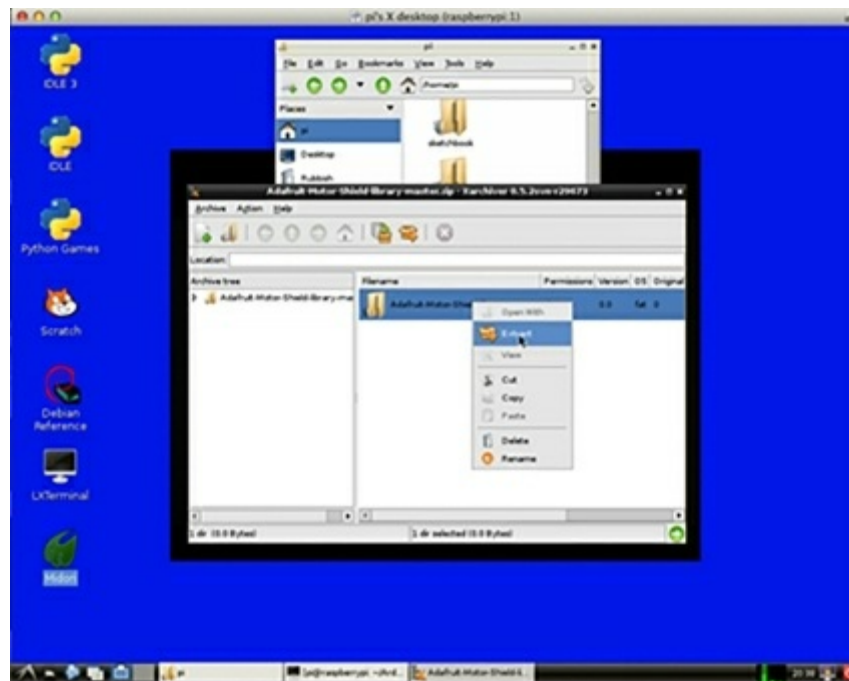
1. Like the previous devices, make sure that your Arduino IDE is closed.
2. After closing the IDE, click the *Zip* button to download the file from GitHub's repository page.



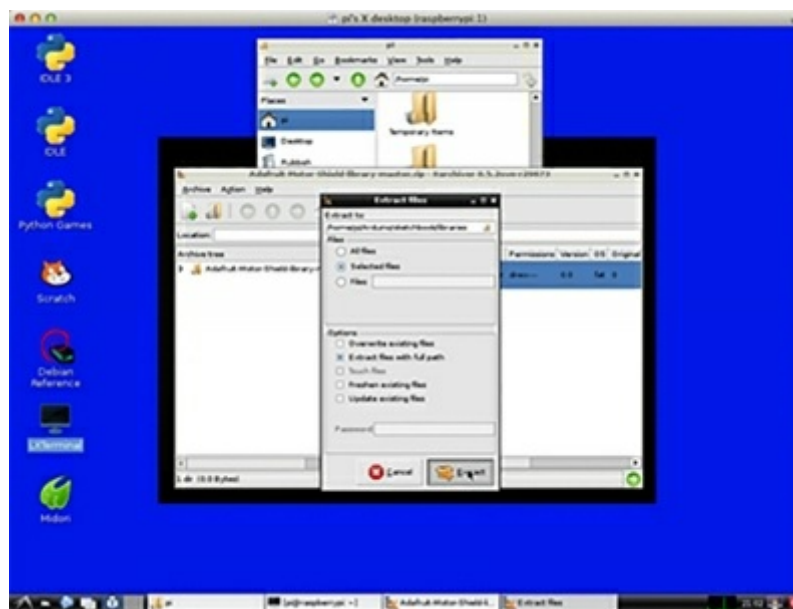
3. Save zip file at a location that is convenient to you.



4. Copy the master library folder wherever you want after you open the zip file.
5. On the menu, select *Extract* and navigate to your Libraries/Sketchbook folder.

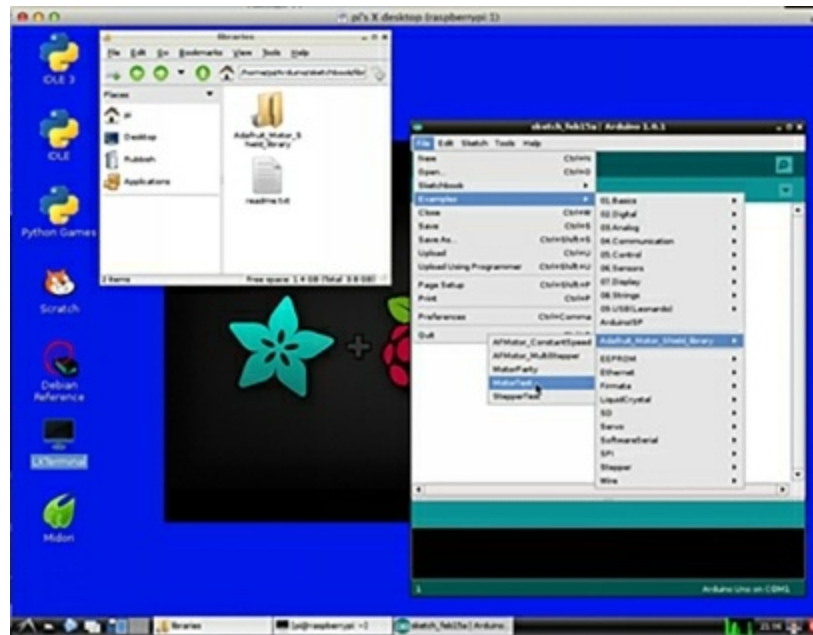


6. After moving from the libraries/sketchbook folder, complete the extraction by putting them in this folder.



7. Give the new folder a name and you can use underscores as well.
8. After naming the file, restart IDE to confirm the libraries are the same as

those on File-Examples menu. Use library examples as tests for you.



9. Select the check-mark icon from the menu to verify that the compiled sketch has no errors.

Common problems with Libraries

There are many common problems that are related to installing libraries to your device.

They include:

- No name type. It is a relatively known mistake library associated errors messaged this is recognized. That is to suggest that the compiler is not capable of discovering the library. There are several reasons that can lead to this:
 - o There is a wrong folder name
 - o The library dependencies are an issue
 - o You forgot to shut-down the IDE
 - o The library you are looking for is not installed
 - o It is a wrong folder location
 - o It is the wrong library name

The reasons above have their solutions which are explained below:

1. There is no 'sketchbook' folder

The device you are using, be it Windows or Mac, you might be shocked to discover that the folder might be under another name and not Sketchbook. You will locate your library's location once you find where you installed it first. If you still can't install the sketchbook properly, look at the previous topic titled "How to install your libraries" to help you.

2. It is the wrong folder name

A file with certain characters in them will not load with the IDE. The dashes that are generated by GitHub do not go well with IDE; that is why you have to rename the folder once you unzip the file. It is a means to get rid of "unwanted" characters. You can replace the (-) with an underscore (_).

3. It is the wrong folder location

The libraries and standard libraries installed are the only ones that can be found by the IDE when in the sketchbook folder. any other folders that hold libraries, the IDE received to be able to locate them.

On the pinnacle degree of the Libraries folder, the Library folder must be at the pinnacle and now not in a subfolder because the IDE gained to discover it.

Take be aware that a few third-celebration library repositories have one-of-a-kind folder structures and it might be as much as you to rearrange matters for your device to permit the libraries documents to be at the pinnacle level of the folder.

d. Several versions

All a couple of variations of a library that you have might be loaded by way of the IDE and compiler errors will occur. you could rename the library folder, however, you have to additionally flow it outdoor the sketchbook Libraries folder to keep away from it from being loaded by the IDE.

4. You forgot to shut-down the IDE

Libraries are searched by the IDE at startup only. At all instances ensure

which you close down the IDE and restart before because it will apprehend a brand new set up library.

f. The library dependencies are trouble

There are a few libraries which can be depending on different libraries. just like the Adafruit picture show libraries which are depending on Adafruit GFX Library.

g. The library you are looking for is not installed

This can be because the library is incomplete. You consequently should download and set up the entire library a brand new, but don't trade the names of any of the documents that are discovered in the library folder.

h. The incorrect library name

The call in the library should match, capitalization, underscore and everything this is protected within the mane with that of #encompass of your sketch. If not, the IDE will no longer be capable of discovering the record you are looking for. Example sketches that are part of the library have the spelling necessary. From the example sketches, copy-paste not to have any typo errors.

i. The "core" libraries

Some libraries like GFX library are not to be directly used. The core graphics functionality provided are for several Adafruit displays yet without a specific driver library they cannot be used for that particular display.

The next chapter we are going to look at some the bootloader.

Chapter 3

Tricks for the Bootloader

The name has been mentioned in Chapter 1 when discussing Arduino. In this particular chapter, we go into detail about what exactly is the bootloader.

What is a Bootloader?

Think of how before you do anything, you have to make sure all that you require are available. For instance, if you want to make a smoothie, you need to make sure there is electricity, the blender works properly and the fruits that you need are there before you blend everything to make your smoothies. The preparation is what the bootloader is.

Before any operating system runs, a bootloader, which is a piece of code, runs first. There are various bootloaders per operating system, all depending on the type of functions it wants to operate or perform.

The bootloader can also contain some commands for modifying and/or debugging the kernel environment and it can contain several ways in which you can boot the OS kernel.

Being that it is the first software, after reboot or power-up, the bootloader has a high processing power and it is board specific.

If you are an Arduino user and do not want to tweak your bootloader, do not try this/ it is advisable not to make any changes to your bootloader, but being a wild card that some of you are, try the tricks below; at your own risk of course.

These are tricks for a fixed up ATmega328 bootloader.

To make the most of your bootloader, there are a couple of changes you will have to incorporate. In order for you to program it, changing the Makefile's ISP TOOL will be a requirement.

To perform this command, there are other commands you can use **make adaboot328; make TARGET=adaboot328 isp328.**

There are several fixes that are available with this version.

1. It repairs the “missing signature bytes” anomaly which makes the avrdude not act accordingly especially if it programs without IDE.

2. 'no-wait' and 'no-hang' fixes are incorporated
3. The EEPROM code is fixed. This allows you to upload and download flash as well as EEPROM memory.
4. Using LED, there is an 'upload feedback'; this is for Arduino clones which do not have TX/RX LEDs.

Because of this particular chip's extended memory, the fuses will vary greatly.

Below are the processes to perform for the fixes above:

1. **"No-Wait" Bootloader**

After uploading the sketch, this hack will automatically start the sketch. The bootloader will begin only when the reset button is pressed. Power will most likely go directly to the sketch once plugged in.

Make sure you follow how these lines are to the t: You can find the code online, as they are free to get.

In Diecimila Arduino or/and New Generation Arduino will the process above work.

2. **No-Hang Bootloader**

There is always the possibility of tripping your bootloader by accident, especially if it is by your communications program. It is an often seen occurrence to those Arduino users who use Diecimila that has an auto-reset.

The quick hack below enables the bootloader to quit if a '0' character is received first. This is an indication of communication between the bootloader and the Arduino software.

Do according to the instructions below:

✓ *Copy code*

1. `uint8_t firstchar = 0;`

Then paste:

Copy code

1. `/* main program starts here */`

```
2. int main(void)
3. {
4.     uint8_t ch,ch2;
5.     uint16_t w;
6.     uint8_t firstchar = 0;
```

✓ Copy:

Copy code

```
1. firstchar = 1; // we got an appropriate bootloader instruction
```

Paste:

Copy code

```
1. /* Hello is anyone home ? */
2.     if(ch=='0') {
3.         firstchar = 1; // we got an appropriate bootloader instruction
4.         nothing_response();
```

From the code above, paste this below it:

Copy code

```
1. } else if (firstchar == 0) {
2.     // the first character we got is not '0', lets bail!
3.     // autoreset via watchdog (sneaky!)
4.     WDTCSR = _BV(WDE);
5.     while (1); // 16 ms
6. }
```

The last two lines can be replaced with **app_start()**

3. Using AVRDUDE to upload sketches

The bootloader in use is a 'stk500' compatible, this states that it is possible for you to use the reliable AVRDUDE so as to program your Arduino.

What you need to do is;

- a) Plug in the USB cable that you have.
- b) Push the reset button first without starting avrdude:-

- o You are required to make the most of `-b 19200` in order to set the baud rate to 19200
- o You will use the `-F` signature if the device signature read does not work as it is supposed to
- o `avrisp` is the programmer type
- o `-p m168` is the device type and
- o The FTDI chip that will be displayed determines the port to use

```

C:\WROXP\system32\cmd.exe
C:\>avrdude -p m168 -c avrisp -P com1 -b 19200 -F -U flash:w:Blink.hex
avrdude: AVR device initialized and ready to accept instructions
Reading : ##### : 100% 0.05s
avrdude: Device signature = 0x000000
avrdude: Yikes! Invalid device signature.
avrdude: Expected signature for ATMEGA168 is 1E 94 06
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "Blink.hex"
avrdude: input file Blink.hex auto detected as Intel Hex
avrdude: writing flash (1108 bytes):
Writing : ##### : 100% 0.78s
avrdude: 1108 bytes of flash written
avrdude: verifying flash memory against Blink.hex:
avrdude: load data flash data from input file Blink.hex:
avrdude: input file Blink.hex auto detected as Intel Hex
avrdude: input file Blink.hex contains 1108 bytes
avrdude: reading on-chip flash data:
Reading : ##### : 100% 0.84s
avrdude: verifying ...
avrdude: 1108 bytes of flash verified
avrdude: safemode: Fuses OK
avrdude done. Thank you.
C:\>

```

Other bootloader hacks on the Android device:

It is possible for you if you want to, to unlock your bootloader on your Android device. Unlocking the bootloader on your android device allows you to install custom operating system software. Some risks that you are likely to encounter once you unlock your phone such as installed applications and your phone not working properly.

There are two kinds of Android devices around the globe; those devices that are not unlockable and those that are unlockable. There are those devices though, that will not permit you to perform this operation; which are the not unlockable devices.

The manufacturer, model of your phone all determine whether your phone will allow you to unlock the bootloader.

Google phones, Nexus, by default are unlockable, so are many Sony, HTC, OnePlus phones which allow you to unlock your bootloader. The process is similar to that used in Nexus.

Those devices that do not permit you to unlock your bootloader mean that you will have to wait for the developer to unlock it for you.

Check the XDA Developers forum in your Device Section to know if your phone will allow you to unlock its bootloader.

For those whose devices are unlockable, do keep reading!

The first and the most important things that you should do is **backup all things that you want to keep**, be it apps, app data, personal files, and data as well. The process will erase all the data on your phone. You can use Titanium Backup Pro to help you backup your user data and app data.

The following steps will help you unlock your bootloader on your Android device:

1.

Download all essential files such as Java, Adb, and Fastboot for Windows phone and also Fastboot installer and Minimal Adb and install them.

This is to help the drivers communicate properly with the Fastboot and Adb commands that will be received by the device.

For Linux users (Ubuntu), you will have to install Fastboot and Adb tools. On terminal you will add the following command:

```
sudo add-apt-repository ppa:phablet-team/tools && sudo apt-get update
```

After the repository has been added, the command below needs to be added to enable you to install the Fastboot and Adb tools.

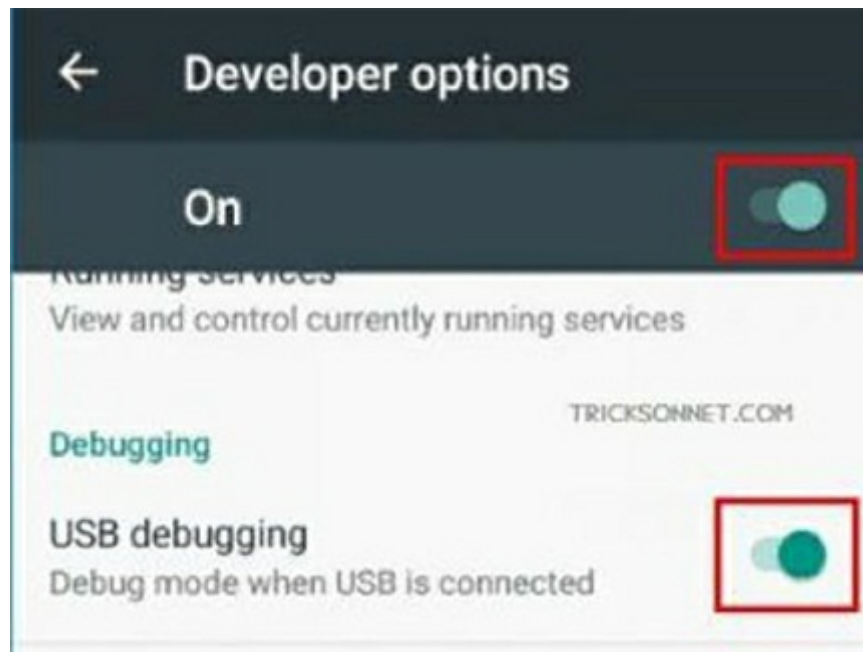
```
sudo apt-get install android-tools-adb android-tools-fastboot
```

2. USB debugging and Developer options on your device are to be enabled.

Developer options can be enabled by the following process:

- o Go to **About Phones** and click on the **Build Numbers** 5 times in a row.

Afterward, enable the USB debugging



There are those devices that will show the option above as Android debugging. You can enable *Allow bootloader Unlocking* or *Allow OEM Unlocking* if you have those options.

3. From the manufacturer, get an unlock key, though it is unnecessary if you have a Nexus device. This is necessary if you have a Sony, or HTC device.
4. Using USB cables, connect your phone and open the commands so as to promote as Admin and key in the following command;

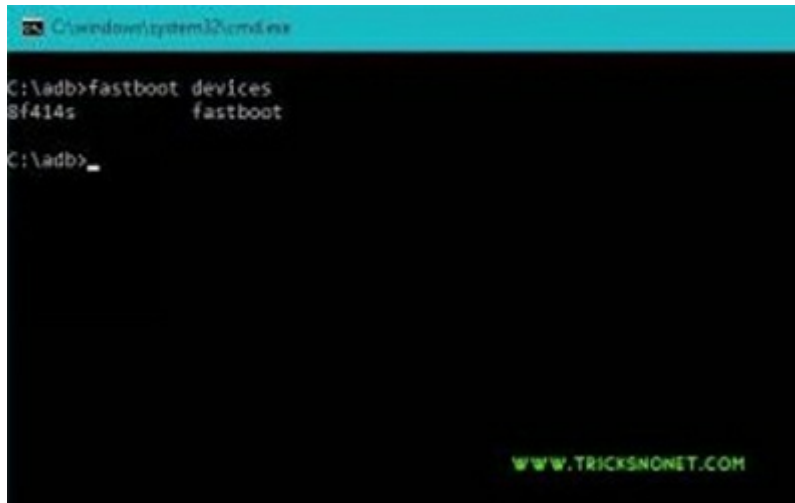
adb devices

Once done, a similar text will appear and you are required to allow ADB connections on your device too. After typing the command, key enter and it will take you to Fastboot Mode.

5. Enter the code below to confirm if you are connected to Fastboot mode

fastboot devices

The output should appear as so:



```
C:\windows\system32\cmd.exe
C:\adb>fastboot devices
8f414s      fastboot
C:\adb>
```

WWW.TRICKSONET.COM

This step brings you closer to unlocking your bootloader on your device. Use this command:

`fastboot oem unlock`

The bootloader on your device is unlocked if the output is like this:



```
C:\windows\system32\cmd.exe
C:\adb>fastboot devices
8f4e6s      fastboot

C:\adb>fastboot oem unlock
...
OKAY [ 0.003s]
finished. total time: 0.004s
C:\adb>
```

WWW.TRICKSONET.COM

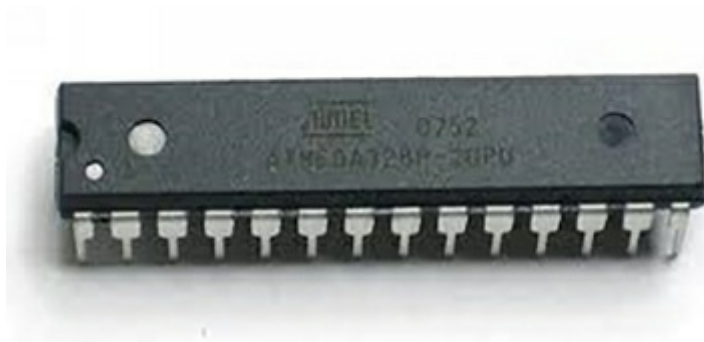
Make sure you have done the previous steps correctly. If any error messages appear, re-check the previous steps and do them again.

Chapter 4

Upgrading Arduino to the Latest Version of Atmega328P Chip

Introduction

The microcontroller ATmega is the brains of Arduino. This product is from the Norwegian chip company ATMEL. it is a low-energy CMOS 8-bit microcontroller this is simply based on the AVR improved RISC structure. It achieves a few exceptional throughputs which might be up to 1 MIPS in step with MHz



The Atmel AVR® does integrate steering sets from 32 well-known reason operating registers which are all associated with the ALU (mathematics good judgment Unit). This lets in the two unbiased registers to be reachable with a single training carried out in a single clock cycle.

This empowers device is designed to optimize the tool for energy consumption instead of processing tempo.

This chip has the subsequent functions:

- The temperature ranges from -40° C to 105° C.
- The advanced RISC architecture has
 - o 131 powerful instructions
 - o 32x8 general purpose working registers
 - o On-chip 2-cycle Multiplier
 - o Fully static operation
 - o It has the most single clock cycle execution

- o A throughput of 20 MHz with up to 20 MIPS
- I/O and Packages
 - o 28-pin PDIP
 - o 28-pad QFN/MLF and
 - o 32-lead TQFP.
- Atmel QTouch Library support with up to 64 sense channels, QTouch and QMatrix Acquisition.
 - It has an operating voltage of 1.8-5.5V.
 - Its peripheral features include:
 - o One ON-chip Analog Comparator
 - o Has separate Prescaler and Compare mode that go hand in hand with 2 8-bit counters
 - o An interrupt and wake-up on Pin Change.
 - o Six PWM Channels
 - o 10-bit ADC 8 channels in QFN/MLF and TQFP package that is used for temperature measurement
 - o Two Master/Slave SPI Serial Interface
 - o A real time counter with separate oscillator
 - o Watchdog timer that can be programmed with separate on-chip oscillator
 - o One 16-bit Counter with distinct Pre-scaler, Capture, and Compare mode
 - o One programmable serial USART and
 - o A one Byte-oriented 2-wire Serial Interface
 - Special microcontroller features:
 - o Interrupt sources both internal and external
 - o Power-save, standby, power-down, extended standby and ADC noise reduction sleep modes and also internal calibrated oscillator.
- Its power consumption is at 1MHz, 1.8V and 25°C with the power-down mode at 0.1μA, the power-save mode is 75μA and the active mode at 0.2mA.

- It has extremely high endurance non-volatile memory segments with:-
 - In-System Self-Programmable Flash program Memory with write/erase cycles.
 - For software security, it has a programming lock.
 - Internal SRMA has 2 Kbytes
 - EEPROM has 1 Kbytes
 - A long period of data retention
 - There are independent lock bits on two optional boot code sections
 - i. True read while writing operation and
 - ii. On-chip boot program for in-system programming
- There is a variation of speed grades like 0-10 MHz @2.7-5.5V and 0-20 MHz @4.5-5.5V.

The QTouch library offered by the Atmel chip is used for hiding capacitive touch buttons, wheels and/or the functionality of sliders that are found in the AVR. Robust sensing is offered by the patented charge-transfer signal acquisition. This also has AKSTM (Adjacent Key Suppression) technology that for key events it is for unambiguous detection.

The ATmega328P has a full suite of system and program development tools that support it. The tools include:

- Macro Assemblers
- In-Circuit Emulators
- C Compilers
- Evaluation kits
- Program Debugger/Simulators

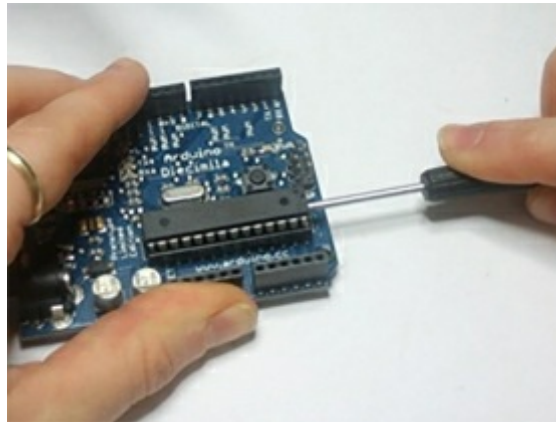
It is quite easy to replace and upgrade your Arduino and it will cost you a few coins. The upgrade will provide breathing room for you as your sketches work the same as they used to. To have the upgrade work for you, you will either program it yourself by using an AVR programmer, “bitbanging” or you will have to buy an already preprogrammed chip.

Below, I will show you how to upgrade your chip.

Chip Replacement

In order for you to upgrade your chip, you need to:

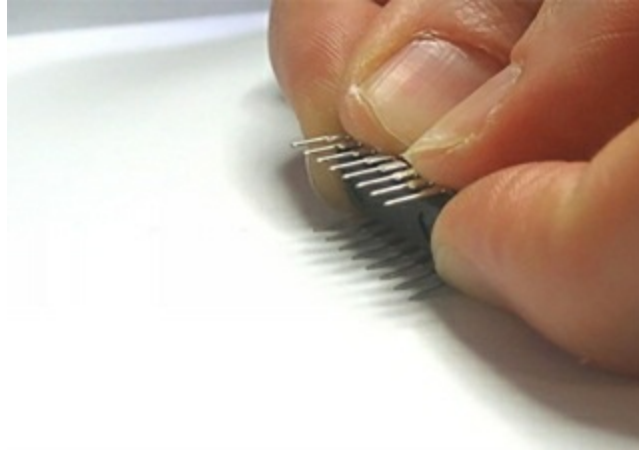
1. From its socket, remove the Arduino microcontroller by using a flat screwdriver or one that is alike. You have to be cautious not to bend the pins



2. Put the removed pin in a safe place. It is recommended that you place it in an anti-static bag.



3. Prepare the ATmega328P next. Whilst you look at the IC's pins, they may be a touch skewed when they are from the manufacturing facility; therefore, make sure you bend them in only a tiny bit so that they can be parallel. Hold both ends of the chip and make certain you operate on a table.



4. Update the old chip by aligning the pins and ensuring the notch of the chip matches the notch in the socket.

Downloading Arduino IDE with the ATmega328P compatibility

There might be some difficulties that you might encounter when upgrading your chip. In case you bought a chip from Adafruit earlier than that were given brought earlier than February 5th, 2009, the chip could have a baud rate set of 19200 that is much like the ones of older Arduinos.

The chips had been changed after February 5th, from 19200 baud change to 57600 baud rate that's near to 3 times the rate! This new baud rate was enforced so that it can work well with the current-at the time- Arduinos manufactured.

The chip's 19200 baud rate will be difficult to upgrade and upload it to your computer or device.

All you have to do is; exit the Arduino application and change in the hardware folder the file on your computer name boards.txt and change it from:

```
atmega328.upload.speed=57600
```

to;

```
atmega328.upload.speed=19200
```

In case you still have the same problem afterward, try BOTH to know which one works best for you.

Chapter 5

Conversion to 3.3V Devices

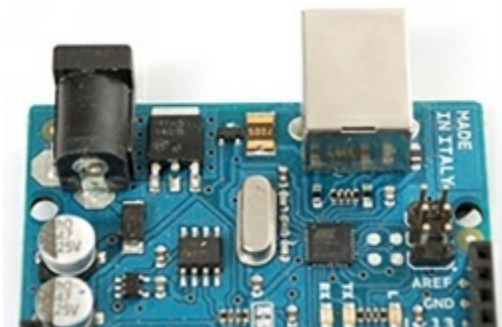
The standard voltage for all official Arduino devices for most microcontrollers and hobbyist electronics was at 5 volts. There are new displays, chips, and sensors that are 3.3V, though not 5V compatible.

Some devices that run on 3.3V power and logic are XBee radios, accelerometers and SD cards. The internals of the accessory that runs with a 5V can be damaged if you try and connect these devices to it.

Some chips, such as the CD4050 are used to perform a level conversion. In case most of your devices are 3.3V, this better for most if not all your Arduino.

The regulator of your Arduino is replaced to enable the DC barrel jack is used together with a regulator that is 3.3v and not a 5V. After this, the USB cable of the 5V is reconfigured to make sure that it can be the same 3.3v regulator too.

The diagram below shows this:



Replace the Regulator

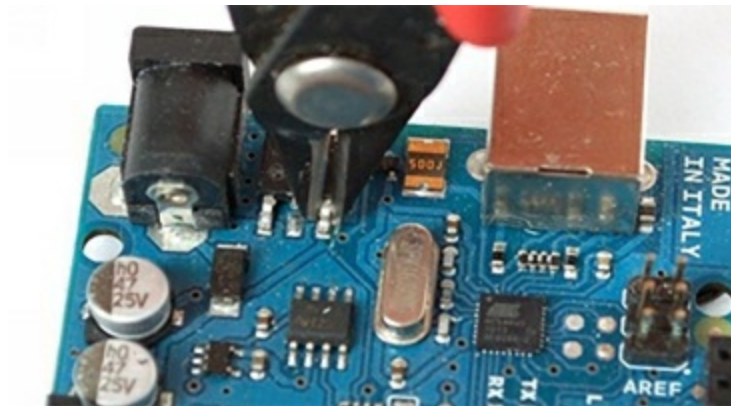
As we need to use the 3.3V instead of the 5V default generator that is currently in place; replacing the 5V with the 3.3V regulator is the initial step.

Below is 1117-3.3V regulator that is in a TO-252-3 package. Though if you do not find an 1117 regulator, that is because there are few manufacturers of the 1117 regulators whereas there are many factories that make the 7805 regulators.

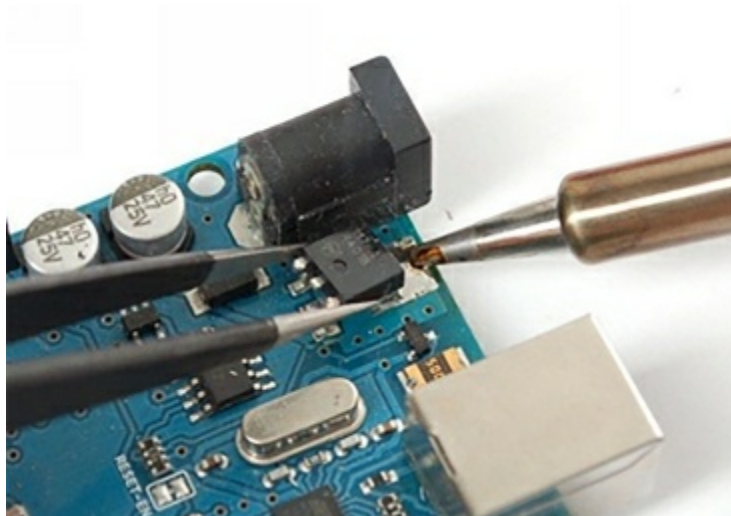


Such regulators are available in any electronic component shop.

The 5V regulator needs to be removed, and that is how we begin this process. To do this with ease, make sure you clip the two legs as shown below:

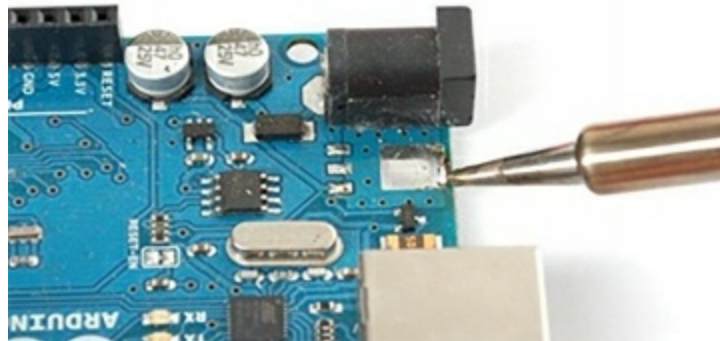


Heat up the tab next to get it hot enough so that you lift off the old part easily. Add solders to the tab, to counter heating the tab up to make it melt into your iron. The reason behind this melting of the solder is to improve the heat conduction on your tab since it is quite large.

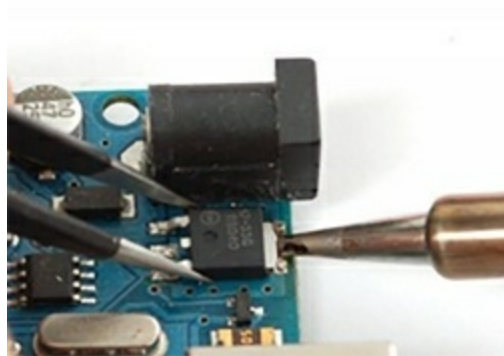


Do away with any clipped parts that are likely to have remained on the tab

right after you have cleaned up the tabs.



Next step is to make sure you align the new 3.3V regulator. The tab should be soldered initially. It is okay to use plenty of solder as it is necessary and make sure you are patient while doing this part. The tab does end up acting like a heat sink for your solder.



Eventually, work on the two legs of the regulator.



Fuse Replacement

What we are going to do next is quite tricky, if not done carefully and with you being cautious, you are likely to head out to the hardware store and get a new tab altogether. Therefore be extra cautious.

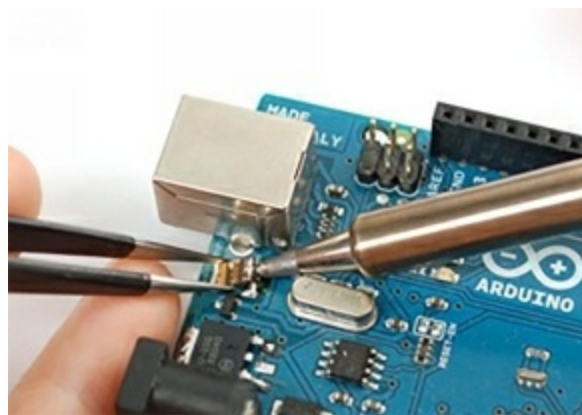
The USB jack that is on the tab provides 5V already. The output of the voltage generator and it are connected or tied. This is useful especially when the DC jack is not powered up.

To ensure the USB 5V goes smoothly through the regulator is by getting rid of the fuse. When you remove the fuse, you can then solder a diode that can be found from the regulator input. Such a diode is like the 1N4001 power diode that is perfect and quite cheap.

What can be visible as a downside is the dearth of a 500mA fuse specifically for the USB jack. The laptop could have a personal fuse, which is fantastic, which can be on the inside of the USB on the laptop. The likelihood of you destroying your PC is negligible, but a word of caution is that a little of your safety is being lost.

To make this happen;

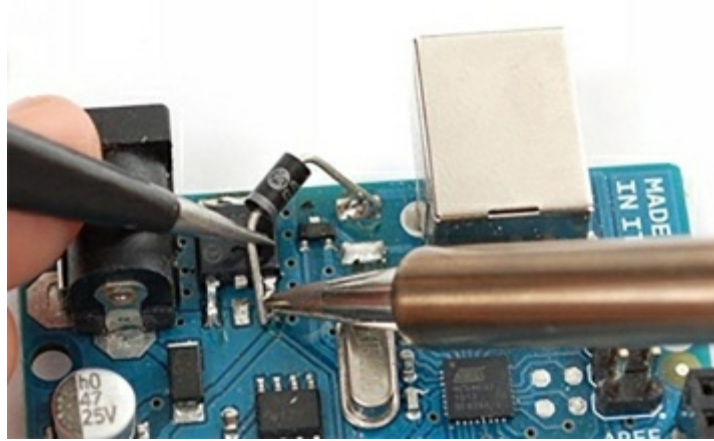
- a) With your soldering iron, heat the fuse. Adding solder to your tab may help out with heat conductivity.
1. This will help to melt both ends when you heat on end since the conductivity level of the fuse is high.



Bend the leads over after clipping the diode short. The anode side is to be soldered to the old fuse pad closest to the edge.

The cathode end, the one striped; solder it to the regulator's right-hand leg. This is illustrated in the diagram below.

When all this is done, the Arduino will instinctively pick whichever power plug that provides greater power to you.



Afterward, you are officially using the Arduino 3.3V! Congratulations on doing it on your own. AVR chips have about 3.6V that runs at 16MHz, which makes this chip of lower power/frequency specification. This is not an issue as the AVRs can at times be overclocked.

Enjoy your 3.3V powered Arduino.

Chapter 6

Tricks for Maximizing Arduino

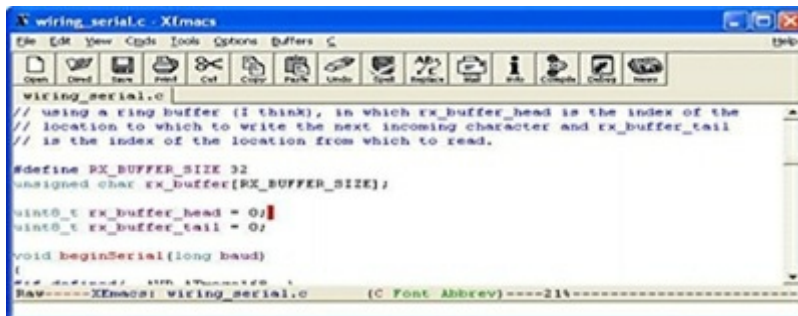
There are plenty of ways for you to maximize your Arduino use and make the most of it on your device.

1. Space in the RAM

You can be able to free up to 100 more bytes when you reduce the serial receiving buffer which are equivalent to 10% of the space in your RAM on the ATmega328P. This is also as large as freeing 128 bytes.

To be able to achieve this:

- ✓ On the **cores/arduino** directory, open it up.
- ✓ Change the name **wiring_serial.c** or **HardwareSerial.cpp** of this file
- ✓ On the top, there is **#define RX_BUFFER_SIZE 128** that shows 128 bytes are being used to buffer.
- ✓ To make changes, like from 128 to around 16, do as shown below;



Extra 2 bytes can be added or changed from what you already have by **rx_buffer_head** and **rx_buffer_tail** from **int** to **uint8_t**.

2. Bumpers

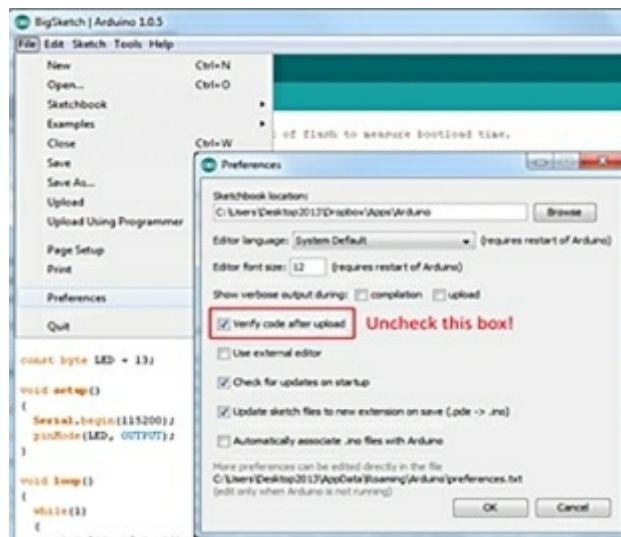
These are devices that can be added to your device so that you can protect your conductive traces from touching the table.



3. Double up your upload speed

This is quite easy to achieve if you follow the following steps:

- ✓ On your Preference window, deselect *Verify code after upload* and click ok. This will take the file to upload at 24 seconds when the box is checked; when unchecked, a file will take up to 13 seconds to upload.



The Arduino IDE will verify if this is true and everything is written as expected. It will appear as below:

Program Step:

Arduino IDE: Hey

Uno: Oh hi

Arduino IDE: I've got some new code for you

Uno: Great! Send it to me

Arduino IDE: Here it is... [30k of bytes]

Uno: Got it, thanks!

Verify Step:

Arduino IDE: Hey

Uno: Oh hi

Arduino IDE: I'm not sure I trust you got everything correctly. Send your flash to me.

Uno: Ok, here it is... [30k of bytes]

Arduino IDE: [Compares Arduino bytes to original bytes] Hmm, looks ok. Please proceed.

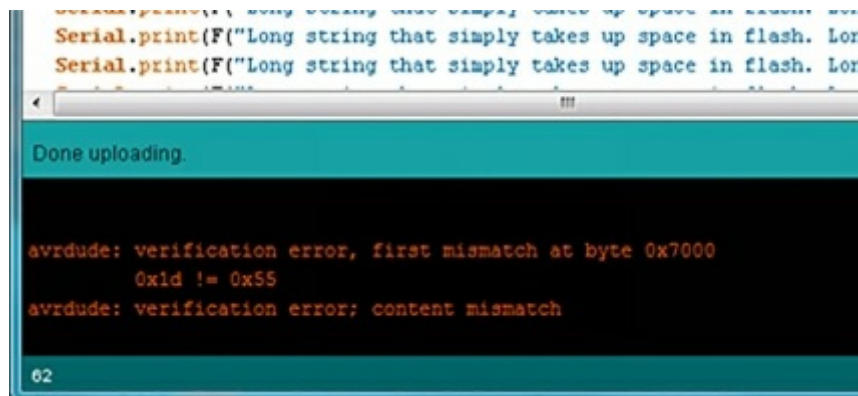
A code is sent, always during programming routines. It is the best and only way to note if any mistakes or errors took place when the transmission was taking place. Letting this recurring step reduces the range of bytes which can be being surpassed back and forth through half of. There are usually two greater mistakes checks that arise; lower at the USB to serial verbal exchange stage- as each USB has a CRC- and also at the bootloader degree wherein each stage of the STK500 bootloader having a cyclic redundancy test.

At these levels, any errors or corruption are caught, they are corrected. The chances of you uploading an incorrectly recorded sketch to the Arduino are extremely low.

There may be occasions in that you need to confirm your code and notice if it official. it is higher to sleep knowing that your mission before deploying it on your Arduino, that the code is correct.

it's miles excellent to confirm after add, when you have an excellent Arduino connection, like a 2km connection over RS485 or a 50ft USB cable. Use your own judgment even though it is unlikely for an error to slip through the CRCs.

A failed verification will look like this when you use an Arduino IDE:

A screenshot of the Arduino IDE interface. The top part shows a code editor with two lines of C++ code: `Serial.print(F("Long string that simply takes up space in flash. Lon` and `Serial.print(F("Long string that simply takes up space in flash. Lon`. Below the code editor is a black terminal window with red text. The text in the terminal reads: "Done uploading." followed by "avrdude: verification error, first mismatch at byte 0x7000", "0x1d != 0x55", and "avrdude: verification error; content mismatch". At the bottom left of the terminal window, the number "62" is visible.

```
Serial.print(F("Long string that simply takes up space in flash. Lon
Serial.print(F("Long string that simply takes up space in flash. Lon

Done uploading.

avrdude: verification error, first mismatch at byte 0x7000
0x1d != 0x55
avrdude: verification error; content mismatch

62
```


Boards that work with this:

LilyPad Simple, Fio, or Uno, are examples of serial to USB IC that most Arduino have. They all have an avrdude bootloader that by default, uses the verification flag.

Boards like Leonardo, Micro, and others that use Caterina bootloader or Teensy boards that use Halfkay bootloaders that do not have any speed advantage but have faster bootloaders.

Chapter 7

Arduino Module: Tricks and Tips

It is quite easy to incorporate computer programming into littleBits when you have an Arduino Module as it is built on Arduino programming environment.

In case you are a beginner in programming microcontrollers, what littleBits does is to care for the electronics while you are busy doing your coding. This module provides resources for the Arduino community, which entails community support.

There are usually three outputs and inputs that enable you with advanced hardware interactions, to program them or permit you communication with your software. It all depends on the extent in which your imagination runs and reaches.

In this chapter, we are going to go over the some of the tips and tricks that will help you make the most of the Arduino Module.

Analog vs. Digital Input:

There are two types of input signals that can be read by the Arduino Module. These are analog signal and digital signal.

- Analog signal is not all about “on” and “off”. These signals work like a volume knob or dimmer switch. They are typically given esteems in the vicinity of 0 and 1023 in the Arduino coding dialect.
- When you turn the handle up in a clockwise way, that is whether you are associated with a dimmer module to your Arduino, the esteem will increment in a consistent way from 0 the distance to 1023.

Sources of info checked a0 and a1 on the Arduino module likewise acknowledge simple signs.

- Digital signal is simply an “on” and “off” signal. It is the signal that is similar to that of a switch or button or trigger. HIGH signal is actually ON and a LOW signal is OFF in the Arduino coding language. On the Arduino Module, all three inputs are able to read digital signals.

Voldemort and Harry Potter: Make Selections using Analog Input

The following is a case of two coin cell control modules that are snared to dimmer modules which are likewise simple flag. From the two one dimmer is selected to be very high while one dimmer is switched greatly low.

At the point when the Voldemort dimmer is stopped, the speaker adjoined to the Arduino plays his selected tune. When the Harry Potter dimmer is connected to, the signature music played is his. The theme music changes, for instance, if the Harry Potter dimmer is still connected and the dimmer is turned all the way down.

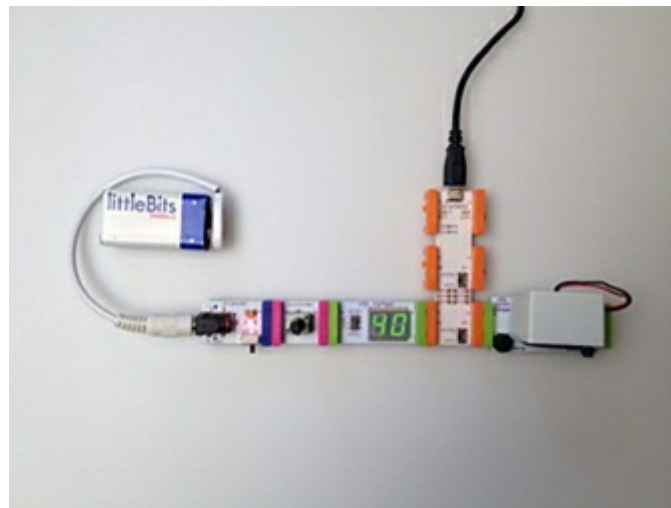


How does this work? The program to run with is selected by the analog signal from the dimmer. Parameters can be set, if you so desire, in the code. Inputs that are between 0 and 511, Voldemort's music will play while inputs that are higher than 511 will make Harry Potter's music to play.

It is really feasible for you to make more than two choices when you have values that range from 0 to 1023 when you are utilizing simple information. The following is an 8-bit Jukebox extend that utilized light sensitivity with four different light routes to make número cuatro shifting choices that utilization just a single info.



It is always difficult to tell the value you have chosen when you use an analog input as a dimmer because of the various selections you use. To help you solve this problem attach to your dimmer a module with letters or numbers, to create ease when you input! You can make 100 different selections from it!



Within the venture above, all we wanted to do is choose from four unique songs and notion of using four special buttons to make the selection simpler.

Logic modules: Build your own

To make littleBits circuits more complicated, logic modules are what you need. Add in very easy coding software and your Arduino can be changed into quite surprising logic modules that you desire.

To achieve this, you need to know the two basic Arduino coding concepts:

- i. use the Boolean Operators concepts of “and”, “or” and “not.”

- ii. The If statement. This works if A happens then B is what you do next. Otherwise, option C is what you will follow up with if B does not work.

Different logic modules could be built from putting the two coding concepts together. For instance, below is a Double AND Module:

```
void setup()
pinMode(A0, INPUT);
pinMode(A1, INPUT);
pinMode(5, OUTPUT);
void loop()
if (digitalRead(A0) == HIGH && digitalRead(A1) == HIGH)
{
    digitalWrite(5, HIGH);
} else {
    digitalWrite(5, LOW);
}
```

A translation of what is written above: IF an excessive (“on”) is used on a0 pin AND the excessive input on a1, THEN on the d5 pin is the high signal, if the inputs aren't high, then a low signal (“off”) is dispatched to pin d5.

The best and greatest way to be introduced to coding is through you personally creating logic sketches. Try and modify the code given above to see what other modules you can come up with.

Programming Basics

The Arduino Software Integrated Dev Environment is responsible to code the Arduino UNO. Go to Arduino, click on Genuino UNO at the Tools option and click on Board Menu. This is different to the microcontroller you are using. The bootloader in the ATmega328 is normally set when the Arduino is manufactured. The bootloader makes it easy to upload code, this is done using the help of the STK500 protocol.

If you are a good programmer, using the In-Circuit Serial Programming, you can bypass the bootloader and start coding with the aid of the Arduino ISP. The ATmega8U2 in the Rev1/Rev2 boards or the ATmega16U2 source code is accessible in the Arduino repository. It also has a DFU bootloader which

turns on when you connect the solder jumper that is found behind the board and rest the 8U2 on Rev1 board. It can also be activated when the resistor that pulls the 16U2 or 8U2 HWB to the ground, makes placing the DFU mode easier.

In windows, you can upload a new firmware using Atmels FLIP s/w, while in Linux environment or the Mac Os, you can use the DFU programmer.

Chapter 8

ArduinoISP

Definition

The ArduinoISP is an In-system-Programmer that is mainly used to run and program AVR microcontrollers. Most people always, if not often, after studying and learning how microcontrollers work for them to create some projects to their liking. The main reason people learn about these is to avoid giving up their dev. board.

It is possible for you to upload sketches directly on your AVR-based Arduino boards using ArduinoISP without using a bootloader. On the other hand, you can still use to restore your bootloader.

At times, making and having your own Arduino variant is what users always want. This is a way to get creative and get your creative juices flowing. It is likely that compatibility with IDE will be okay. What is common among most users is how they can burn a bootloader onto an AVR chip. Since they are blank when bought, you are required to set an IDE that is compatible with it; this is only achievable if you have AVR programmer like USBtinyISP.

It is easy to burn a bootloader onto your AVR chip quite easily. We are going to learn how soldering a 28-pin ZIF socket onto a proto shield makes a permanent bootloader burner. This is going to be epic! You can also generate a clock by using the PWM output line of an Arduino.

This process will help you restore chips that have been wrongly situated in the oscillator category or you can also make changes to the ones that have been set from an external oscillator, which is the case for most Arduino bootloaders, to internal ones like Lilypad.

The equipment required are:

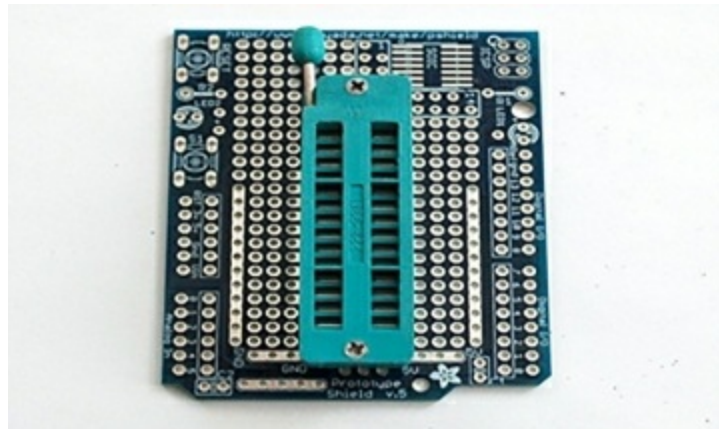
- As seen an Arduino
- [a](#) proto shield kit
- A 28-pin ZIF socket- a plain socket can be used but it is ideal to use a ZIF
- [A](#) couple of wire
- One blank ATmega328P

There will be extra few items with your kit if you bought it from Adafruit. The items might include a Piezo beeper, buttons, LEDs among other things that can be useful when doing a Standalone version of this process. Keep them to the side; they might come in handy on another occasion.

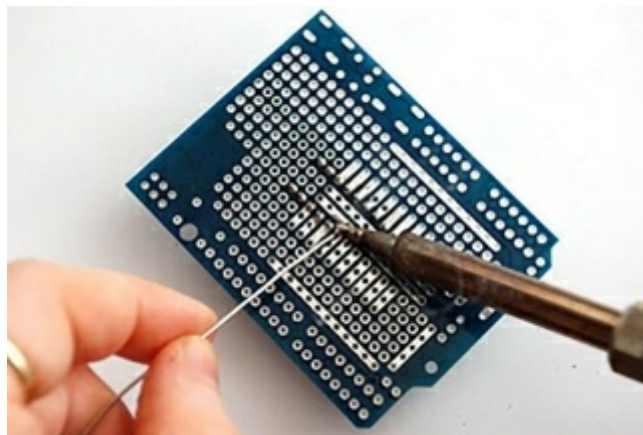
Now to Assemble the item:

How the process should be:

- The ZIF socket should be placed onto the proto shield like shown below:



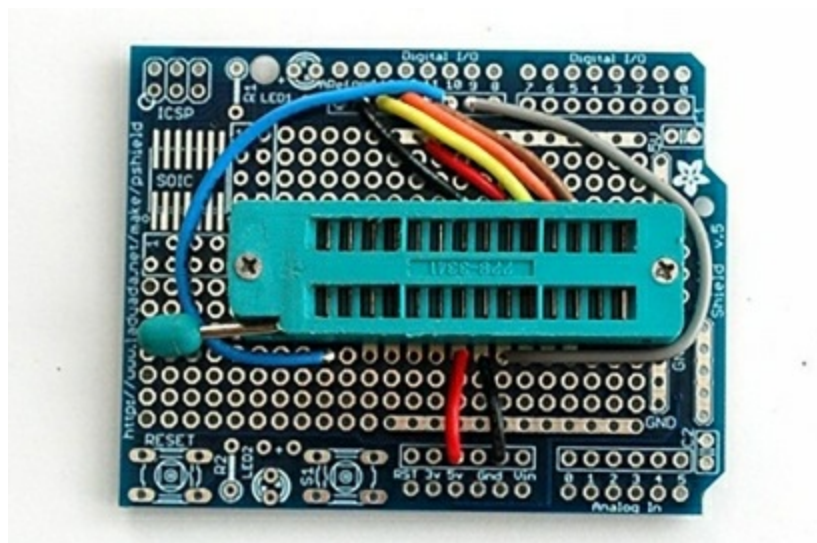
- Make sure that all 28 pins are soldered properly to enable proper connection!



- The wires on the Arduino should be soldered to the ZIF socket as instructed below:
 - Pin 7 to 5V - Red
 - Pin 9 to digital 9 – Gray
 - Pin 22 to Ground - Black

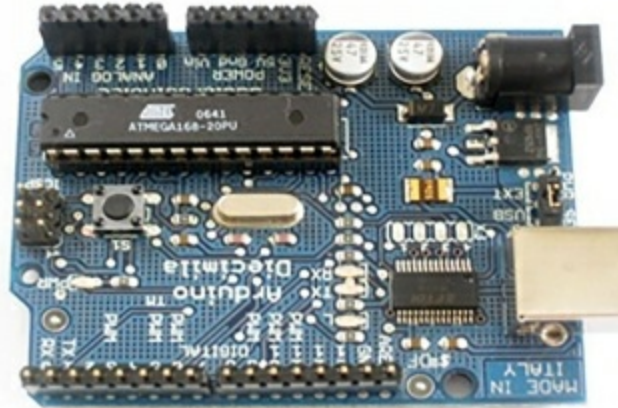
- Pin 19 to digital 13 - Yellow
- Pin 17 to digital 11 - Brown
- Pin 18 to digital 12 - Orange
- Pin 1 to digital 10 - Blue
- Pin 20 to +5V - Red
- Pin 8 to Ground - Black

Do not, and I mean do not under any circumstance should you forget, when connecting the wire to the ZIF socket, to bend it, meaning the wire over underneath, when soldering.



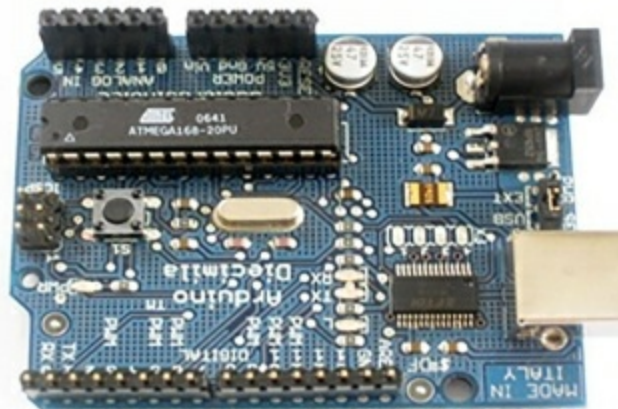
➤ Follow the photo shield process when you solder into LED1 position the Red LED and into the LED2 position, solder the Green LED. The two 1.0 K resistors should be soldered as well next to the LEDs.

Immediately afterward, solder a wire from LED2 breakout, you can identify this because it is white in color, to analog 0. From LED1 breakout, which is also white in color, take a wire and solder it to digital 8 like below:



- As a final step, solder the header. This allows the shield to be placed on the break 0.1" male header then break better off, in the Arduino sockets, place it in. Afterward, the shield is placed on top to solder it in its place.

The diagram below shows you how it should look like:



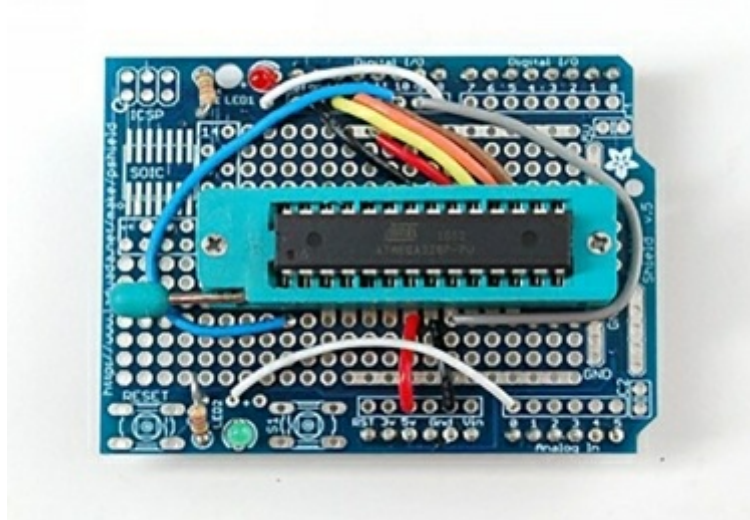
Load the Code

The procedure I am about to give you is not for Arduino 1.5.2, use the mainstream released Arduino, it is the latest, instead.

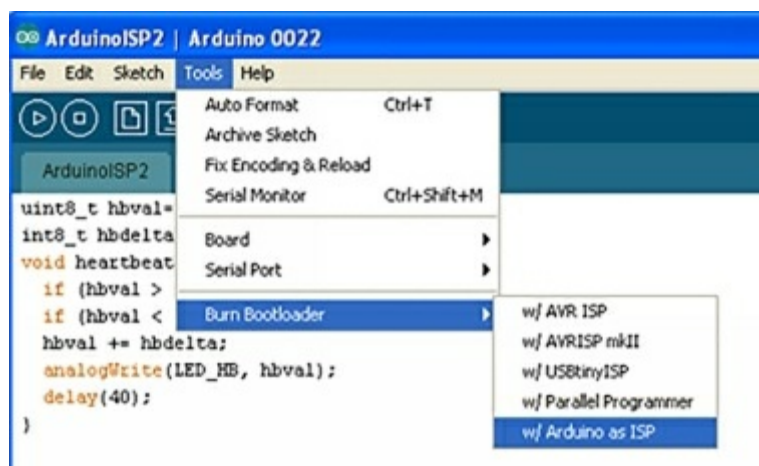
We are going to load the sketch next! From GitHub repository, get the code from here and then you can paste it into a new sketch. You can only be able to upload it to Arduino after this.

- When you put the plug at the top, lift the latch, place the chip inside and return the latch as it was before.

Make certain that the orientation of the chip is as shown below-the lever being on your left side, you are able to see the text:



- The serial port still has the USB attached to it; this is still the same that you had selected previously, click on *Tools-Burn Bootloader-w/Arduino as ISP*.



With IDE's new versions, from your *Tools-Programmer* menu select *Arduino as ISP* and click on the *Burn Bootloader*.



- Green LED is on when you are programming, after completion, the

message below will be received and it will go off.



And you are done! As an addition, different kinds of bootloaders can be burned like Duemilanove or Lilypad. It is all circumstantial and what you want to use.

How to use ArduinoISP with AVRdude

From the command line, it is quite easy for you to use ArduinoISP, especially when using AVRdude.

Summary

All the talk or rather what you have been reading in this book, how is the Arduino better or compared to other starter kits. An example is the Raspberry Pi starter kit.

Raspberry Pi is a small, itsy bitsy computer that you can use to learn about programming. It is also a general purpose computer that mostly has a Linux operating system. It runs on several multiple programs and is much more complicated than Arduino.

On the other hand, Arduino is a microcontroller motherboard that runs one program at a time several times over. It is quite easy to use. It is the best to use for beginners.

There are various uses for each:

Arduino

For repetitive tasks, it is easy to use such as opening and closing doors, driving a simple robot.

Raspberry Pi

When in need of a full-fledged computer, use this. It can be to do Bitcoin calculations or perform multiple tasks.

To help you decide on which one to use; according to the project you want to do, if you require more than two “ands” then use the Raspberry Pi but if you can describe the project in less than two “ands” then get the Arduino.

There are other microcontrollers that you should know, as a way to keep your options open and also brag to people that you finally! know what Arduino is and what other options are available.

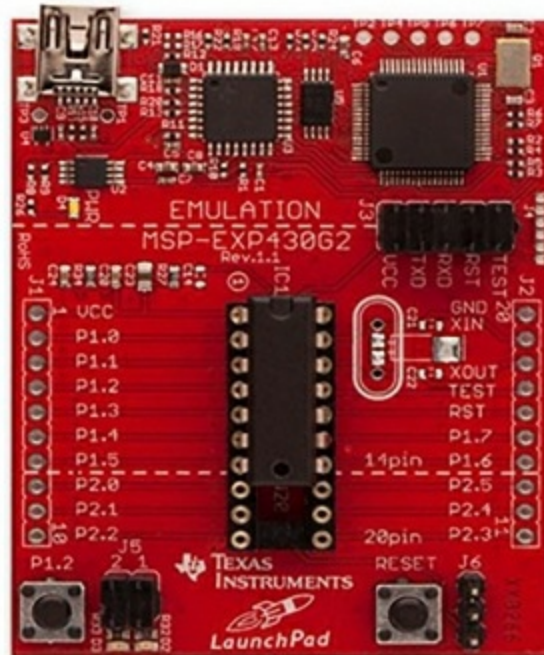
1. MSP430 LaunchPad

This microcontroller is of low-cost, low power consumption that is manufactured by BeagleBone. The LaunchPad kit comes with a second chip and costs \$4.30.

The LaunchPad's chip grants a saving mode; power saving mode to be specific that does awaken immediately making it an excellent option for

remote sensors. It is only partial o what Arduino costs and is a great alternative for simple projects and if you do not want to splurge.

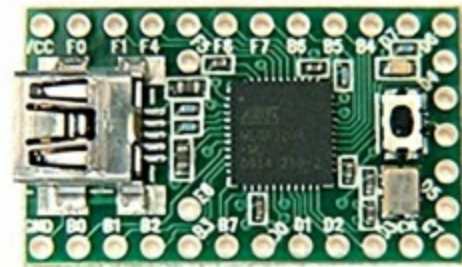
The downside is that it has only 512 bytes of RAM while Arduino has 2KB.



2. Teensy 2.0

This board and its namesake support and run Arduino software, sketches, and libraries. This makes this tiny 1.3 by 0.7 inch of hardware is great to be used by advanced users who want to move on from Arduino.

Teensy++ 2.0 is slightly larger than Teensy 2.0, on the other hand, it can be used to embed it into projects that do not require tons of space. For \$20, you would get Teensy 2.0 and for \$27.50, you would get Teensy++ 2.0. They have 16MHz AV processors.



3. Nanode

It was designed to work in a similar manner to Arduino but mainly for projects that are Internet-connected. Nanode has an ATmega328 processor that is the same as Uno and it is possible to program it using Arduino IDE.

To connect to the Web it has an open-data API Cosm. It can be used to perform tasks such as send data to cloud, perform as a smaller website or you can use it to follow up on online feeds. For web-connected sensors or controls, it is a good development tool.

It sells for \$56.57, which is higher than Arduino. Get that soldering iron ready as it requires you to assemble it!



4. STM32 Discovery

It is from STMicroelectronics, which is a low-cost alternative, selling for \$9.88. It has more power than other low-costing microcontrollers though it has a 32-bit ARM Cortex M3 core that runs at 24MHz with 8KB of RAM.

The downside to this is that it has a smaller user community and documentation will be less than you are used to with the Arduino, but it will help you out in the long run.



5. Pinguino PIC32

This tool was mainly created for art students. The shape and size are similar to that of Arduino Uno. In order to run open-source IDE, it has open-source hardware.

This board is not supported by Arduino. You are advised that Pinguino will most likely not work with Arduino sketches and/or libraries.

This microcontroller sells for \$25.99 but it is advised, by the company-take this seriously, extremely seriously- that you as a buyer are comfortable and experienced with the technology first and foremost. This microcontroller is not good to be used or handled by beginners.



These are not the only microcontrollers in the world they are just the less

known. It is all up to you and what you want to use, depending on your project.

Conclusion

Awesome job! You have completed reading this book! Congratulations.

I hope that after reading the book it will aid you in installing a library in your device of choice, make the most out of your Arduino module and also get to install on your own the new ArduinoISP to your board.

What you need to do next is to learn about coding and various ways to create new logic modules and further what you have already learned from this book and pass it on to someone else.

Coding can be of great use to you, maybe in the close future. The amount of coding language that is there in the universe is great and being part of the crew (like being part of a dance crew, hehe) you will protect your device better and get to teach someone else. It is something considered for geeks and the nerds, but t currently, being a nerd or geek is the new thing! Brains and beauty my dear reader, always brain and beauty.

Mayhap the next time someone has an issue with their chip or library, you can offer insight on what went wrong and how to rectify it. And retain the bragging rights of having learned it on your own!



Finally, if you enjoyed this book, and found it in any way helpful and resourceful, I would like to request you for a favor, a huge favor, please on Amazon, leave a review for this e Book and rate it, please! I will appreciate it immensely.

Gracias!

ARDUINO

***Simple and Effective Strategies
to Learn Arduino Programming***

DANIEL JONES

Introduction

Thank you for downloading this book, ***“Arduino: Simple and Effective Strategies to Learn Arduino Programming.”*** It is a great choice for you to better understand this programming language for effective creation of LED solutions for all kinds of purposes.

The world is growing into advertisements and lighting in all manner of ways. Motorcycles, helmets, public vehicles and even bicycles are some of the few places where we find LEDs being used a lot. Some of the uses are in directional relay, braking systems, and even for relaying a message. There are also interesting billboard signs that are made from LED lights that make an image look like it's moving, because of the way the LED lights blink in a particular pattern.

To understand how this system works, you need to understand how to create programs, which are rather called sketches, to make this particular sequence work for you. To that end, we are going to show you the basic elements of Arduino, how it works before we get into the heavy part of coding. You will want to have some little knowledge of coding to make the strategies in this book easy to grasp.

You will learn the functions to use when you want to code very easy and even complex algorithms that can make your Arduino deliver the best patterns. You will also get some cool and tough assignments, that will make you think out of the box, and stretch your understanding of what we have learned to the next level. You will also look at some points that highlight the tricks that a programmer can use to make Arduino programming easy. We all know that making the perfect system is not possible as a start, but we most definitely know that this book is a great read, informative, and it will form the first base of programming. Do enjoy the read!

Chapter 1

Introduction to Arduino

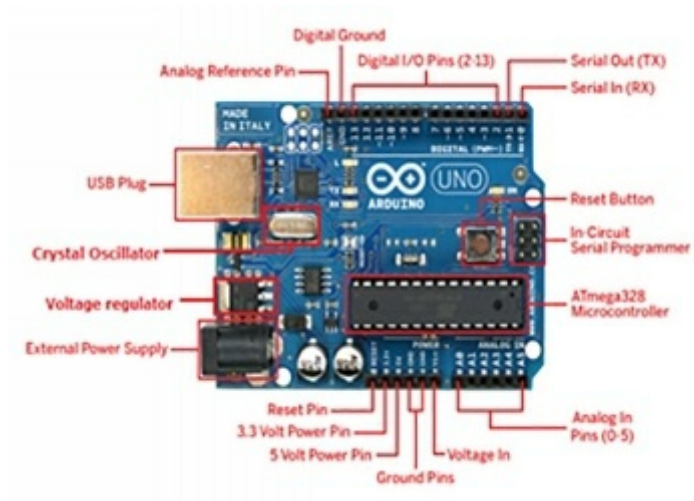
Before we get into the basic sketches that you can do in Arduino, let's first have a short overview of what is Arduino is. Arduino is an open source prototyping platform based on user-friendly hardware and software. You must have heard some people saying that Arduino is a microcontroller, which is wrong. Technically, it has a programmable circuit board which is called microcontroller.

The key features of Arduino can be summed up to:

- Digital or analog input signals are read by the Arduino boards which are then turned into outputs to activate a motor, turning on/off a LED and multiple of actions. This board can be controlled by the set of instructions sent to microcontroller through and uploading software knows as Arduino IDE, which we will discuss later on what it is all about.
- In order to load a new code onto the circuit board, you can just simply use a USB Cable that you can connect to your computer. In short, a hardware or programmer is no longer necessary.
- Arduino is user-friendly, meaning you can learn to program it easily since it uses a simplified version of C++
- Lastly, a standard form factor that breaks functions of the microcontroller into a more accessible package is provided by Arduino.

Arduino Board Components

Even though there are different Arduino boards available, they are all programmed through the Arduino IDE. Now, let's discuss the different components on the Arduino Boards. We will use the Arduino UNO board since it is the best board to get started in terms of electronics and coding. Most Arduino boards have these following components in common.



USB Plug

This is used for uploading sketches from the computer to the board. This is also one way of supplying power to the Arduino through the USB port by connecting to your laptop or PC using a USB Cable.



Barrel Jack/ External Power Supply

Another way of supplying power to the Arduino is through the barrel jack. A barrel jack connector allows connection from a battery or an AC-to-DC adapter to the Arduino. This kind of connection is usually used when the project is not anywhere close to a computer.

Voltage Regulator

The voltage received by the Arduino is controlled by the Voltage Regulator, it also stabilizes the DC voltages utilized by the processor.

Crystal Oscillator

Arduino calculates time through the crystal oscillator. The frequency of the

crystal is 16 MHz for high fast clocking.

Reset

Resetting makes the controller start from its initial state and will restart the program it has in its memory.

Resetting the Arduino can be done in two ways. You can press the reset button or use reset pin by connecting an external reset button to it.

Power and Ground Pins

3.3V. A 3.3 output volt supply; generated by FTDI (Future Technology Devices International) chip.

5V. A 5 output volt supply used to power the microcontroller.

GND (Ground). Ground pins used to ground the circuit.

VIN. This pin is an input voltage to the Arduino when an external power source is utilized.

Analog Pins- consisting of 5 input pins A0-A5. The signal from an analog sensor is converted into a digital value to be able for the microprocessor to read.

Microcontroller

Terms to know:

I/O- Input/Output

IC - Integrated Circuit

PWM- Pulse Width Modulation

DIP- Dual-Inline-Package

SMD- Surface Mount Device

These microcontrollers are usually manufactured by ATMEL Company. You can say that it is the brain of the board because it executes the instructions in your program. The main IC (Integrated Circuit) varies from board to board.

ATmega328- used on Arduino UNO R3. Its data-bus architecture and internal registers are designed to handle 8 parallel data signals which make it an 8-bit device. It has 14 Digital I/O pins, 6 of which provide PWM output. It has 32 KB Flash memory, its SRAM is 2 KB and 1KB EEPROM. Other microcontrollers are ATMEGA32u4, ATMEGA2560, and AT91SAM3X8E.

ICSP pin

In-Circuit Serial Programming pin. This pin is mainly used for boot-loading and to communicate with the SPI (Serial Peripheral Interface). Meaning that this pin is used when an external programmer or another Arduino microcontroller will be used. It is also considered as an expansion of the output.

Power LED indicator

When you plug the Arduino into a power source, the LED lights up which indicates that your board is correctly powered up. When it doesn't light up you must check your connection and see what is wrong.

TX and RX LEDs

TX means transmit and RX means receive. The TX led flashes when sending serial data while RX flashes when receiving data.

Digital I/O

There are 14 digital I/O pins on the board of which 6 of these provide PWM output. If you configure these pins they work as input digital pins that can read logic values 0 or 1, or as digital output pins to drive different modules such as relays, LEDs, and etc. "~" labeled pins can be used to generate PWM

AREF Pin

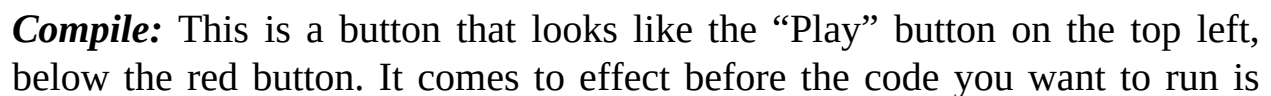
The reference voltage used for analog input and is configured by the analogReference. Anything less than 0V or more than 5V can't be used as external reference voltage on the AREF pin because it can possibly damage the microcontroller.

The Integrated Development Environment (IDE)

This is the main requirement in building, opening and editing sketches. These sketches are what we know as programs in everyday programming. Sketches are what defines what the board does. First, you need to download IDE.

- Go to <http://www.arduino.cc>. Download the latest Software. You're going to have to drag and drop it into C: Program Files once it is downloaded,
- Using the USB cable, plug in the Arduino to your computer using the USB cable. Go to Start > Control Panel > System and Security > Device Manager.

- We are now going to look at parts of the IDE. We are going to start from the left to right, to the top and lastly to the bottom.
- There are buttons that are placed along the top, on the menu items of the IDE.



sent to the board. This is how you get the code converted to instructions that the computer/board understands. This is what is referred to as compiling, in programming.

Stop: This is the second button that has a box at its mid part. It is right next to the compile button. It is used to stop the process of compiling. It is one of the buttons that at times, it does not make any sense because most programmers rarely see the need for its use.

Create: This is the button, third from the compile button. It opens a new window when you want to start creating a sketch.

Opening an already existing sketch: This is the fourth button from the top left. This will load a sketch from its storage place on the computer.

Save sketch: This is the fifth button from the top left corner that has a downward facing arrow. It is used to save the changes to the current sketch being worked on.

Upload to Board: This is the 6th button from the top left button. It is used to compile and transmits the data to the board via a USB cable.

Serial Monitor: This is the central communicative part that links the computer and the Arduino.

Tab Button: It is the only button on the top right corner of the window. This is what is used to create several files in the sketch you are working on. It is used in advanced programming which will be discussed later on

Sketch Editor: this is the white space in the middle of the window that provides a programmer a place to write, edit and update the sketches.

Text Console: This is the black strip that is located at the bottom of the window. It is used to show that the IDE is working on. It also displays the error message if a mistake is seen when you are typing your sketch. These errors are commonly referred to as syntax error.

Line Number: At the bottom of the window, there is a line number that Located at the bottom left. This is the number that shows you where the cursor is on. It is most useful when you want to make changes on the error messages flagged. Compiler messages are normally listed according to the line number; this makes making changes quite easily.

The first circuit

As we start getting to our first code, we are also going to look at how a Light Emitting Diode (LED) is connected. A diode is a one-way system, in that, electricity only flows in one direction, if you try to hook it up in the reverse order, it won't work. If the power source is directly plugged to the LED and it is ground, the diode will be destroyed because there will be a lot of current going through it.

To avoid that from happening, a resistor will be used to limit the transmitted current in the diode. A resistor, if you look at it in a lay man version, it works like a water hose pipe. The resistor's high value works as a small pipe that allows very little electricity flowing through it. It is not an accurate representation of a resistor, but it is close enough to the resistor's function.

Resistance is normally represented by the SI unit ohms. There are bands on the resistors that are color coded. These bands are used to make an individual know the value of the resistor. A 330-ohm resistor will be active. On this particular resistance, you will find the following color bands: Orange to orange to Brown. Resistors are not hard to use because any way you plug in the resistor is okay.

On the LED, there are two ends of that wire. The two "legs" or leads are what we call the cathode, and on the other opposing end, we have the anode. The longer head is the anode. When you plug it in reverse order as we discussed earlier, it won't work, nor will it be damaged, there is no need to worry about that.

Steps of making our first circuit

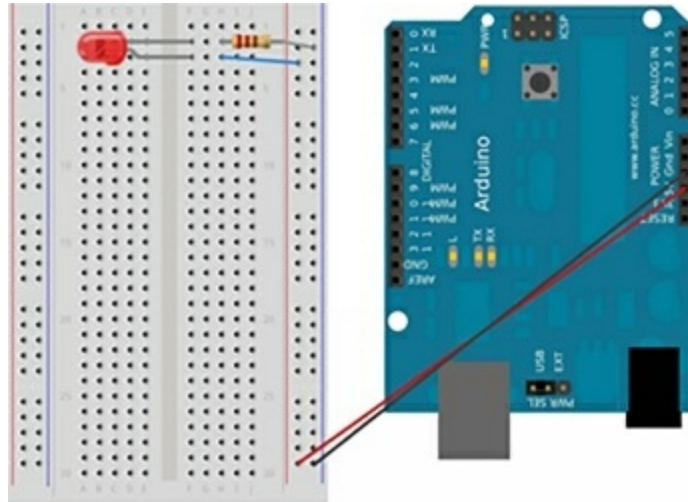
- 1) Connect the ground using a wire to the row at the board's last right column on the bottom row, on a place labeled GND.
- 2) Connect the power that is on the Arduino place labeled 5V, to the bottom row- right column. The V stands for voltage.
- 3) Connect the resistor to the ground column that is far right, to the h2 end.
- 4) Connect the short leg of the LED, cathode, to f2. Through the breadboard, the resistor is the most important part of this connection. Therefore, this connection gets attached to the resistor.
- 5) On point f3, connect the positively charged electrode.
- 6) From h3, adjacent to it, connect a conductor to the column located on the

right

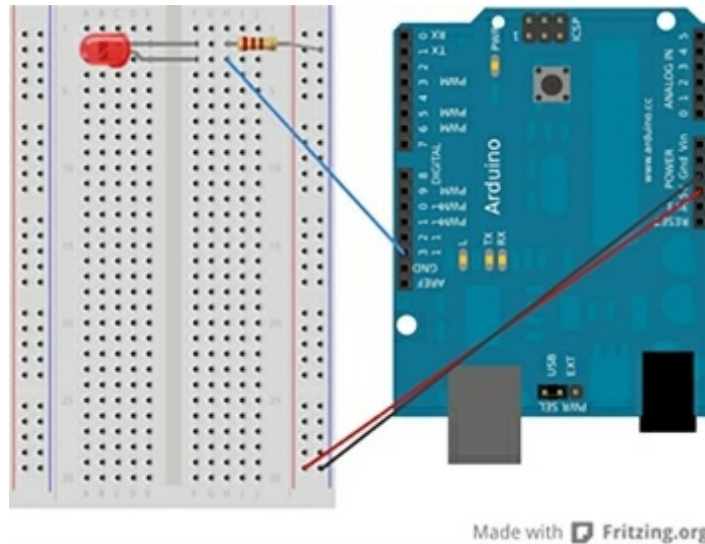
7) power up the Arduino by plugging in power.

8) You should notice the LED light up. If the LED is still off, check the power by unplugging it; check to see that all your connections are well done and make sure that you have not connected the LED in reverse order. After that, plug the power again.

You will have successfully made your first circuit when the LED lights up.



To have the Arduino control the LED of the circuit, some modifications will have to be done, and we are going to show you how. Pin 13 that is on the Arduino will be connected to the wire from h3. It is possible for any pin to work; but in this instance, it is this pin that is being used because Arduino lights up an LED when the default program is run. The LED on pin 13 can be easily used to check if all is well without looking for any other software.



NOTE: Always ensure that you unplug the Arduino before changing any circuitry changes.

The first Program

Now, we need to write a sketch that will control the LED. It is mandatory for the sketch to have 2 functions, which is a basic requirement. Now, a function is a programming statement that is called the title it holds inside a program. The two MUST functions are as follows:

1. setup () - This is a usually called at the start of the program
2. Loop () - When Arduino has power, this function is initiated several times

With this in mind, the shortest form of Arduino Program that is valid goes something like this:

```
Void setup () // start
{ }
Void loop () //continuous repetition
{ }
```

In programming languages, there is always the well-known program, “Hello World” that is showcased on the screen. To the microcontroller world where we are in, this phrase or first program is indicated by a blinking of the light, “on” and “off.” to show or see that everything you have set up works perfectly, this is the simplest test you can perform.

We shall be looking at sketches in its entirety and explain the details after the

code. If you go through something that you cannot make something out of it, keep on reading, and it will be clear.

Let us look at this program, to show you how we shall be breaking down the codes.

```
Const int PinkL = 13;  
Void setup ()  
{ pinMode (PinkL, OUTPUT); }  
Void loop ()  
{digitalWrite(PinkL, HIGH);  
    delay (600);  
    digitalWrite(PinkL, LOW);  
delay(600); }  
On the first part  
Const int PinkL = 13;
```

This line is used to define a constant that is used throughout the program to specify a particular value. All pins are recommended to have this because it makes it easy for software change if the circuit is still the same. In programming in Arduino, the constants are commonly named starting with the letter “k”. It makes it easier when going through the code to identify constants.

The second to part

```
Void setup ()  
{pinMode (PinkL, OUTPUT);}
```

The OUTPUT is pin 13. This now makes Arduino control the coding to the pins, instead of reading from it.

The third part

```
Void loop()  
{digitalWrite (PinkL, HIGH);  
delay(600);  
digitalWrite(PinkL, LOW);  
Delay(600);}
```

This is where the core part of the code is. A HIGH is written to the pin that leads to the turning of the LED. When you place HIGH, it means that 5V is

the pin's output. The other option we have is LOW, which means that you are putting 0V out.

A delay() is called to delay the number of milliseconds that is sent to it. Since we send 600. There will be a delay of 0.6 of a second. The LED goes off, and this is attributed by the LOW that is written as an output on the pin.

A 600 milliseconds delay will be activated.

This will be the sequence until the Arduino goes off or the power is disconnected from it.

Before you start digesting more content, try this program out and ensure that it works just fine. To test if you have set your LED in reverse order, the following might happen. On the UNO board, you have pin 13 connected to a Light Emitting Diode connected. When it blinks and the breadboard LED does not blink, then you might have connected your LED in reverse. In case you see that it is blinking once in a second, then the program has not been sent to the Arduino successfully.

Placing comments

The programs we normally write are usually meant for the computers and not for people to understand once they are opened up. There is a good provision that allows us, humans, to read the program easily and the computer will have no clue about it. This is made possible by the use of comments that are added to the program. There are two comments that are possible in this program:

1. The block comment style starts with two characters, /* which progresses until */ is seen. Multiple lines are then crossed and here are a few examples.

```
/* This is the first line*/  
/* the program was successful*/  
/* we  
*are  
*going  
*far */
```

2. Commenting can be done on a line that has the backslash operator //. this is the part that is meant for human and not machine. It is another way to place a comment.

When you add comments in a program, you will have a code that looks like

the statement above.

You will find in the following pages, that if there is no number next to the line of code, it indicates a comment continuation from the line at the top. We might not showcase this in perfection because we are using a limited space in our book. You will find a hyphen at the line's end that is continued and a hyphen along the continuation line. This is just our way of handling it, but in an IDE, you won't find it and you need not type them.

```
/*  
  * Program Name: Blink123  
  * Author: James Aden  
  * Date written: 24 July 2017  
  * Description:  
  * Turns an LED on for a sixth-hundred of a second, then for another  
  * sixth-hundred of a -second on a continuous repetitive session  
  */  
/* Pin Definitions */  
Const int PinkL = 13;  
/*  
  * Functions Name: setup  
  * Purpose: Run once after system power up  
  */  
Void setup(){pinMode(PinkL,OUTPUT);}  
/*  
  Void loop(){digitalWrite(PinkL,HIGH);Delay(600);  
  digitalWrite(PinkL,LOW);Delay(600);}*/
```

Gotchas

If you find out that your program does not compile, or it gives you a different result than what you need, here are a few things that people get confused about:

The programming language is normally sensitive to case of letters. For instance, myVar is considered different to MyVar.

Tabs, blank lines, and white spaces are equivalent to a single space, making it easier for one to read.

Code blocks are normally grouped using curly braces, i.e. “{“ and “}”

All open parenthesis have a corresponding closing parenthesis, i.e. “(“ and “)”

Numbers don't have commas. So instead of writing 1,000, ensure that you write 1000.

All program statements **MUST** end with a semi colon. This means that each statement except the following two case:

- In comments

- after curly braces are placed “}”

Assignment task to test what you have learned:

1. Alter the delay time of your LED before it comes back on to stick to 1.5 seconds. Leave the ON time of the LED limited to 600 milliseconds.
2. From pin 13, change to pin 2, making it the new connection to the LED. Keep in mind that both the circuit & and the program will be different.

Chapter 2

Light the World

If Statements

An IF statement is a control structure that allows you to alter the way a sketch is implemented and even allow you to run code several times. It is the first control structure that we are going to look at using an example of a program.

```
Const int PinkL = 13; // this is assigning the Pin an integer value  
Void setup() // this is a function  
{pinMode(PinkL, OUTPUT);} // this is placing pin 13 as the  
OUTPUT  
Int d_Time=500;  
Void loop()  
{delayTime=d_Time - 50;  
If(delayTime <=0) { //if delay time is less than zero equal to zero,  
reset  
delayTime=500;}  
digitalWrite(PinkL,HIGH);Delay(delayTime);digitalWrite(PinkL,L(
```

Give a wild guess on what the sketch above does. Don't worry if you can't guess, but try to at first. We are going to go through it to ensure that we understand what the code does and how to incorporate the same thorough process in our programs.

By now, after going through other programs, we have already understood what is happening from “const int PinkL=13;” to the start of “int d_Time=500;”

Now, let's start off with the line that we have not looked at before.

```
Int d_Time=500;
```

This line is almost the same as the **const** line. It's just missing the **const** keyword. This line is a variable that can alter in the program, but we are assigning it a starting value of 1000. If a variable is defined in curly braces, that means that the variable can only be used in the curly braces. This is what is called as a local variable. If a variable is defined outside curly braces, then it is defined as a global variable which can be used anywhere in the code.

The code starts to get interesting as you go down. We have
`delayTime=d_Time-50;`

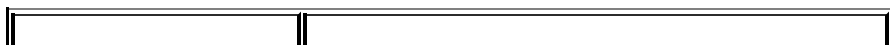
The `delayTime` value is being changed by subtracting 50 from the `delayTime`'s original value. We have the original value of 500, the new value will be 450 after this line is executed. Let's look at other operations in Arduino language.

Operator symbol	Description
=	This is an assignment operator
-	A subtraction operator
+	An addition operator
/	This is a division operator. Be cautious about it when you are using integers. With the use of integers, no rounding off is applied. For instance, $5/2==2$
%	This is a Module operator. It is responsible for providing as an output, the division remainder. For instance, $5\%2==1$
*	This is a multiplication operator

Let's dive in into the following code

```
If(delayTime<=0){//a reset is made if the value of the delayTime .  
  delayTime=500;}
```

This is the part that ensures that the light blinks. Since we are minusing 50 from 500. We are avoiding it from getting to 0 or a negative number. We can use several kinds of operators to ensure that this is achieved.



Operator symbol	Description
==	It means it is equivalent
!=	This means that is is not equivalent
<	This means that it is not more than
>	This is a bigger than
<=	This is to a value lower or same as
>=	This is a bigger than or same as

We would have just tested if the delay time that we are looking into is equal to 0, but since a negative value is not what we are looking for, the condition has to be confirmed first. In essence, this has to be done because it is very important. By now you should have known that if the delayTime <= 0, it is reset to 500.

The part that is remaining is meant to turn the LED both on and off. But, instead of a fixed number being used, to change the time delay, a variable comes to play while program is running. That's an interesting twist to use.

ELSE Statements

You can use an *else* statement with an *if* statement, if the *if* statement is false. You will get what we mean in the next few paragraphs.

```
Const int PinkL=13;
```

```
Void setup()
```

```
{pinMode(PinkL, OUTPUT);}
```

```
Int d_Time=500;
```

```
Void loop()
```

```
{if (d_Time<=-50){// if the delay time is below or equal to 50, it is reset
    d_Time=500;
```



```

    }
Else
    {delayTime=d_Time-50;}
digitalWrite(PinkL, HIGH);
Delay(delayTime);
digitalWrite(PinkL, LOW);
Delay(delayTime);
}

```

If you are observant enough, in the above code, this line “delayTime=500;” is executed only when the delayTime<= 50. The code that follows, “delayTime=delayTime-50;” is executed, but not a situation when both are executed. One of the two has to be executed,

Now, instead of comparing 0, we have compared to 50. This is done because we are comparing before a subtraction of 50 is done, but in the previous section, we compared afterward.

If a question like this pops up, what will you say?

Question: What would happen if we did not compare it to 50, but we did compare it to 0?

While Statements

This is a statement operates the same way that an if statement works, but it repeats a bunch of code when the condition is valid. Let us look at an example. And notice that there is no use of the else statement.

```

Const int PinkL=13;
Void setup()
{pinMode(PinkL, OUTPUT);}
Int d_Time=500;
Void loop()
{while(d_Time>0){
digitalWrite(PinkL, HIGH);
Delay(d_Time);
digitalWrite(PinkL, LOW);
Delay(d_Time);
}
}

```

```

d_Time=d_Time-50;
}
While(d_Time<500){// while delaytime is less than 500
d_Time=d_Time+50; //this has to be done to avoid looping with
delayTime=0
digitalWrite(PinkL, HIGH);
Delay(d_Time);
digitalWrite(PinkL, LOW);
Delay(d_Time);
}}

```

Try and guess what the above code does

Wait for it.

Answer: it starts to blink fast then it slows down

Truth statements

In this programming language, one unusual thing about it which is also found in other programming languages is that it does not define a true statement, explicitly. It first defined what is false, and then it declares everything else as true. That's how the language works. It is easy to make a programming mistake which involves confusing = and ==. You should always remember that one “=” sign assigns values to variables or constants, while a double equal sign is used to test and see if two values are the same. For instance, if we had the intention of making a light blink in a fast blinking pattern, and we used an equal sign instead of double equal signs, this would be the code.

```

Int d_Time=500;
Void loop()
{if (d_Time=0){//This should not be the case, it should have ==
d_Time=500;
}
digirtalWrite(PinkL, HIGH);
Delay(d_Time);
digitalWrite(PinkL,LOW);
Delay(d_Time);
}

```

```
d_Time=d_Time-50;
}
```

In this sample, the d_Time will be assigned 0. The digit 0 will be checked by the if statement to see if it is true. But remember that 0 defines false. It wont execute the d_Time=500, because each time the loop () function is executed, d_Time will be 0. This is not the outcome we are looking for.

For instance, if we wanted to reduce the blinking speed and reset, but we placed a single equal sign accidentally, instead of double equal signs.

```
Int d_Time=50;
Void loop()
{if(d_Time=500 {// this should not be the case, it should be ==
d_Time=100;}
digitalWrite(PinkL, HIGH);
delay(d_Time);
digitalWrite(PinkL, LOW);
Delay(d_Time);
d_Time=d_Time+50;}
}
```

The d_Time in this situation will be assigned to 50. To check if 50 is true, it will be checked by the if statement. Always remember that everything that is not 0 is true. Therefore, each time it assigns 500 to d_Time, the rate of blinking will never change. It is not easy to follow through these changes. So if you find an unusual error, check again to verify that it is all well.

Combinations

There are moments where testing many things comes naturally. For instance, you may want to test and see if two numbers can be taken by a single variable. While it is possible for people to choose multiple **if** statements, it is way easier to use logical combinations to achieve the same result. There are several ways you can do this.

Operating symbol	Example	Description

&&	(G<20)&&(H>10)	Logical AND returns true is condition G and H are true, otherwise, the result is false.
	(G<20) (H>10)	If G or H' conditions are true, Logical OR returns true, otherwise, the result is false.
!	!(G<15)	If G's condition is false, the Logical NOT returns true, if not, then it is false.

One thing that is not obvious is the fact that the NOT operator can be used as a High or Low variable. For instance

```
Int l_Status=LOW; //assigning the l_status the LOW signal
Void loop()
{l_Status=!l_Status; // it toggles value of the l_Status
DigitalWrite(PinkL, l_Status);
Delay(500);}
```

The l_Status is LOW in this instance, and when l_Status=!l_Status, it gets HIGH. When the next loop comes to life, led Status is HIGH and when l_Status=!l_Status it gets LOW.

The FOR statements

The **for** loop is a programming structure we are going to look at briefly. It is mostly used when you want a particular code to run for some time. Let's look at a sample code.

```
Const int PinkL = 13;
Void      setup(){pinMode(PinkL,OUTPUT);}Void      loop(){for(int
i=0;i<4;i++)
{digitalWrite(PinkL,HIGH);Delay(200);DigitalWrite(PinkL,LOW);Delay
Delay(200); // 200 milliseconds}
```

While you were reading the code above, you might have seen something new that looks like a mistake, but it is not. We are talking about i++. What you should know is that programs like to make coding less complex by making using different methods. So, programmers came up with several shortcuts that include the one we have seen above for things that are common.

Assigned operators can be combined by these compound operators. We shall look at other compound operators later on.

Operator	Meaning	An instance
++	Incrementing operator	G++ means the same as G=G+1
--	Decrementing operator	G-- means the same as G=G-1

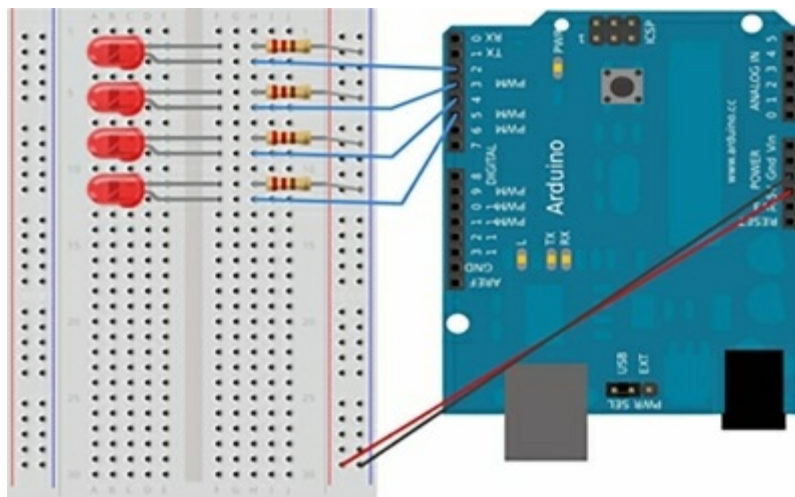
Three statements exist within the **for** statement. The composition is as follows.

```
For(first-statement;condition;second-statement)
{ //statements }
```

The first-Statement occurs first and once. The condition is tested Every time through the loop. If the statement is true, the code that is in the curly braces and the second- statement is parsed. The block's statement is accessed if the condition is false.

Brand New Circuit

We are going to take things a little bit further; we will hook up more LEDs in order to get more complex and interesting patterns.



Let us go to a step by step guide

1. Connect the ground right column, at the bottom row.
2. The power +5V IS connected to the next column on the right-side,

bottom row. It is not a necessary action, but it makes sure that you power up the ground columns.

3. LED1

- The resistor is hooked to the h2 on one end, and the opposite side on the ground (the other end)
- Connect the shorter leg which is the cathode to f2. this then connects it to the resistor using the breadboard.
- On f3, the positive electrode of the LED is connected.
- Pin 2 on the Arduino is the connected to j3

4. LED 2

- Another resistor is connected to one side of h5, and the other side is connected to the ground, which is the right corner.
- Connect to f5 another LED cathode
- Connect to f6, the same LED anode, which is the long end
- Connect to pin 3 on the Arduino, the j6

5. LED 3

- With one end on the ground, connect another resistor's end in h8
 - connect to f8, another LED cathode
 - connect to f9 the same LED anode
- Connect to pin 4 on the Arduino, j9

6. LED 4

- On the h11 end, connect a resistor and the other end of the resistor, connect it to the column in the right (ground)
- On f11, the cathode LED is connected
- On f12, connect the LED anode, which is the longer leg
- on Arduino's pin 5, connect j12

We are going to look into a code that will ensure our H/W is correct. We are also going to look at software to make sure that the H/W is correct then try the full S/W on new hardware.

The code below is used to set up LEDs on pins between two and five and in a sequential basis, the LEDs are turned on and off.

```
Const int PinkL1=6;
```

```

    Const int PinkL2=7;
    Const int PinkL3=8;
    Const int PinkL4=9;

Void setup()
    {pinMode(PinkL1, OUTPUT);
    pinMode(PinkL2, OUTPUT);
    pinMode(PinkL3, OUTPUT);
    pinMode(PinkL4, OUTPUT);}

Void loop()
    {// the LEDs will be turned on in order one by one
    digitalWrite(PinkL1, HIGH);
    Delay(50);
    digitalWrite(PinkL2, HIGH);
    delay(50);
    digitalWrite(PinkL3, HIGH);
    Delay(50);
    digitalWrite(PinkL4, HIGH);
    Delay(50);
    // each LED is turned off in order
    digitalWrite(PinkL1, LOW);
    Delay(50);
    digitalWrite(PinkL2, LOW);
    Delay(50);
    digitalWrite(PinkL3, LOW);
    Delay(50);
    digitalWrite(PinkL4, LOW);
    }

```

Even though this code works well, there are a lot of mistakes that are visible and we need to fix this.

Arrays

This is a group of indexed variables. Let us look at the example to understand

more.

```
    Const int l_num1=4;
    Const int kPinkL[k_num1] = {6,7,8,9}; // pins 6-9 have Light Emitting
    Diodes
Void setup()
    {for(int I=0; I<k_num1; i++) {pinMode(kPinkL[i], OUTPUT);}
    }
Void loop()
    {for(int i=0; i<k_kPinkL; i++) {digitalWrite(PinkL[i], HIGH);// each
    loop high
    Delay(50);}
    For(int i=k_numLEDs-1; i>=0; i--) {digitalWrite(kPinkL[I], LOW);//
    each loop low
    Delay(50);}
    }
```

What do you think that this code does?if you have no clue, we are going to get into it right away, part by part.

```
    Const int k_num1=4;
```

We will first define the elements that will be going to use in the array. This is used to make sure that there is no read or write after the array.

```
    Const int kPinkL[k_num1] = {2,3,4,5}; // pin 2-5 have LEDs
```

Second, the array is defined. Notice how the elements are in brackets. It is always better to use constants, even though we use actual numbers. The arrays are assigned numbers here. The curly braces store the values and commas separate each value. It is confusing when you look at arrays because they are zero indexed. This means that the element at the first array position in the array called k_PinkL is k_PinkL[0]. Now, k_PinkL[3] gets to be at the end of the array. There are only 4 elements, this starts from 0-3.

```
Void setup()
    {for (int i=0;i<k_num1; i++) {pinMode(kPinkL[i], OUTPUT);}
    }
```

A **for** loop is used here in order to check the array's elements as it prepares them on becoming **OUTPUTs**. Square brackets having an index inside is what is used to access each element in the array.

N.B.: It 's nice to question some things even though you might not have all the answers all the time. Like for instance, one might be wondering why we used an array instead of just using pins 2-5. To answer this question, yes it is true we could have used the pins, but it is not a good idea to use it. If for instance at a later date you will want to use different pins that are not close to one another, it is only possible to use the arrays, because pins will not work in this manner.

In the For loop statement that has // ***each loop high***. This is similar to what is done in setup. In this loop, we shall be going through each LED and with a delay time of 50 milliseconds, we are going to be turning them on, one by one.

In the For loop statement that has // ***each loop low***, it is used to go in reverse, through the entire loop. The array is zero indexed when we start with **k_num1-1**. if k_num1[4] started, it would be beyond the array. Since we do not want to miss index 0, we check ≥ 0 first.

Assignments

1. Create a sketch that ensures that a single LED is lit up 5 times consecutively for a second on and off. Then for half a second, it is turned on and off five times, consecutively.
2. Create a program where LEDs are turned on in any pattern you prefer.
3. Create a program that uses arrays that intend to light up LEDs from top to bottom, and then it starts all over again in the reverse order so that at any given time, one LED is on. This is what is referred to as Larson light or a Cylon.

Chapter 3

Programming

Name Variable

The first part of the sketch, you will have elements being named. We shall look at it later on

Programming Setup

In programming, you are telling the sketch, for instance, the type of pin will serve as the pin and where the output will be on the board. Now, as an output, the output should be put out. For instance, with this pin, a LED lights up. As an input, the voltage should be read out. The board's input pin gets a high voltage, and it is recognized.

Loop

This part will be repeated by the board over and over again. In the sketch, it starts from the beginning to the end, and then it starts from the beginning and again to the end, you get the picture.

Let's begin

LED and Sound

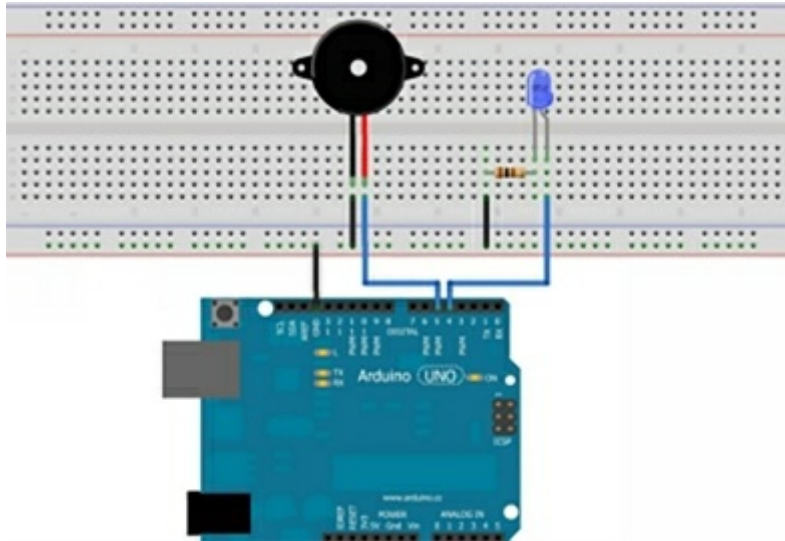
Since we have already seen how an LED blinks in the previous chapter, we are going to see how Light and Sound are programmed.

Goal: We are going to have a piezo speaker and an LED. This is with the target of the LED blinking and the speaker to beep.

Requirements

One LED, a 200-ohm resistor, breadboard, microcontroller, cables and a piezo speaker.

The Setup



The code

Int LED=4; // pin 4 has a LED

Int beep=5; //pin 5 has a speaker

Void setup()

{pinMode(LED, OUTPUT); // the output is pin 4

pinMode(beep, OUTPUT); //pin 5 is the beep output}

Void loop()

{digitalWrite(LED, HIGH); // LED lights up

digitalWrite(beep, HIGH); //it turns on the speaker

Delay(500); //a delay of half a second will be put in place for both sound and LED

digitalWrite(LED, LOW); //LED goes off

digitalWrite(beep, LOW); //turns off the speaker

Delay(500); // we wait for half a second to pass with no sound or light

}

// the program now starts from the start of the loop. It will be and light up once more. If the delay is changed, it will now change the lighting and sound speeds.

LED and Push Button

Goal: Our goal is to have the LED lights up after 5 seconds when the button is pressed

Requirements

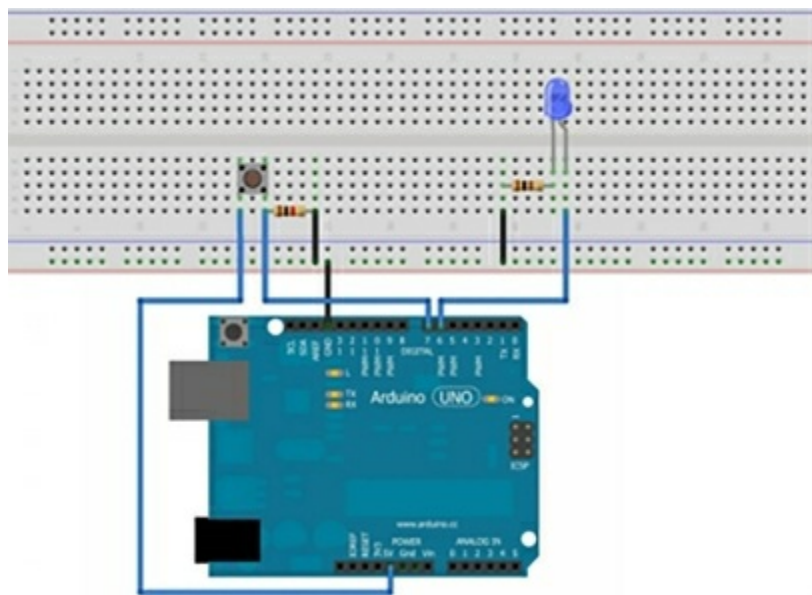
Arduino, 100-ohm resistor, 1000 ohm resistor, breadboard one blue LED, push button and cables

The microcontrollers digital pins read voltage level, but pins also have the capability of making the voltage go off. We are going to add something else to the equation, a push button. The push button accessibility to the microcontroller will allow the voltage to be on the pin. This is compared to the microscopic reaction that is on the pin.

Difficulty

Floating electrons escape slowly. The microcontroller will think that the push button has been activated. The microcontroller will assume the button is pressed, this will go on until there are no electrons on the pin. To solve this problem, the best solution is to ground the pin using a 1000 ohm resistor. The electrons escape faster, and it will be defined by the microcontroller that the button was pushed.

PULLDOWN, which is the resistor, ensure that the voltage is pulled to 0V, always. If you are using a different kind of resistor that has a different value, an electrical short will emanate when the button is pressed.



To code this in a program, it will look as follows

```
Int LEDblue=6;
```

```
Int button=7;
```

```

    Int buttonstatus=0;
Void setup()
    {pinMode(LEDblue, OUTPUT); // output is on pin 6
    pinMode(button, INPUT); // pin 7 is connected as an input
    }
Void loop()
    {buttonstatus=digitalRead(button); // pin 7's value is read out. The
    resist will be saved in buttonstatus. LOW means 0V and HIGH means
    5V
    If (buttonstatus==HIGH) // high voltage value is achieved if the button
    is pushed
    {digitalWrite(LEDblue, HIGH); //LED lights up
    Delay(5000); // 5 seconds long wait
    digitalWrite(LEDblue, LOW); //LED gets turned off after 5 seconds
    }
    Else
    {digitalWrite(LEDblue, LOW); //LED lights up}

```

Below is the code to make LED Blink

```

//LED Blink
int LEDPin = 7; //the Arduino pin that is connected to the LED
void setup() {pinMode(ledPin, OUTPUT); // initialize the pin as an
output}
void loop() { digitalWrite(ledPin, HIGH); //turn LED on
delay(1000); // wait for 1000 milliseconds (one second)
digitalWrite(ledPin, LOW); //turn LED off
delay(1000); //wait one second
}

```

Here is another Button Sketch:

```

//Button Press Detection
int buttonPin = 7;
void setup(){
    pinMode(buttonPin, INPUT); //this time we will set the pin as INPUT

```

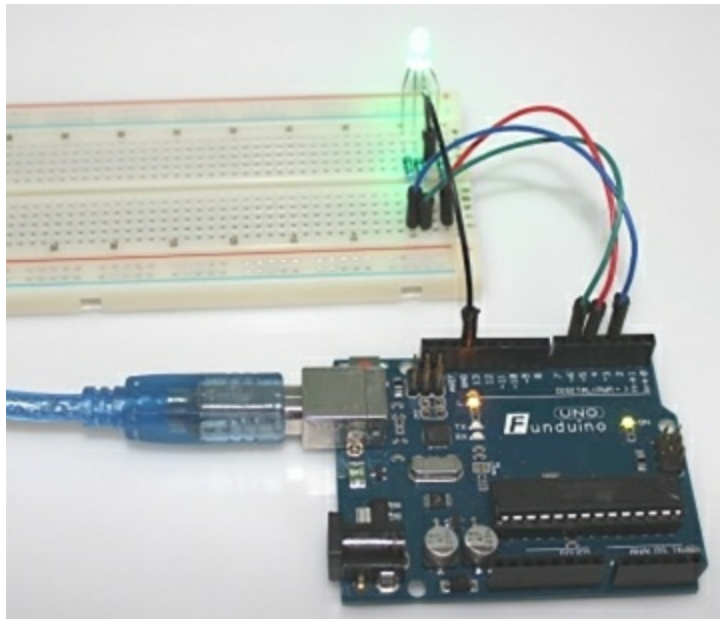
```
Serial.begin(9600);//initialize Serial connection
}
void loop(){
  if (digitalRead(buttonPin)==HIGH){//if button pressed
    Serial.println("pressed");
  } else {
    Serial.println("unpressed");
  }
}
```

RGB RED LIGHTING UP

Goal: RGB LED lighting up in different colors

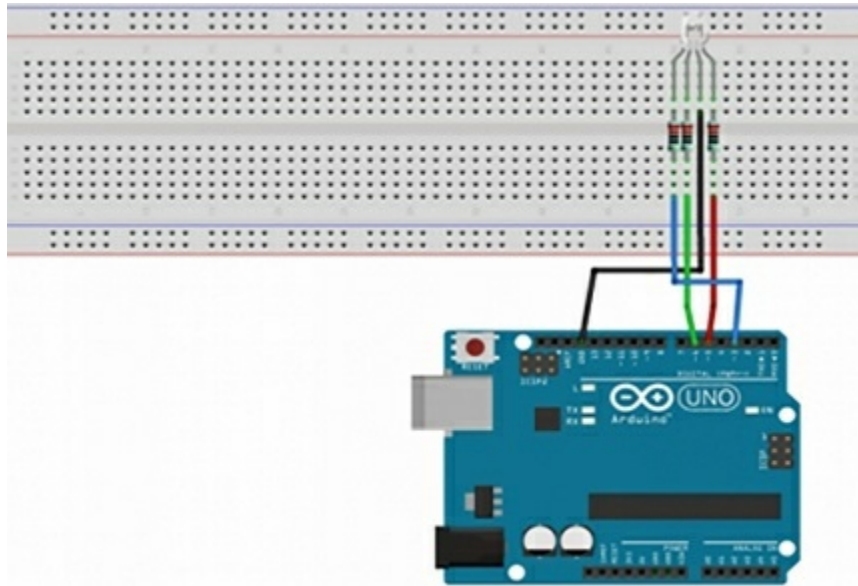
Requirements

One RGB LED, 3 200 ohm resistors, cables, breadboard and a microcontroller

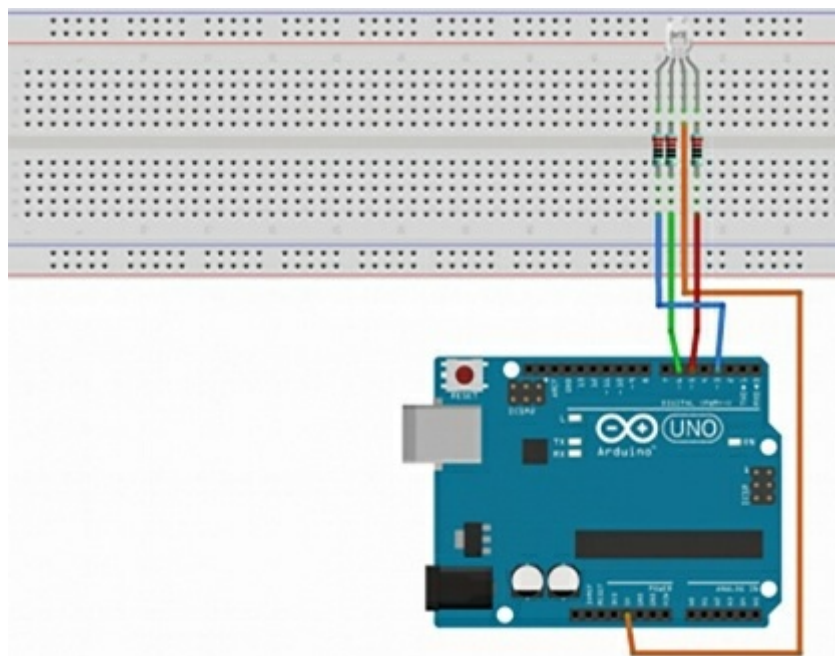


What is an RGB LED? One might ask. This is an LED that has the capability of lighting up in different colors. Most of us have encountered the initials RGB when we were in elementary school. If you have forgotten, it means, Red, Green and Blue inside this LED, and there are 3 different LEDs. They can all be lit separately and even turned off differently. When you look at the RGB LED, the longest one is the anode or cathode, it all depends on the version you are using.

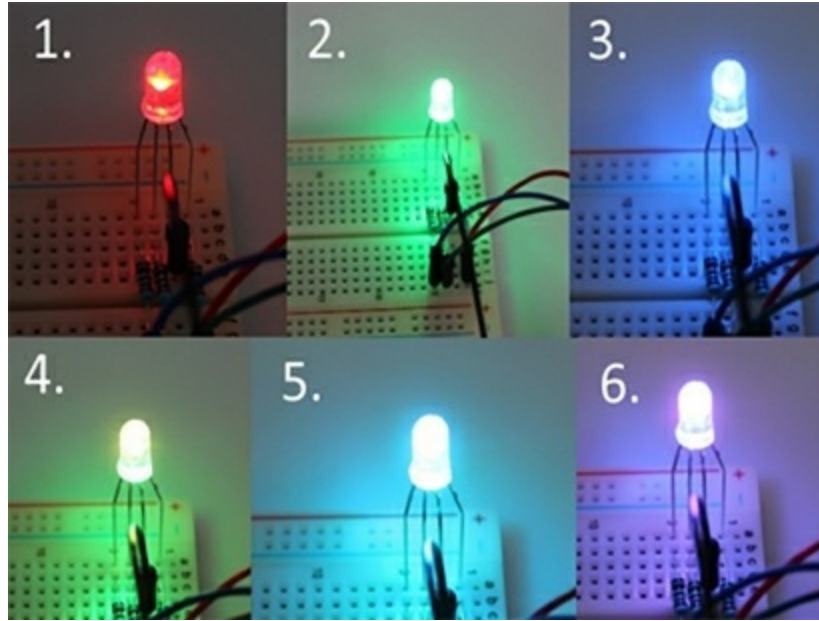
1st Version: For a common cathode, the longest contract is the cathode(-). in this instance, the other three contracts need to have a 5V (+).



2nd Version: or a common anode, the longest contract is the anode(+). it means that the other contracts will have GND(-) voltage.

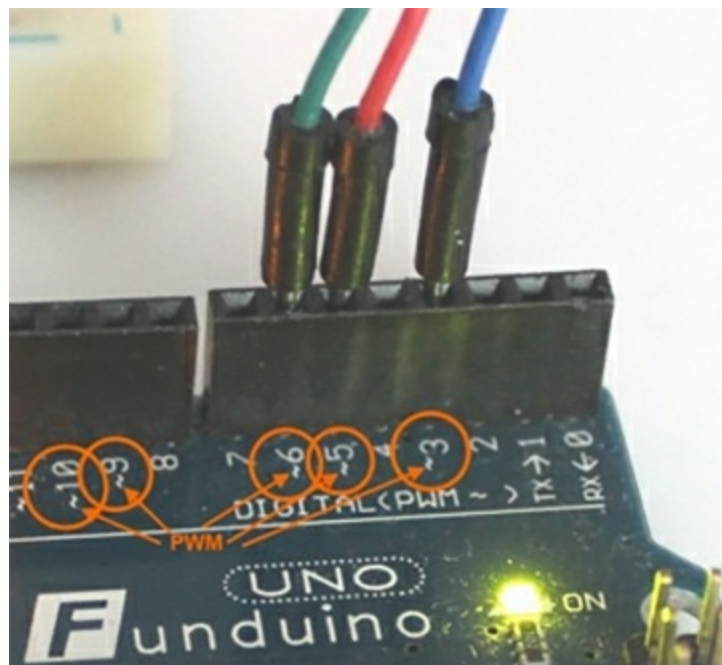


If you distort the colors, you will be able to create even more colors. For instance, if you distort blue and green, you will get the color yellow.



To find out the type of RGB that is available, change the positive terminal (+) with the negative terminal (-) in the LED. It will work if it is connected in the right way.

On the output of Arduino, which is a digital microcontroller, it can only turn 5V on and off. In order to create different colors, you will use a Pulse Width Modulation. You can use the PWM in the pins.



Voltage pulses from 0V to +5V is allowed by the PWM. This makes the voltage go on and off in milliseconds. When you have a high PWM, the 5V

signal will remain stable on the pin. When you have a PWM that is low, the alternative will be achieved, and 5V won't be there.

We are going to look at a sketch that works for the versions of RGB

Sketch 1: Idea here is to turn on and off Red, Green and Blue

```
Int LEDblue=3; // pin 3 is blue
Int LEDred=5; // pin 5 is red
Int LEDgreen=6; // pin 6 is green
Int b=1000; // with a 1 second delay
Int brightnessblue=150; // 0 to 255 defines a single color's brightness
Int brightnessgreen=150;
Int brightnessred=150;
Int dark=0; // 0 value stands for 0V, LED is off
Void setup()
{pinMode(LEDblue, OUTPUT);
pinMode(LEDgreen, OUTPUT);
pinMode(LEDred, OUTPUT);}
Void loop()
{analogWrite(LEDred, dark); // red is off
Delay(b); //break
analogWrite(LEDred, brightnessred); //red is on
Delay(b); //break
analogWrite(LEDgreen, dark); //green is off
analogWrite(LEDblue, dark); //blue is off
analogWrite(LEDgreen, brightnessgreen); //green is on
Delay(b); //break
analogWrite(LEDgreen, dark); // green is off
}
```

Sketch 2

The aim of this code below is to ensure that two colors are turned on and off simultaneously. We will be able to create more colors, like purple and yellow.

Let's begin

```
Int LEDblue=3; //pin 3 has blue color
Int LEDgreen=6; //pin 6 has green color
Int LEDred=5; //pin 5 has red color
Int b=1000; // b is the delay of 1 second
Int brightnessred=150;
Int brightnessblue=150;
Int brightnessgreen=150;
Int dark=0; // this means that LED is off

Void setup()
{pinMode(LEDblue, OUTPUT);
pinMode(LEDgreen, OUTPUT);
pinMode(LEDred, OUTPUT);}
Void loop()
{analogWrite(LEDred, brightnessred); // yellow is produced when red
and green is on
(LEDgreen, brightnessgreen);
Delay(b);
analogWrite(LEDred, dark); // yellow is off when red and green are off
analogWrite(LEDgreen, dark);
analogWrite(LEDgreen, brightnessgreen); // turquoise is on when blue
and green are on
analogWrite(LEDblue, brightness);
Delay(b);
analogWrite(LEDblue, dark); // turquoise is off when blue and green are
off
analogWrite(LEDgreen, dark);
analogWrite(LEDred, brightnessred); // purple is on when blue and red
are on
analogWrite(LEDblue, brightness);
Delay(b);
analogWrite(LEDblue, dark) //purple is off when blue and red are off
analogWrite(LEDred, dark);
```

```
}
```

Sketch 3- You can rewrite the above Sketch and still get the same result:

```
//RGB LED - fading between colors  
//pin connections  
int red = 9;  
int green = 10;  
int blue = 11;  
void setup(){  
  pinMode(red, OUTPUT);  
  pinMode(blue, OUTPUT);  
  pinMode(green, OUTPUT);  
}  
void loop(){  
  fader(red,green);  
  fader(green,blue);  
  fader(blue, red);  
}  
void fader(int color1, int color2){  
  for (int brightness=0;brightness<256;brightness++){  
    analogWrite(color1, 255-brightness);  
    analogWrite(color2, brightness);  
    delay(10);  
  }  
}
```

Motion Detectors

Goal: when motion is detected, a piezo speaker beeps

Requirements

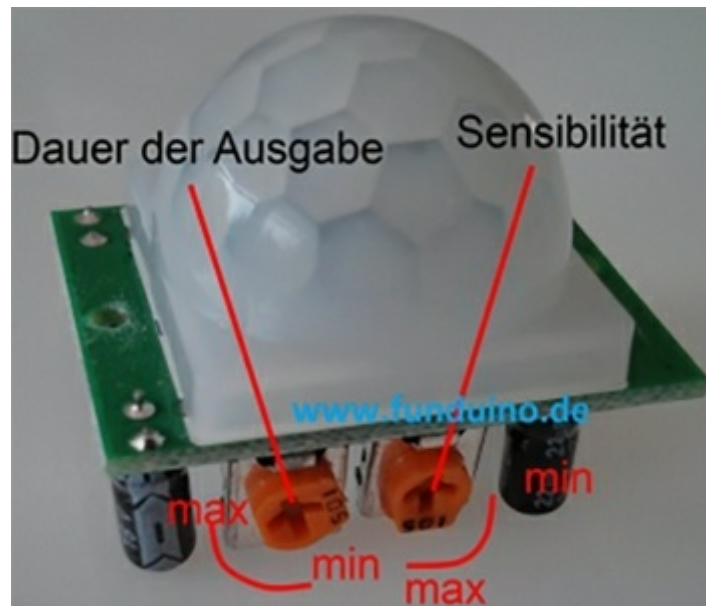
Motion detector, breadboard, Arduino, piezo speaker and cables

You should read the values of the voltage of a motion detector and utilize them in output

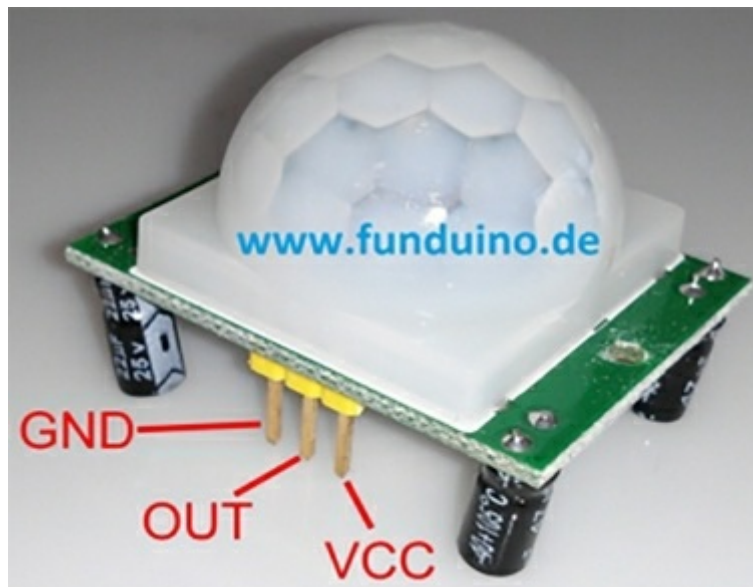
A motion detector is also referred to as a PIR sensor. It is a simple device. Since once movement has been detected, a 5V is placed on a pin. The data is

then read by the microcontroller, where it is still processed.

Knobs are used to easily adjust the motion detector's reach and the duration of the output signal as seen in the image below



On top of the motion detector is a plastic lens that can be easily removed. Underneath it, an IR-detector is visible, and the three contacts are indicated. GND(-), VCC(+) and OUT(Signal Output). This is indicated in the figure below.



At the bottom of the detector, you will also notice a jumper. This is responsible for switching between two nodes.

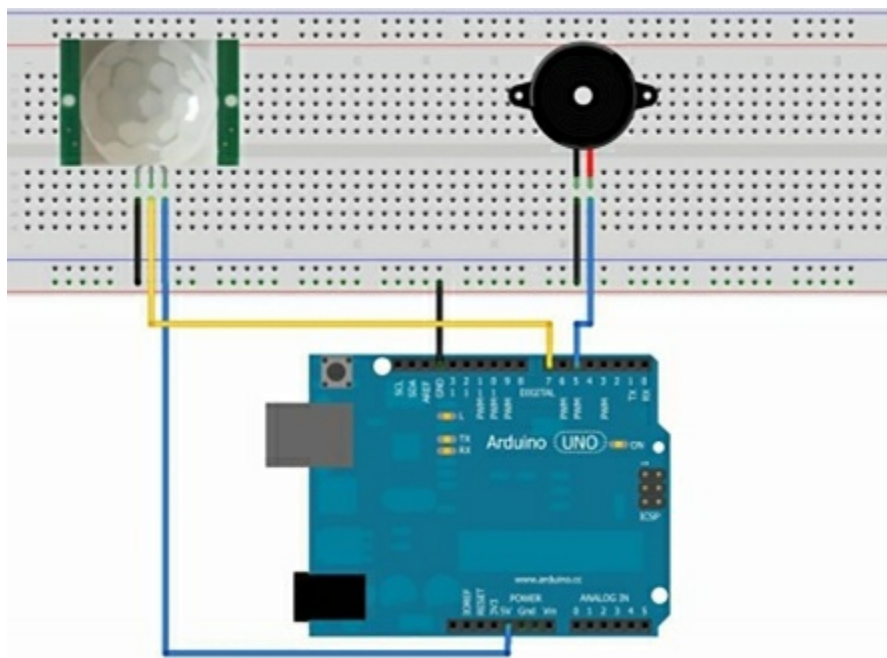
- 1) The output signal will be active especially when any movement is

detected. One mode is activated when the jumper is placed between the two contacts. This mode is used mostly for Arduino projects.

2) The other jumper that is outside, is maintained for a specific period of time, just after a movement has been recognized by the jumper. After that, the signal will go off, even while movement is still present. After some time, the movement will be activated.



Setup for A Motion Detector and a Speaker



Lets jump into the code

```

Int piezo=5; // piezo is assigned value 5
Int movement=7; // movement is assigned value 7
Int movementstatus=0;
Void setup()
{
pinMode(piezo, OUTPUT); // output is pin 5
pinMode(movement, INPUT); //input is pin 7
}
Void loop()
{movementstatus=digitalRead(movement); // pin 7's value is read and
the result saved //as movementstatus
If(movementstatus==HIGH) // if movement is identified( high voltage)
{digitalWrite(piezo, HIGH); //..the piezo beeps
Delay(5000); // 5 seconds delay time
digitalWrite(piezo, LOW); // piezo gets quiet} // end of if statement
Else
{// else open
digitalWrite(piezo, LOW); //the piezo speaker is off} //close else
statemnt
} // end of loop

```

Chapter 4

Graphics

Graphics on LCD

Just like we have seen how Arduino can decide the inputs of code, we can also draw pictures on an LCD using Arduino. First, we will begin by static graphics. Arduino will then decide on what it will draw, concerning the input. Before we get to the drawing bit, we are going to first explain the details that are necessary for sending graphics to the display.

Hex and Binary

Counting is normally done using Base ten. When you are using binary, each digit has to be multiplied by 10 like how we have indicated in the example below:

1

$10=1 \times 10$

$100=10 \times 10$

$1,000=10 \times 10 \times 10$

When you go for a number like 534. What this is:

$(5 \times 100) + (3 \times 10) + 4$

There are only 2 digital states in electronics, ON or OFF, HIGH & LOW. They are usually represented as either 0 or 1. so, in base 2 which is what we understand as binary, each value of the digit is 2 times as much as the number on the right. For instance

1

$10=1 \times 2$

$100=2 \times 2(4)$

$100=2 \times 2 \times 2(8)$

Therefore, when 1101 is identified by a computer, we all know now that is means

$(1 \times 8) + (1 \times 4) + (0 \times 2) + (1 \times 1)$ or 13

Binary numbers are then placed as inputs in the code when you start with a 0b.

Int value=0b1101

You will have to type a lot. Imagine 200

$$(1 \times 128) + (1 \times 64) + (0 \times 32) + (0 \times 16) + (1 \times 8) + (0 \times 4) + (0 \times 2) + (0 \times 1)$$

Int value=0b11001000

Computers are lazy; they don't like to put the extra effort in work. If they would be hardworking as we think, we couldn't program for them. Computers work by storing data in bytes. To make things easy, computers use hexadecimal, which use base 16, the hexadecimal. Now, in base 16, we are going to see how the calculations are done:

1

$$10 = 1 \times 16(16)$$
$$100 = 16 \times 16(256)$$
$$1000 = 16 \times 16 \times 16(4096)$$

T represents 200 in base 16; it would look like this

C8

$$(12 \times 16) + (8 \times 1)$$
$$192 + 8$$

200

You can start coding by putting 0x in the compiler, and it will know that you are placing a hexadecimal. In this particular instance, it's the best example where you can use both upper and lower case letters and they both mean the same thing.

Int value=0xC8

It is not different to the compiler when you write

Int value =200;

Int value=0b11001000;

Int value 0xC8

How to use Graphics

We are now done with the basics; let's get down to the real beef of this chapter. We want to display images on the LCD. Pixels that are located on the screen normally switch between two modes, an "off" and an "on". In the controller chip, it produces signals "ON" as "1" and "OFF" as "0". It is more like binary.

We shall be writing to the display, a column of 8 pixels at one go. For instance, if we want to the 8 pixels on, we can send a binary number 0b11111111 or 0xFF which is the hexadecimal representation. If we want to turn off all the 8, then we can send 0b00000000. To represent it on a screen, we can have the following.

Since data is sent a column at a time on the LCD, we are going to have the following

0b00000001, 0b00000010, 0b00000100, 0b00000100, 0b00010000, 0b00100000, 0b01000000 and 0b10000000. now representing it in hex, we are going to have the following in base 16.

0*01, 0*02,0*04,0*08 we will let you write the rest. 1 becomes the representation of each top pixel in each column, the next is 2 then 4 and so on.

To see this visually, open up your browser and log onto this website.

“www.introtoarduino.com/utls/pcd8544.html”

It will look like this:



The image height can be adjusted in a pack of 8 sets, while the width can be adjusted even to the width of the display screen. A pixel is easily turned off by right clicking while turning it back on requires you to click the square.

The target here is making a thermometer look-a-like graphic that will be displayed on the screen, like the figure below.



After drawing, select the output box and paste it into the Arduino program. If any difficulties come up, reload the web page. Let us write a sketch and see how it turns out. Let's do the entire program; then we shall dissect each part.

```
#include <PCD1111.h>
```

```
Const int f_pCLKin=5;
```

```
Const int f_pDINin=6;
```

```
Const int f_pDCin=7;
```

```
Const int f_pRin=8;
```

```
Const int f_pT=A0;
```

```
PCD8544 lcd(f_pDINin, f_pCLKin, f_pDCin, f_pRin);
```

```
Const int D_pHEIGHT=1;
```

```
Const int D_pWIDTH=5;
```

```
Const byte degreesBitmap[1 * 5] = {0x00,0x07,0x05,0x07,0x00};
```

```
Const ont T_pHEIGHT=2;
```

```
Const int T_pWIDTH=10;
```

```
Const      byte      ther_Bitmap[2      *      10]      =
{0x00,0x00,0x48,0xfe,0x01,0xfe,0x00,0x02,0x05,0x02,0x00,0x00,0x62,0x
```

```
Const int L_pHEIGHTlcd=6;
```

```
Const int L_pWIDTHlcd=84;
```

```

Void setup()
{lcd.init();
lcd.setCursor(10,0);
Lcd.print("Temperature:");
Lcd.setCursor((L_pWIDTHlcd,T_pWIDTH)/2,3);
lcd.drawbitmap(ther_Bitmap, T_pHEIGHT, T_pWIDTH);}
Void loop()
{float temperatureF=getTemperatureF();
// Celcius conversion
Float temperatureC=convertToC(temperatureF);
Lcd.setCursor(21,1);
Lcd.print(temperatureC);
Lcd.drawBitmap(degreesBitmap, D_pWIDTH, D_pHEIGHT);
Lcd.print("C");
Lcd.setCursor(21,2);
Lcd.print(temperatureF);
Lcd.drawBitmap(degreesBitmap, D_pWIDTH, D_pHEIGHT);
Lcd.print("F");
Delay(500);}
Float getTemperatureF()
{int reading=analogRead(f_pRin);
Float voltage=(reading * 5.0)/1024;
//using500mVoffset,convert10mvperdegree=degrees((voltage-
500mV)*100)
Return(voltage-0.5)*100);
}
Float convertToC(float temperatureF)
{return(temperatureF*9.0/5.0)+32.0;}

```

The first line

```
#include <PCD1111.h>
```

#include lets the machine know that the file in question should be picked, and before it compiles, it places the file content in the stead of the #include

statement. An #include normally has the angle brackets to denote that the library directory is being looked into. We are going to get straight into our pin definition, where we will create a variable

```
PCD8544 lcd(f_pDINin, f_pCLKin, f_pDCin, f_pRin);
```

A variable type PCD8544 is being defined and it tells the computer which pins the Arduino is connected to.

When the variable is being defined, we are telling what dc, pin din, clk and reset are attached to.

```
Const int D_pWIDTH=5;
```

```
Const int D_pHEIGHT=1;
```

```
Const byte degreesBitmap[1*5]={0x00,0x07,0x05,0x07,0x00};
```

The code above is used to define a small graphic that will be used as a degree symbol. It will be 5 pixels by 1 line high.

```
Const      byte      ther_Bitmap[2      *      10]      =  
{0x00,0x00,0x48,0xfe,0x01,0xfe,0x00,0x02,0x05,0x02,0x00,0x00,0x62,0x
```

This code simply shows the size of the graphic thermometer which is 10 pixels wide by 2 lines. So it means it is 16 pixels high, since each line is 8 pixels.

```
Const int L_pWIDTHlcd=84;
```

```
Const int L_pHEIGHTlcd=6;
```

Here, we are going to look at the LCD's height and width. We are going to calculate where the central place is.

```
{lcd.init();
```

```
lcd.setCursor(10,0);
```

```
Lcd.print("Temperature:");
```

lcd.init() this line initializes the lcd. After returning, we are able to use the LCD. We then go ahead and set the cursor to a position on the screen and we print a message "Temperature:".

```
Lcd.setCursor((L_pWIDTHlcd,T_pWIDTH)/2,3);
```

```
lcd.drawbitmap(ther_Bitmap, T_pWIDTH, T_pHEIGHT);}
```

Here, there is a trick that has been applied to center things. To place a graphic at the center, you can start but find the average position for the lcd width and

the graphic width. Test it a few times if you want to. This is used to know the starting column. The line starts at 3 to be on the display. Remember that this is actually the fourth line because lines are 0 based.

***drawBitmap()** is then called to draw the bitmap.*

```
lcd.drawbitmap(ther_Bitmap, T_pWIDTH, T_pHEIGHT);}
```

This now simply draws the degree bitmap.

How to make a chart

Below is a program that exhibits the temperature and a graph of historical temperature. We are going to code the sketch first and go through the parts.

```
#include <PCD8544.h>
```

```
const int f_DCPin = 7;
```

```
const int f_RPin = 8;
```

```
const int f_TPin = A0;
```

```
const int f_CLKPin = 5;
```

```
const int f_DINPin = 6;
```

```
PCD8544 lcd(f_CLKPin, f_DINPin, f_DCPin, f_RPin );
```

```
// A symbol (5x1) graphic
```

```
const int D_pWIDTH = 5;
```

```
const int D_pHEIGHT = 1;
```

```
const byte degreesBitmap[] = { 0x00, 0x07, 0x05, 0x07, 0x00 !!  
"" };
```

```
// A bitmap graphic (10x2) of a thermometer...
```

```
const int T_pWIDTH = 10;
```

```
const int T_pHEIGHT = 2;
```

```
const byte ther_Bitmap[] =  
{ 0x00, 0x00, 0x48, 0xfe, 0x01, 0xfe, 0x00, 0x02, 0x05, 0x02,  
0x00, 0x00, 0x62, 0xff, 0xfe, 0xff, 0x60, 0x00, 0x00, 0x00!!  
"" };
```

```

const int L_pHEIGHTlcd = 6;
const int L_pWIDTHlcd= 84;
const int G_pHEIGHT = 5;
const int MAX_TEMPIn = 100;
const int MIN_TEMPIn = 50;
void setup()
{
  lcd.init();
  lcd.setCursor(10,0);
  lcd.print("Temperature:");
  lcd.setCursor(0, L_pHEIGHTlcd - T_pHEIGHT);
  lcd.drawBitmap(ther_Bitmap, T_pWIDTH , !!
  "" T_pHEIGHT);
}
int xChart = L_pWIDTHlcd;
void loop()
{
  float temperatureF = getTemperatureF();
  float temperatureC = convertToC(temperatureF);
  lcd.setCursor(21,1);lcd.print(temperatureC);lcd.drawBitmap(degreesBit
  D_pWIDTH , D_pHEIGHT);
  lcd.print("C");if(xChart >= L_pWIDTHlcd){xChart = T_pWIDTH  +
  2;}
  lcd.setCursor(xChart, 2);

          int          dataHeight          =
  map(temperatureC,MIN_TEMPIn,MAX_TEMPIn,0,!!""G_pHEIGHT*8);
  drawColumn(dataHeight);
  drawColumn(0);
  xChart++;
  delay(500);

```



```

}
float getTemperatureF()
{
    int      reading=analogRead(kPin_Temp);float      voltage=
    (reading*5.0)/1024;return (voltage-0.5)* 100;
}
float convertToC(float temperatureF)
{
return(temperatureF*9.0/5.0)+32.0;}
constbyte                                     dataBitmap[]=
{0x00,0x80,0xC0,0xE0,0xF0,0xF8,0xFC,0xFE};
void drawColumn(unsigned int value)
{ byte graphBitmap[G_pHEIGHT ];int i;
if(value> (G_pHEIGHT*8)){value=G_pHEIGHT*8;}
for(i=0;i <G_pHEIGHT;i++){graphBitmap[i]=0x00;}i=0;
while(value>=8){graphBitmap[G_pHEIGHT-1-i]=0xFF;value-
=8;i++;}
if(i!=G_pHEIGHT                                     ){graphBitmap[G_pHEIGHT-1-
i]=dataBitmap[value];}
    lcd.drawBitmap(graphBitmap,1,G_pHEIGHT);}

```

We have seen most parts of this code, we shall concentrate on the parts that are new.

```
const int G_pHEIGHT = 5;
```

The graph height is defined to be on 5 lines, this means that it will be 40 pixels.

```
const int MIN_TEMPin=50;
```

```
const int MAX_TEMPin=100;
```

This shows the top and bottom values of a graph. These are random values assigned to constants.

```
    lcd.setCursor(0, L_pHEIGHTlcd - T_pHEIGHT
    );
```

```
lcd.drawBitmap(ther_Bitmap, T_pWIDTH, !!
""T_pHEIGHT;
```

We are subtracting the thermometer's height from the LCD height. This places the thermometer on the lower left corner.

```
int xChart = L_pWIDTHlcd;
```

This new variable being defined above is used to track our position on the chart. The starting value will be **L_pWIDTHlcd**.

```
if(xChart >= L_pWIDTHlcd){
xChart = T_pWIDTH + 2;
}
```

```
lcd.setCursor(xChart,2);
```

If the **xChart** is located at the peripheral of the screen, then it is reset 3 after the width of the thermometer. The 2 provides space before the chart and after the graphic. The initial value is noticeable to be the LCD-WIDTH, though it will be reset the first time. This is done if we change the spacing. The cursor is then set to start the next thing the LCD sees will be at the specific location set.

```
int
dataHeight=map(temperatureF,MIN_TEMPin,MAX_TEMPin,0,!!""G_p.
drawColumn(dataHeight);drawColumn(0);//marker to see current chart
position
xChart++;
```

We calculate the columns height by using the **map()** function. We then draw a column by calling the **drawColumn()** function. To draw an empty column, we call the **drawColumn()** with zero in order to draw an empty column after the first column was drawn. After seeing where we are, we then increment **xChart** for us to be at the next spot on the graph when we get to it.

The **drawColumn()** is the new stuff. It adds segments to 8 and calculates the last segment's size.

```
if(value > (G_pHEIGHT * 8)){
value = G_pHEIGHT * 8;
}
```

We always have to make sure that our value does not exceed what it should

be. If it happens, writing past the array size will be really bad and hard to solve. Therefore, this code checks if we are within the boundary.

```
for(i=0;i <G_pHEIGHT;i++){graphBitmap[i]=0x00;  
In graphicBitmap, the value of everything will be set.  
i = 0;while(value>=8){graphBitmap[G_pHEIGHT-1-i]=0xFF;value -  
=8;i++;}  
if(i != G_pHEIGHT){  
graphBitmap[G_pHEIGHT - 1 - i] = dataBitmap[value];  
}  
lcd.drawBitmap(graphBitmap, 1, G_pHEIGHT);  
}
```

The last part is abit tricky. Let's go over another example to understand what we mean.

Let's say the value at “**while**(value >= 8)” is value==20

```
If value>=8- yes it is,  
graphBitmap[G_pHEIGHT - 1 - i] = 0xFF; // all is turned on  
value -= 8; Now the value is 12 here (20-8)  
i++; (i is now 1)
```

We go back to “**while**(value >= 8)” is 12 greater than 8. Yes it is.

```
graphBitmap[G_pHEIGHT- 1 - i] = 0xFF;  
Value-=8. Now value is 4  
i++ (i is now 2)  
Here, 8 is greater than 4  
if(i != G_pHEIGHT){  
graphBitmap[G_pHEIGHT - 1 - i] = dataBitmap[value];
```

Since i is 2, the graphBitmap[2] will be set to a value of dataBitmap[4] that is 0xF0

So, out graphicBitmap will look like this at the end

0x00, 0x00, 0xF0, 0xFF,0xFF

Assignment

1.You need to change the program and change the circuit and connect them

with a button, and a graph to start over when the button is pressed.

2. To take it to the next level, the program and the circuit need to be changed so that a button is used to change from Fahrenheit to Celsius and vice versa.

3.Challenge has an auto scale on the graph. This means that the graph at some point has a minimum temperature both at the bottom part and at the top part, where it has the maximum temperature observed. The LCD should display the graph.

Chapter 5

References in Arduino

This reference describes every part of Arduino language. We have not defined the appendix here though. If you want to get more details other than what we will describe in this section, kindly refer it online.

We are going to divide this section into three main sections. These are structures, variables & constants and functions

Control Statements

While loops- continuously loop until the tested variable is changed, otherwise it will keep on going.

Do....while - not done in this book, but it is like a while statement, except with one execution and then the condition evaluated.

Break- not done in this book. This code goes out of the current block when it is encountered

If Statement- the block of statement gets executed if the expression is true, if not, the statements are skipped.

If...else statement- this is the optional else statement that follows the *if statement* when the expression is false

Switch case- Allows the programmers to determine various codes that will be performed in different conditions

Code writing Format

{ } (brackets) - it defines a block of code

// (double dashes) -single comment line

/* */ - multiple comment line

>> - shift right operator

; (**semicolon**) - it ends a statement

Variables

HIGH or LOW - used to set the state of the pin

INPUT or OUTPUT - used with pinMode

True/false - false is equal to 0, true is not false

Data Types

Void- it declares ONLY functions- here, null or the output. The function will return no information if it is indicated as void

Boolean- occupies one byte of the memory and holds the true or false values

Byte- an 8-bit unsigned number from 0-255 is stored in a byte

Int- in terms of number storage the primary data type is the integers, a 2-byte 16-bit value is stored in the int

Word - a 16-bit unsigned number is stored by a word on the UNO and other ATMEGA based boards

Conversion

Char() char type

Byte() byte type

Float() float type

Long() long type

Word() word type

Int() int type

Functions

Digital Input and Output

Name Explanation

analogread() function- enables the programmer to read the applied voltage on the pins

pinMode() - it describes INPUT or OUTPUT

digitalWrite()- used to write the value (HIGH or LOW) to a digital pin. The digitalWrite() enables HIGH and disables LOW if the pin is configured as INPUT. With OUTPUT, voltage will be set as 5V or 3V for HIGH and 0V for LOW

digitalRead() -reads a pin value

Time

Name Explanation

Mills() - Returns the milliseconds that Arduino has been on

Micros() - returns the microseconds

Delay() - pauses the program for the amount of time

Math

Min()- If there are 2 values, the minimum is displayed

Abs()- returns the absolute value

Max()- returns the maximum of 2 values

Pow()- To the 'power of' calculation

Sqrt()- square root is displayed

Trigonometry

Tan()- tangent of an angle calculation is done

Cos()- the cosine of an angle calculation is done

Sin()- the sine of an angle calculation is done

Random Numbers

Random()- generate a pseudo-random number

randomSeed()- initialize the pseudo-random number generator

Bits and Bytes

Bit()- the value of the specified bit

bitClear() - set a bit of a variable to 0

bitSet() - set a bit of a variable to 1

LowByte()- lowest of a variable

highByte() - highest byte of a variable

PCD8544 (LCD Controller) Library

Init() - It is called first, you can do anything else with the LCD.

Clear()- Clearing of the screen happens with this

drawBitmap()- this takes the bitmap followed by the number of columns(1-84) and the lines(1-6) of data being passed to it

setInverse() - you can call this function with true or false

Tips for Arduino Programming

We are going to look at a few random tips that can help programmers. There are various tips that you can look into on the internet, such as:

- How one can provide space for RAM, by storing text in the memory of the program. This Memory trick will make sure that your programs are readable.
- How to save RAM using text strings
- Each time the Arduino is reset, the RAMs content is normally reset. It can also happen when the power is lost. Power loss and reset are used retain the program's memory. When the program begins, the program you have written uses the normal strings which are copied from the program memory and directed into the RAM.
- Arduino boards normally have on average, 2kb RAM. Therefore, if your program uses lots of strings, you will definitely run out of RAM.
- Your program normally stops because of the delay function. When you look at the blink program that is basic, most of the time, the program will be doing nothing.
- In the Arduino microprocessor, it has a timer called the watchdog timer. This is a dead man switch. If your program does not inform Arduino that things are going according to plan, then it can possibly blow the Arduino. Now, to make the most out of the timer, you will need to put it in the library, you are going to make the watchdog active, and finally, inform the watchdog that the system is functioning as usual.

Conclusion

Thank for making it through to the end of this book I hope that after reading the book it will aid you in installing a library in your device of choice, make the most out of your Arduino module and also get to install on your own the new ArduinoISP to your board.

What you need to do next is to learn about coding and various ways to create new logic modules and further what you have already learned from this book and pass it on to someone else.

Coding can be of great use to you, maybe in the close future. The amount of coding language that is there in the universe is great and being part of the crew (like being part of a dance crew, hehe) you will protect your device better and get to teach someone else. It is something considered for geeks and the nerds, but t currently, being a nerd or geek is the new thing! brains and beauty my dear reader, always brain and beauty.

Maybe the next time someone has an issue with their chip or library, you can offer insight on what went wrong and how to rectify it. And retain the bragging rights of having learned it on your own!



Finally, if you enjoyed this book, and found it in any way helpful and resourceful, a review on Amazon is always appreciated!

ARDUINO

***Best Practices to Learn and Execute Arduino
Programming***

DANIEL JONES

Introduction

This book contains proven steps and strategies on how to use Arduino, both the hardware and the software.

You are going to learn some of the best practices that you can put into place in order to use Arduino effectively. It is not going to be easy, but that is alright because with practice you are going to be able to grasp the concepts that you are going to want to know when it comes to working with Arduino.

Chapter 1

Learning What Arduino Is

Arduino is a computer company that is going to work with both the software and the hardware that a computer is going to use as it runs. The community that uses Arduino is going to be those who design and manufacture microcontrollers and kits that go with these microcontrollers which will be located on a single board.

Arduino boards will be used when it comes to building digital devices and other interactive objects that are going to have the ability to control things in the physical world around you. These products will use an open-sourced hardware and software that will be licensed under the LGPL – GNU Lesser General Public License – along with the GPL – GNU General Public License – which is going to make it to where the boards can be manufactured and distributed to the public. You are going to be able to purchase Arduino boards pre-assembled or in a kit that will enable you to build the kit yourself.

The board's design is going to be based on the sort of controllers and microprocessors that it contains. The board is going to be equipped with digital and analog input and output pins that will allow the board to interact with expansion boards and other circuits. One feature that the board contains is that it can communicate with various interfaces. This means that you can use USBs on some of the Arduino models that are going to allow for programs to be loaded on the board off of a personal computer.

The microcontrollers are going to be programmed to use C programming language and even C + +. The board is also going to use a compiler toolchain that will be more traditional so that it can provide an integrated development environment – IDE – that will be based on the programming language that it works with.

This project was started in 2003 in Italy as a program for the students that were attending the Interaction Design Institute. The reason that this project was started was to try and provide a cost-effective way for a novice and a professional to create a device that would allow them to interact with the environment around them and through the use of sensors and actuators.

Some of the most common examples of how devices interact would be motion detectors and thermostats.

The Arduino name came from a bar that you will be able to find in Ivrea, Italy in which the founders all met so that they could discuss their ideas. This bar was named after the March of Ivrea and the King of Italy from 1002 to 1014.



Chapter 2

Hardware, Software, and Applications

Since Arduino is an open-sourced piece of hardware, there will be reference designs to the hardware that will be sent out through the Creative Commons Attribution-ShareAlike License which ultimately means that they will be available on the Arduino website. The layout and the production files will be available depending on the version that you are currently working with.

Whenever the source code was released to the public thanks to the GNU General Public License, it made it to where everyone had access to the code. But, the official Bill of Materials that are tied to the boards has not and most likely will not be published by anyone who works for Arduino.

Even though you can find the designs for the software and hardware for free, the name Arduino is exclusive to the boards to make sure that it cannot be used for other boards without the company's expressed permission. In fact, the documentation that is place for the Arduino name states that the project is open to working with others in order to incorporate them into the official product.

Some products are going to be compatible with Arduino boards that were released commercially in order for them to avoid having other products on the market with the duino suffix. A lot of these boards are going to have an Atmel 8-bit AVR microcontroller. This consists of various pins, features, and flash memory. In the event that you are using a 32-bit board, you will be using an Atmel SAM3X8E that was released in 2012. All of the boards will have a single or double row of pins that will facilitate the connections needed for programs and circuits to work correctly.

The boards that you are able to purchase are:

1. Arduino RS232
2. Arduino Diecimila
3. Arduino Duemilanove
4. Arduino Uno R2
5. Arduino Uno SMD R3
6. Arduino Leonardo

7. Arduino Pro
8. Arduino Mega
9. Arduino Nano
10. Arduino LilyPad 00
11. Arduino Robot
12. Arduino Esplora
13. Arduino Ethernet
14. Arduino Yun
15. Arduino Due

Shields

Arduino and devices that are compatible with the Arduino boards are going to use circuit expansion boards that are going to be known as shields that will be plugged into the pinheads. Shields will provide controls that are needed for three-dimensional printing and other applications like GPS or liquid crystal displays. You will even have the option of making your own shield.

Software

When a program is written for Arduino, it is going to be written in any programming language that the programmer wants to write it in as long as it produces a binary machine code for the target processor. The Atmel will create a development environment that is going to work with the microcontrollers.

The integrated development environment is going to be provided by the Arduino project so that the applications that are cross-platform will be written in Java. This was started with the IDE for languages that do the processing and writing. It also had an editor that is going to enable you to cut text and paste it while also being able to search for it and replace it. You will also be able to use syntax highlighting, automatic indentation, and brace matching that is going to be provided by one click mechanisms for programs to be compiled and uploaded to the board. The message area is going to hold text consoles, toolbars, and an assortment of other buttons that are going to help when it comes to improving how the board functions.

The Arduino has two primary functions:

1. `Loop()`: this will be completed once you finish the `setup` function. The `loop` function is going to be executed by the main program. This will be what controls the boards until it is turned off or restarted.
2. `Setup()`: once the sketch has been started by this function the function is going to be called. As the user, you will use this function to initialize constants and other libraries that will be needed so that you can use a sketch.

Applications

Here are several of the applications that you are going to be able to use with an Arduino board.

1. A sensor system that will be used to detect bovine milk adulteration
2. Xoscillo – this is an open-source oscilloscope.
3. A low-cost data glove that will work with virtual reality applications
4. Arduinome – this is a device that is going to mimic the Monome through a MIDI controller.
5. An automatic titration system that is going to work with Arduino as well as a stepper motor.
6. OBDuino – a computer that will be used for on-board diagnostics that you do to most of the modern-day cars.
7. Testing the quality of water
8. Ardupilot – this software is for a drone that works with Arduino
9. The Arduino phone which is a phone that you can build yourself.
10. Gameduino – this shield will allow for you to recreate the 2D retro video games

There are plenty of other applications that you can get to work with Arduino, and you are going to have the option of building your own applications if that is what you are wanting to do. You are not going to have to sell them with the Arduino name if you do not want to. You can keep them and use them for yourself. But, keep in mind, if you are going to work for the Arduino

company, you will be getting your applications out to more people than you are ever going to be able to imagine.

Chapter 3

The Data Types You Will Find in Arduino

Any data types that you find in Arduino will be written in the C programming language and are going to refer to an extensive system that you will use to declare functions and constants. Constant types are going to determine how much space a constant is going to take up on the storage as well as how the bit pattern will be interpreted.

The data types that you will use with Arduino are:

1. String-char array
2. String- object
3. Word
4. Unsigned int
5. Int
6. Byte
7. Unsigned char
8. Char
9. Boolean
10. Invalid
11. Long
12. Unsigned long
13. Short
14. Float
15. Double
16. Array

Invalid

The invalid keyword is going to be used whenever you are using function declarations. Just like the word indicates, the function is going to be returned with no information when it is called upon.

Example:

```
Invalid loop () {  
# this is where all of the rest of your code is going to be
```

}

Boolean

The Boolean will have two values. The value can either be true or it can be false. For each Boolean constant that is located, at least a single byte on the memory is going to be saved.

Example:

Boolean var = false; # your constant declaration will be here and then initialized as false

Boolean state = state; # your constant declaration will be here and then initialized as true.

Char

The char data type is going to take up a byte of memory for every character type that is stored. The character literals will be using single quotes while a double character is going to use double quotes. While the numbers will store as numbers, you are going to notice that the encoding is going to be particular when it comes to the ASCII chart. In other words, this means that you are going to be able to do mathematical equations that work with characters that you can find in the ASCII chart.

Example:

Char chr_z = 'z' # the constant will be declared with the char type, and then it will be initialized with the z character.

Char chr_M = 77 # the constant is declared here and then initialized with the character 77 because 77 is the value that is assigned to M on the ASCII chart.

Unsigned char

You will discover that the data type will be unsigned and it is going to also occupy a single byte in the program's memory. Unsigned char data types will fall between zero and two hundred and fifty-five.

Example:

Unsigned char chr_m = 24 #the constant is going to be declared with this unsigned char before initializing it with your m character

Byte

A byte will be a piece of the memory that is going to store up to eight bits of unsigned numbers that are going to fall between zero and two hundred and fifty-five.

Example:

Byte l = 4 # the constant is going to be declared for the byte type and then initialize it with the number 4.

Int

The integer data type will be the main data type that is used for numbers that will be stored in the memory. Integers will be stored in two-byte values and will go all the way up to sixteen-bit values. The range for an integer will be -32,768 to 32,767 which ultimately means that the minimum value is going to end up being -2^{15} while the maximum value is $(2^{15} - 1)$.

The size of the integers is going to vary based on the type of board that is being used. The Arduino Due will allow for integers to be stored on four-byte values. The range is going to be -2,147,483,648 to 2,147,483,647. This means that the minimum value is going to be -2^{31} while the maximum is $(2^{31} - 1)$.

Example:

Int counter = 43 # the constant of integer will be declared and then initialized with the number 43.

Unsigned int

An unsigned integer is going to work the same way that a regular integer is going to work, but they are going to be stored in two-byte values. Instead of storing negative numbers, it is going to focus on storing positive values. Ranges will be 0 to 65,535 $(2^{16} - 1)$. The Due will store four bytes that will range for 0 to 4,294,967,295 $(2^{32} - 1)$.

Example:

Unsigned int counter = 109 # the constant of the unsigned int is going to be declared and then initialized with 109.

Word

When you used an ATMEGA board, the word is going to be stored in a

sixteen-bit number that will end up being unsigned. Whenever you use the Due or the Zero, it is going to be a thirty-two bit of unsigned numbers.

Example:

Word a = 293 #the declaration for the word type is going to be here before it is initialized with 293.

Long

Long constants are going to constants that will be extended in size so that they can be stored in the number storage. They are going to be saved in four bytes and fall in the range of -2, 147, 483, 648 to 2, 147, 483, 647.

Example:

Long = 103843 # the constant is going to be declared a long constant type and will be initialized with 103843.

Unsigned long

The unsigned long constants are going to be the constants that are extended in size for the number storage that will store them in thirty-two bits. Unlike a standard long, the ones that are unsigned are not going to store negative numbers, but only positive ones. They will be stored in the range of 0 to 4, 294, 967, 295 ($2^{32} - 1$).

Example:

Unsigned long = 1093773 #this constant is going to be declared an unsigned long before initializing the number 1093773.

Short

The short is a data type that is going to hold up to sixteen bits. The ATmega and ARM-based boards will use sixteen-bit value storage. The range is going to be -32, 768, to 32, 767. The minimum value will end up being -2^{15} while the maximum is $(2^{15} - 1)$.

Example:

Short var = 32 #the short constant will be declared before initializing with the value 32.

Float

Floating point number data types are going to be decimal point numbers. These numbers are going to be used in order to approximate the analog along with the constant values since they are going to hold a greater resolution than an integer.

Floating point numbers will have the ability to be as big as 3.4028235×10^{38} and as low as $3.4028235 \times 10^{-38}$. Floats are going to be assigned to store information on four bytes of memory space.

Example:

Float number = 3.234 #the constant of a float is going to be declared before being initialized with the number 3.234

Double

If you are using an ATMEGA board, the double precision float numbers are going to take up to four bytes. These double implementations are going to use the same float; however, they are not going to gain precision. Whenever you are using the Arduino Due, the double will have an eight-byte precision.

Example:

Double number = 43.234 # the constant is going to be declared as a double before being initialized as a 43.234.

Chapter 4

Constants

– Local and Global

Constant scope

Whenever you are working with the C programming language, the constants are going to have a scope. This scope is going to be the area that the Arduino program is going to have up to three places where this constant will need to be declared. The constants that have to be declared are:

1. Inside of the definition for the functions parameter which is going to be known as the formal parameters.
2. Outside of the function listed with global constants
3. Inside of the block that holds the local constants

Local constants

These constants will be listed inside of a function and are going to be declared the local constants. They are only going to be declarations that can be found inside of the function or code block. These constants are not going to work outside of a function.

Example:

```
invalid setup () {  
    }  
invalid loop () {  
    int e, a;  
    int r; # local constant declared here  
    e = 0;  
    a = 0; # this is where the initialization is going to actually start  
    r = 39;  
}
```

Global constants

A global constant is going to be a constant that is defined outside of the function and is going to usually be located at the utmost part of the program. Global constants are going to be able to hold values that are assigned for the life of the program.

Example:

Float A = 0; the global constant has been declared here

}

```
Int e, a;
```

$$E = 0;$$
 $R = 39;$

}



Chapter 5

Operators

Operators are symbols that will inform the program compiler which mathematical or logical function has to be executed inside of the expression that you have placed in the program. Because Arduino is written in C programming language, it is good to know that this language is rich in operators that are built into the program.

Some of the operators you are going to be working with are.

1. Boolean operators
2. Compound operators
3. Comparison operators
4. Arithmetic operators
5. Bitwise operators

Arithmetic operators

As you know, arithmetic is going to be math, therefore these operators are going to help you with the math symbols that you are going to need to use in Arduino so that you can tell the program what you are trying to do.

1. *Modulus (%)*: Modulus is going to give you the remainder once division has been done example: $z \% b$
2. *Division (/)*: The top number (numerator) is going to be divided by the bottom number (denominator). Example z / b
3. *Subtraction (-)*: The second number is going to be subtracted from the first. Example $z - b$
4. *Assignment (=)*: This operator is going to take the value that is located on the right and declare that it is equal to the constant that can be found on the opposite side. Example $z = b$
5. *Addition (+)*: The operands will be added together to get the sum of the two numbers. Example $z + b$.
6. *Multiplication (*)*: Both numbers are going to be multiplied together. Example $z * b$

Comparison operators

These operators are going to compare both of the operands so that you can see how they stack up next to each other. Not every operand is going to be equal to each other, and there are going to be various other comparison operators that you are going to be able to use in order to compare two constants.

1. *Greater than or equal to (\geq):* The number on the left will be checked to see if it is either equal to or more than the value found on the right. Should this be accurate, the condition will be marked as true example $Z \geq B$
2. *Equal to ($=$):* The two numbers will be checked to see if they are equal. If they are, then the condition is going to be true example: $Z == B$
3. *Less than or equal to (\leq):* The value that is on the left will be checked to see if it is less than or equal to the value that is on the right. Should it be, then the condition will be true. Example: $Z \leq B$
4. *Not equal to (\neq):* The two values are going to be checked to see if they are equal. If they are not then the condition is going to be considered true. Example: $Z \neq B$
5. *Greater than ($>$):* The value on the left will be checked to see if it is more than the one on the right. If it is, then the declaration evaluates to be true. Example: $Z > B$
6. *Less than ($<$):* The value on the left will be checked to make sure it is less than the other. If it is, then the declaration will evaluate as true. Example: $Z < B$

Boolean operators

These are going to be the operators that are going to consist of the not and or operators. You can think of these operators as the misfit operators that do not really fit anywhere else.

1. *Not (!):* This is the logical NOT operator. The logical state will be reversed. If the position is determined to be true, then the logical NOT will cause the value to read as the opposite. Example: $!(Z \&\& B)$
2. *And (&&):* The logical AND operator will check to see if the operands are non-zero. If this is found to be true, the condition is going to be true example: $(Z \&\& B)$
3. *Or (||):* The logical OR operator will see if there are any non-zero

operands. If there is one or more, then the condition will be considered as true.

Bitwise operators

These operators are going to tell you where you should move a constant and if you make that move, what is going to happen to the constant.

1. *Move left (<<)*: There is going to be a movement of the values on the left and they are going to be moved by the number of bits that the operand tells them to move. Example $Z \ll 3$
2. *And (&)*: Binary AND will copy the bit to the outcome that is found in both operands if any. Example $Z \& B$
3. *Not (~)*: Binary one's complement operator will flip the bits. ($\sim Z$)
4. *Or (|)*: Binary OR is going to copy the bit that exists in one or both operators. Example $(Z | B)$
5. *XOr (^)*: The binary XOR will copy the bit only if it is in one of the operands, it cannot be found in both. Example $(Z \wedge B)$
6. *Shift right (>>)*: The left operator will be transferred to the right as many times as the number on the right is going to indicate. Example $Z \gg 4$

Compound operators

The compound operators are going to be the operators that are going to combine two of the operators that were just mentioned above. Not every operator can be combined together because they will end up giving you an error message. But, there are a few that can be tied together and you will see them listed below.

1. *Compound division (/=)*: Divide AND assign operator. The left operand will be divided by the right and then assigned to the left operand. Example $Z /= A$ which is equivalent to $Z = Z / B$
2. *Increment (++)*: The integer will be increased by one value each time. Example $Z ++$
3. *Compound multiplication (*=)*: The multiple AND assignment operator will multiply the right number with the left one and then assign it to the left operand. Example $Z *= B$ which is equivalent to $Z = Z * B$
4. *Decrement (--)*: The operand will decrease the integer by one each

time, example: Z--

5. Compound subtraction (-=): The Subtract AND assignment operator is going to subtract the right number from the left and then assign it to the left. Example Z -= B

6. *Compound addition* (+=): The right operand will be added to the left and then assigned to the left operand example Z += B

7. *Compound bitwise and* (&=): Bitwise AND assignment example Z &= 4

8. *Compound modulo* (%=): The modulus will be taken with the two operands and then assigning it to the left operand example Z % = A

9. *Compound bitwise or* (|=): bitwise inclusive OR and assignment example Z |= 3

Chapter 6

Control Declarations

A control declaration will be structured to make decisions, but it is only going to be able to do so if you specify at least one condition, if not more, that will give the declaration the ability to be evaluated by the program. This is going to be the same as any declaration that is placed into the Arduino program, they are going to be carried out until the condition is found to be true. Any other declaration that has been placed into the program is going to be executed until the declaration is found to be false.

Control declaration

'If' declaration: the if expression is going to be located inside of a set of parentheses or in the group declarations. Should the expression be evaluated as true, then it is going to be carried out. If it is found to be false, then it is going to be skipped over by the program.

Syntax:

Form A:

if (condition)

single line of code goes here;

Form B:

if (condition) {

multiple lines of code;

}

'If... else' declaration: you are going to be placing an else declaration into the equation that will be executed should the declaration be found as false.

Syntax:

If (condition) {

multiple lines of code;

}

else {

multiple lines of code;

}

Else if declarations: this is a declaration that is going to be similar to the if declaration and is going to be able to be followed by the else declaration. But,

it can also be followed by an else if declaration that is going to be used whenever you are testing multiple scenarios by using one 'if...else if' declaration.

Syntax:

```
if (condition1) {  
multiple lines of code;  
}  
else if (condition2) {  
multiple lines of code;  
}  
else {  
multiple lines of code;  
}
```

Switch case declarations: this declaration is going to be similar to an if declaration. But, it is going to be the declaration that controls the flow of the program that will also allow the programmer to specify various codes that have to be carried out inside of the conditions that you have set forth.

Syntax:

```
switch (constantGoesHere) {  
case one:  
// code internally  
break;  
}  
case two: {  
// code internally  
break;  
}  
default: {  
// code internally  
break;  
}
```

Conditional operator: A provisional operator that is going to consist only of the ternary operator that can only be found in the c programming language.

Syntax

Express 1? expression 2: expression 3

Control Statements

Chapter 7

Loops in Arduino

Most programming languages will give you a controlled structure that is going to allow for more complex execution paths to be carried out. Loops and loop declarations will be declarations that are going to allow for a group of declarations to be carried out multiple times and will follow the general set up that you see for loops in most programming languages.

‘while loop’: these loops is going to always run the expression that is inside of the parentheses until the condition becomes false. Something will have to change, like the tested constant, or the loop is ever going to end.

Syntax:

```
While (condition) {  
  multiple lines of code;  
}
```

“Do while” loop: the do while loop is going to operate much like a while loop. The while loop is going to constantly run until the condition that is being tested at the top of the loop is met until the time that the body of the loop will be executed.

Syntax:

```
do {  
  multiple lines of code;  
}  
while (condition);
```

‘for’ loop: A for loop will take the declaration and carry it out a set number of times that has been determined beforehand. These controlled expressions are going to be initialized, tested, and even manipulated inside of the parentheses for the loop that is being worked with.

Syntax:

```
For (initializingConstant; endCondition; increment) {  
  // declaration block  
}
```

Example

```
for (number = 3; number <= 32; number++) {
```



```
// your group declarations are going to be carried out 29 times.  
}
```

‘nested’ loop: With C programming, you are going to be able to take one loop and place it inside of another loop which is going to be known as nesting. When you nest loops the first loop is going to be executed before the second one is executed. This is done just so that you do not have to complete extra steps.

Syntax:

```
for (initializerConstant; endCondition; increment or decrement) {  
    // internal code  
    for (initializerConstant; endCondition; increment or decrement) {  
        // internal code  
    }  
}
```

Example:

```
For (start = 3; start <= 6; increment++) {  
    // the declarations here are going to be executed 3 times  
    For (s = 9; s <= 15; s++) {  
        //these comments will be carried out 6 times  
    }  
}
```

Infinite loop: there is no condition in this loop which means that it is not going to have anything to stop it from running which causes it to run forever.

Syntax:

```
for (;;) {  
    // internal code  
}  
OR  
while (true) {  
    // internal code  
}
```

Chapter 8

Arduino Functions

The Arduino functions are going to allow you to structure various segments of code in an effort to perform an individual task more efficiently. A lot of the time, you will have to create a function so that you can ensure that the same action is carried out multiple times in the program without you having to constantly put the expression in the program to be executed again.

Some of the advantages for standardizing code fragments when it comes to your functions are:

1. The code will be easier for you to reuse over and over again even if it is in a different program. This makes it to where it is modular and by using functions more often than not, your code is going to become more readable.
2. The functions are going to help the programmer stay organized which in turn is going to help with the conceptualization of the program.
3. Functions are going to make your sketch look smaller and more compact. This will happen because the blocks of code are being reused.
4. Functions can code a single action which will allow the function to be evaluated and debugged at the same time.
5. Errors are going to be reduced even if the code has to be modified.

Arduino's sketch will have two functions that are going to have to be included in the program. These functions were discussed when you saw the setup and the loop. Any other function will be created outside of the brackets and will work for one of these primary functions.

Syntax:

```
Return type function name e (argument1, argument 2, ...)  
    {  
    Declarations  
    }
```

Function declaration

Functions will be declared outside of another function and it is going to be placed before or after the loop function.

A function can be declared in one of two ways. The first way is to write the function out known as the function prototype so that it can be placed before the loop function. The loop function is going to consist of the following parts.

1. Function name
2. Function argument type – you will not be required to write the argument name out
3. Function return type

Remember: the prototype will have to be followed by a semicolon!

Example:

```
Int add_func(int a, int e) //the function will be declared here {  
Int u = 0;  
U = a + e;  
Return u; //the value will be returned here.  
Invalid setup () {  
Declaration block  
}  
Invalid loop () {  
Int outcome = 0;  
Outcome = add_func (2, 4); //the function call  
}
```

The second method that you are going to have to choose from is to define the function. If you are wanting to declare the function, you are going to have to declare the loops. Make sure that you have all of these parts for this particular method.

1. Function name
2. Function body – the declarations will be listed inside of the function so that they can be executed whenever the function has been called.
3. Function argument type – you are going to have to name your argument here
4. *Function return type*

Example:

```
Int add_func (int, int); //your function prototype
```

```
Invalid setup () {  
Group of declarations  
}  
Invalid loop ()}  
Int outcome = 0;  
Outcome = add_func (3, 5) // the function call  
}  
Int add_func (int a, int w) //the declaration of the function  
Int o = 0;  
O = a + w;  
Return to //the value that needs to be answered.  
}
```

Chapter 9

Arduino Strings

Any text that you place in Arduino is going to automatically be placed in a string. These strings are going to be displayed on your LCD monitor on the serial monitor window that IDE uses. Strings can be useful whenever it comes to the storing of input from a user. One instance of this would be if a user was to use a keypad that was connected to the Arduino board and input answers to the questions that are displayed on the screen.

There are two diverse types of strings that you are going to use in Arduino.

1. The character arrays – these are going to be the same as any other string that is used in the C programming language.
2. The Arduino string – this string is going to allow the Arduino user to create a string object while using the sketch.

String character arrays

The first-string type is going to be used with a series of characters that are going to be known as the char data type. The arrays will be a consecutive series of the same constant types that will be stored in the program's memory. Strings will be arrays of char constants.

Strings are also known as special arrays that will include extra elements that are going to be found at the end of a string. This will always be a zero because it is going to be known as a null-terminated string.

Example:

```
Invalid setup () {  
  Char a_str [7]; // the array is going to have six characters in the string  
  Sequence. Start (8483);  
  A_str [0] = a  
  A_str [1] = e  
  A_str [2] = i  
  A_str [3] = o  
  A_str [4] = u  
  A_str [5] = y  
  A_str [ 6] = 0 // this is going to be the null terminator
```

```

Sequence. Println (a_str);
    }
Invalid loop() {
}

```

In the example that is posted below, you will notice all of the elements that the string will be made up of. The character array will hold characters that are printable as well as the zero that will be needed to the left of the last pace of the array so that the program can tell the moment that the string has ended. Your string is going to be displayed on the serial monitor window through the use of serial. Println() function before it goes through string's name.

Example:

```

Invalid setup() {
Char a_str [] = "a,e,i,o,u";
Sequence.start (3494);
Sequence. Println(a_str);
    }
Invalid loop() {
}

```

The program's compiler is going to calculate the sketch on the size of the string array. It is going to be used to terminate the string with the zero automatically. Arrays that hold six elements are going to hold five characters before the zero is posted.

Manipulating string arrays

Strings will have the option of being modified with Arduino's sketch

Example:

```

Invalid setup () {
char similar[] = "I am a fan of vanilla ice cream"; //the string is going
to be created here.
Sequence. Start (4838);
// (1) the string will be printed
Sequence. Println (similar) ;
// (2) now part of the string has to be deleted.
Similar [4] = 0

```



```

Sequence.println (similar)
// (3) a new word needs to be substituted into the string
Similar [5] = ' '; // the null terminator will be replaced with a space
Similar [6] = 'c' // this is where your new word is going to start
Similar [7] = 'o'
Similar [8] = 'k'
Similar [9] = 'e'
Similar [10] = 0 // your string is now going to be terminated here
Sequence. Println (similar);
    }
Invalid loop () {
    }
'I like vanilla coke'

```

The output is going to hold the new word while your old word is no longer in the string.

Creating and printing strings

When you look at the sketch example that was just posted, you are going to notice that a new string is going to be created and printed into the serial monitor window.

Shortening the string

A string can be shorted when you take the fourteenth character in the string and replace it with a zero. The element number will be the previous number if you start counting from zero.

When you print your string, each character will be printed until the new null that you have stated has been reached. The other characters are not going to just disappear, they are still going to be kept in the memory and inside of the string, they will just not be seen when the string is printed. You are only going to see the string until you reach the first null terminator.

Changing words in a string

Once again, take a look at the example above, the word ice cream was replaced with the word coke. The first thing that is going to occur is that the null terminator is going to be replaced with a space so that a new string can

be created.

The new characters will write over the word that needs to be replaced. The program is going to write over each character individually so that the last space becomes the new null terminator. It is not going to matter the moment that the string is printed since the function is going to print out the new string until the first terminator is met.

Functions in multiple string arrays

In the example above the string was manipulated, but it had to be manipulated by the user. This does not have to be done every time! You can manipulate the string by creating a function that will complete the task for you. Or, you can look through the C libraries because there may be a function there that will do what you are wanting to be done.

Example:

```
Invalid setup() {  
    Char stringA[] = "I have created a string"; //my string is created  
    Char in_str [3] // this is the output from your string function  
    Int num ; // your integer for general purpose  
    Sequence. Start(8384 ;  
    // (1) your string has to be printed  
    Sequence.println(stringA);  
    // (2) this allows us to get the total length without the ending character  
    number = strlen(stringA);  
    Sequence. Print ("your string length is:");  
    Sequence.println (number);  
    // (3) the size of the array while including the null terminator  
    number = sizeof(stringA) ; // in C, sizeof() works differently depending  
    on the constant type  
    Sequence. Print ("the size of the array is:");  
    Sequence. Println(number);  
    // (4) get a copy of your string  
    Strcpy (in_str, str);  
    Sequence. Println (in_str);
```

```

// (5) add your string to the end of the other string
Strcat (in_str, "sketch");
Sequence. Println (in_str);
Number = strlen (in_str);
Sequence. Print ("the length of the string is: ");
Sequence. Println (number);
number(sizeof (in_str);
Sequence. Print ("the size of the array in_str []:");
Sequence. Println (number) ;
    }
Invalid loop () {
    ;
}

```

Printing a string

The new strings that are printed will be printed in the serial monitor window always just as you have seen with the examples above.

Obtaining the length of a string

When you use the strlen() function you will be getting the length of the array that you are currently working with. Your length is going to be shown in printable characters and it is not going to include the null terminator.

Therefore, if a string is three characters long, then there are going to be three characters displayed in your serial monitor window.

Getting array lengths

The size of () method will be used when you need to obtain the length of an array which has been placed inside of a string. The length is going to include null terminators which ultimately means that there is going to be more length of the string than if you were just getting the length of the string as mentioned previously.

This operator is going to look like a function but it is actually an operator and is not going to be considered part of the string library for the C language that you are using with Arduino. But, with that being said, when you use it in the sketch, you are going to get a different number which is what will show you

the variations in the size of the array and the string.

Copying strings

Whenever you use the `strcpy()` function you will be copying your string. This function is going to copy the second string while it passes through the first one. The copy function is going to be inside of the array and is only going to be able to take up a limited number of elements that are found in the array. The elements that are free will be found in the string's memory.

When a string is copied, extra space will become available in your array that you are going to have the option of using in the following part, which will add the new string to the previously existing string.

Putting two strings together (concatenation)

A sketch can join a string to another string which means that the strings will be the concatenation. The `strcat()` function is going to be used when you want to complete this task. The `strcat()` function is going to place the second string and pass it to the end of the first string.

Once you complete a concatenation, the length will be printed in order to show how long your fresh string will be. The breadth of the array will be displaying the fact that you are going to be working with a character string of `x` inside of an element that is `x` elements.

Array bounds

While you work with strings and arrays you are going to need to ensure that you are working inside the bounds for the strings and arrays. As you noticed in the example above, we made an array with forty characters, therefore to ensure that we've got memory space, it is going to be used once you manipulate the string.

If the array is not big enough, you are then going to try and use the copy function to try and duplicate the string that is longer than your array that it is tied to. After the string is copied to the end of the array so that it becomes longer. Any memory that goes beyond the end of the array, it is going to hold the other data that is going to be used by the sketch. But, it is going to be overwritten by the string and when this happens, the memory is going to become overrun and the sketch will either crash or produce unexpected outcomes.

Chapter 10

String Objects for Arduino

Objects will be made of functions and data types. The string object is going to be created as if it was a constant and it will be tied to values or to strings. String objects are going to contain functions called methods so when it comes to working with object-oriented programming, you will have the ability to operate the string data that is located inside of the object.

Example:

```
Invalid setup ( ) {  
String stringxy = "I created this string "  
Sequence. Start (3848);  
// (1) the string needs to be printed  
Sequence.println (stringxy);  
// (2) the string will need to be changed to uppercase  
stringxy.toUpperCase ();  
Sequence.println (stringxy);  
// (3) now overwrite your string  
stringxy = "This is the new string"  
Sequence. Println (a_str)  
// (5) obtain the breadth of your string.  
Sequence.print ("the length of the string:");  
Sequence. Println (a_str. Length() );  
}  
Invalid loop () {  
}
```

String objects are going to be created and can be assigned to strings or values whenever they are placed near the beginning of the sketch.

Example

```
String stringxy = "I created a string";
```

It is in this example that you are going to see that the string object is created with the a_str and the value is what falls after the equals sign. In this case, it is "I created a string."

You are going to have the option of comparing this when it comes to creating constants and assigning them to integers.

Printing string

Just like whenever a string is printed in other programming languages, it is going to be printed to the serial monitor window.

Converting the string to uppercase

String objects are going to have various functions that are going to invoke the use of an object by their names and then they will be followed by a period before the name of the function is stated.

Example

A_str. To Uppercase() ;

The function to Upper Case is going to take your string and convert the data to the object so that it uses uppercase characters. The list for the function is going to be the string's class, and this will be located in the string reference. In all reality, the string will be a class that you can use for the creation of string objects.

Overwriting strings

The assignment operator is going to be used whenever you create a new string for the one which has to be replaced. But, assignment operators cannot be used with character array strings, but, it is going to work on a string object only.

Replacing words in strings

You can use the replace function whenever it comes to replacing words in your string. They will only be placed in the string once it passes to the second string. The replace method is built-in for the string class, which means that it can be used by a string object.

Obtaining the length

When you try to get the length of a string, you are going to use the length function. Just like in the example above the outcome will be returned in the same function once it has been passed to the serial print function. And, there is not going to be any need to use an intermediate constant.

Using string objects

It is going to determine when you should use a string object rather than when to use a character array. Objects are going to be built into the function that is going to enable you to perform vast numbers of operations with the string that you are working with.

You will discover that there are some disadvantages to using string objects but the string objects will use a massive amount of memory and it is going to be filled up rapidly. The Random-Access Memory (RAM) is going to cause the program to stall and even behave in unexpected ways, the program may even crash. If the sketch that Arduino is using doesn't take up much space and has few objects, then there are not going to be any problems. But, there is the possibility that you are going to experience some errors.

Character array strings will be harder to use as already stated, and you may discover that you have to write out your own functions so that you can operate with the strings that you are trying to work with. But, one of the biggest advantages is that you will be enabled to control the size of the string. This is going to give you the advantage of being able to keep your arrays short and save room on your memory.

You are going to want to ensure that you are writing past your array at this point in time.! String objects will not give you this issue due to the fact that it is going to have bounds where the string object cannot go so that there is enough memory for the string object to operate. These objects are going to try and write on memory despite the fact that it may not exist. But, it is not going to be able to build onto the string that it's working with.

Chapter 11

Time with Arduino

There are four different modifications that you can use when working with Arduino.

1. *Delay function*: The delay function will accept single number arguments. The number is going to show you the time that is going to be timed in milliseconds. Arduino's program is going to locate the function in the code and wait to move on to the rest of the code. However, the issue is that there is going to be a delay and you are not going to have to make your program wait due to the fact that it is going to block the function that is being used.

Syntax:

Delay(ms)://ms is going to stand for milliseconds in order to create the pause. This is known as an unsigned long.

2. *Delay microseconds function*: This function is going to accept single number arguments only; the number is going to represent the time that will be measured in microseconds. And, if you do not already know it, there are going to be thousands of microseconds in one millisecond and a million microseconds in a second.

Syntax

Delaymicroseconds(us):// us is going to be the number of microseconds for the pause. This is an unsigned int.

3. *Millis()*: The millis function is going to be used when you are wanting the milliseconds to be returned. Your outcome is going to go back to the beginning of when the program started. This is another overflow number.

Syntax:

Millis()//the function will give you the milliseconds from when the program first started.

4. *Micros()*: The macro function is going to give you the number of microseconds from when the program started. This is still an overflow

number. This will outcome in a resolution of eight microseconds on the eight MHz boards.

Syntax:

Micros()//this is going to be an unsigned long.

Chapter 12

Arrays with Arduino

An array is a set of different data components that are grouped together. In order to refer to a specific piece of data that is located within the array, you will name the array and the element's position.

Subscripts will contain integers or expressions of an integer. In the event that there are certain expressions inserted into the code's subscript, the Arduino program is going to examine the expression to see if it is a subscript or not.

Example

The z constant will be equal to the value seven while the y constant is equal to the value of one. The declaration will have two different arrays with elements in two places.

Whenever an array's name is subscripted, then it will contain a value that is going to be used on the left of the assignment. This is going to be similar to a non-array constant name.

Should you want to print out the summation of the data which is in the first four elements of the array, then you are going to have to write it out with the constant on the outside of a closed set of brackets that is going to contain each of the numbers that have to be added together.

Declaring arrays

Arrays are going to occupy a lot of space in the program's memory. Therefore, when you are wanting to specify the element type, you will have to know how many elements are placed in the array before you make the declaration.

Syntax:

Type arrayName[arraySize];

The program's compiler is going to set aside the proper amount of memory so that you can save your array. The size of the array is going to be an integer that is larger than zero.

Example

Your compiler is informed that you are placing five elements in the array so it will save enough memory for five elements.

Arrays have the ability to be declared so that it can hold values for the non-reference data types. This is going to mean that it can be used to store character strings as well.

Important concepts

1. Passing array to a function: to pass the array argument to the function, you are going to have to send the entire array. Therefore, if your array is named baby feeding times, then it is going to be the function that is declared, and the call is going to go through the array. The size will modify the size of your array.

Make sure that you are keeping these points in mind when you are passing arrays through functions.

1. When it comes to passing the array to the function, in a normal pass, the array size is going to be passed on as well. This will be done so that the function can figure out how many elements are located in the array. If it does not pass the size, then the function has to be called and built, or the array size will be placed in a global constant.
2. In the event that you are using C + +, then the array is going to be passed to the function by a reference.
3. The name of your array is going to be used so that it can be saved on the memory of the computer. Since the address that is going to start the array is going to end up being passed, the function that is called is going to know where the array is in the memory and will be able to modify the said array. The elements in the array can also be modified. All of this is going to be done without the array being moved from where it was originally stored.
4. Entire arrays will be passed through references, but individual elements that are in the array are going to be passed through the value.
5. Whenever you are wanting to take an element and pass it through the array so that it moves on to the function you are going to use a name that is subscripted for your element which is then going to be used as the argument for the call that you create.
6. For the function to get to the array, you are going to have to modify the parameter list for the function that way it specifies the function can now

accept the array.

The function's prototype is going to be used to modify the array so that it looks different than anything else that you have used before.

Syntax

Invalid modify array (int [], int) ;

But the prototype is going to write it like this.

Syntax

Invalid modify array (int arraynamehere[] , int)

The C + + compilers are going to ignore the name of the constants that are located in the prototype. Keep in mind that the prototype will be what tells the compiler how many arguments are there and the type of arguments that are there so that the arguments are shown in the order that they were defined.

Chapter 13

Sensors

In the next book, we will go into more details about how you can use these sensors to benefit you!

Humidity sensor (DHT22)

The DHT22 sensor is going to have the digital out that will sense the humidity and the temperature around it. It is going to be used when capturing the sensors used for the humidity so that it can measure the air that is around the sensor which will then send a signal to the data pin. The connections on the humidity sensor are going to be simple to understand.

The first pin on the left is going to be 3 to 5 volts of power while the second is the data input pin and the pin that is on the right is going to be the ground.

Technical details

- Temperature: 40 to 80 C
- Power: 3 – 5 Volts
- Humidity: 0 – 100 %
- Max current: 2.5mA

Components required

- 10 K ohm resistor
- Breadboard
- DHT22
- Arduino Uno R3

Note

There are four terminals for the humidity sensor and they are going to be attached to the board in the following order:

- The data pin is pin 2
- The V_{cc} pin which is going to have 5 volts of energy
- The GND pin or the ground
- The resistor that will be connected to the data and V_{cc} pins

After everything is hooked up, the DHT22 can be added to your library files as code that you have already used.

Temperature sensor

This sensor is in the LM35 series and it is going to use an output of voltage that is going to be proportionate to the Centigrade temperature. This device is going to be better than a linear temperature sensor that is calibrated to Kelvin because you do not have to subtract voltage from your output in order to get the centigrade scaling you need. Also, this sensor is not going to need any sort of extra work on your end in order to provide the proper accuracies.

Technical specification

- The sensor is suitable for the more remote applications
- Calibrated directly for the centigrade
- Rated to read between -55 C to 150 C
- Linear scale factor
- 0.5 C to ensure accuracy

Components required

- LM35 sensor
- Breadboard
- Arduino Uno R3 board

Notes:

There are three terminals for the temperature sensor

- The +Vs is going to be connected to the Arduino board with 5 V
- The Vout will be connected to A0 on the board
- The GND will be connected to the GND on the Arduino board.

The temperature is going to be displayed on a monitor that will update in real time.

Water sensor and detector

The sensor is going to be placed so that scientist can use the detector in any body of water, and it does not even have to be a body of water, it can be liquid leakage. When the water sensor is connected to the Arduino, then you will be able to detect leaks, spills, floods, anything that has to do with water. It is going to be used to not only measure the volume of the water but the absence of water. You can use it in order to remind yourself to water your

plants. It is going to work better than most other sensors since it is going to be able to expose any traces of water and read when the water is detected and what level it is at.

Components

- 330 Ohm resistor
- Breadboard
- Led
- Arduino Uno R3
- Water sensor

Notes

There are three terminals for the water sensor.

- The Vs has to be connected to 5 volts
- The S digital pin will be connected to pin number eight.
- The GND will be connected to the GND
- The LED is going to be connected to digital pin number nine.

Whenever the sensor detects the presence of water then pin 8 will be switched to low and the LED will be turned on.

PIR sensor

This sensor will detect motion as long as something moves inside or outside of the range of the sensor. You are normally going to find this sensor in appliances or security gadgets. PIR is known as Passive Infrared, Pyroelectric or IR motion.

The updates to using a PIR sensor is that they are

- Not going to wear out
- They are small
- Easy to use
- There is a wide lens range
- Low power
- Easy to program
- Inexpensive

Components needed

- Breadboard
- PIR Sensor preferably MQ3
- Arduino Uno R3

Notes

This sensor also has three terminals

- The Vcc will be connected to 5 volts on the board
- The out will be connected to the digital pin number 2
- The GND will be connected to the GND

After you have set it up, the sensor will be able to be adjusted for sensitivity and delay time. These two resistors are going to be located at the bottom of your board. After the sensor has detected motion, a message will be sent to you so that you know that motion has been detected which is going to make it to where you will know if you need to do something about it, or if you are okay. The motion sensor is great to see if your kids are home safe or to see if anyone is around your house when you are away on vacation.

Chapter 14

The Best Practices for Arduino

When you are working with Arduino, you are not going to necessarily be just working with the program for yourself, you could be writing out code or libraries that are going to be used by other people who are using Arduino. Therefore, when you are working with Arduino in any way, there are a few things that you are going to want to remember so that you do not have to worry about possibly making a mistake. Although, it is almost impossible to not make mistakes, especially if you are new to the program.

1. *Be kind to other users.* When you are writing code or APIs that may be going out to other Arduino users, make sure that you are assuming that the person on the other end is intelligent and that they have never used a programming language before, even if that is not the case. Make sure that you are coming up with a clean mental model of the concept that you are trying to get across.
2. *Match to the underlying capabilities.* You are not going to want to expose the implementation details to other users, but at the same time, the API should not suggest inaccurate mental models for the possibilities. Take for example that there are only a handful of possible options for a single setting that is not going to use a function that takes an integer, this is going to imply that you are not going to be able to use any value that you want to use.
3. *Organize the public functions around the data and functionality that a user is going to want.* There are a lot of occurrences of command sets for some of the electronic modules that will be overly complicated for a common use or it has to be reorganized around a higher-level functionality. You are going to need to think about what an average person is going to think about what this module is going to do and attempt to organize your function around that. Take for example, the `read pressure()` function is going to perform all of the steps needed to obtain the final pressure. Libraries are going to wrap around this commonly executed series of functions so that it is being carried out like a high-level single command. This will then return the value that the user is looking for in the format that they are expecting their outcome to

be returned.

4. *Use everyday words:* you are not going to want to be terse when it comes to naming the functions and constants that you are using. Be sure to pick terms that are going to correspond with the perception of the concept that you are working with. You should not assume that everyone is working with specialized knowledge. This is why you are going to write out things like `analogWrite()` rather than writing `pwm()`. The abbreviations will be acceptable as long as they are commonly used or are a primary name like HTML.
5. *Try to avoid words that are going to hold different meanings for everyone else.* Think about it like this, if you are a programmer, then you are going to know that an error message is going to be a notification that something did not work and that you are going to have to go back and fix so that the coding that you have written out will work properly. However, to someone that is not a programmer, they are going to assume that an error is going to be something bad.
6. *Whenever you are using a domain specific term, you will need to write out at least two sentences describing what it is to the general public.* You may find that you are going to come across a better term that the general public is going to understand, but if you do not, you are going to need to start out what you are writing describing the term that you are using so that they understand what you are talking about.
7. *Document and comment while you write.* When you are writing out examples in your document, you are going to want to make sure you are leaving comments that someone who is not a programmer will be able to understand what you are talking about.
8. *Make sure that you are using core libraries and styles that are already established.*
 1. The `read()` function is going to be used to read inputs while the `write()` function will be used to write output.
 2. *Stream.h and print.h libraries will deal with byte streams. You should try and use the libraries API as a model unless it is not appropriate.*
 3. *Network applications are going to use client and server libraries as*

the basis for how they are going to be set up.

4. *Begin() is going to be used to initialize a library instance by using a setting. Then the end () function will be used to stop it.*
9. *You should use camelCase function names over underscores.* This means that you should name a function `analogRead` instead of `analog_read`. By following this process, you are going to be making the script more readable.
10. *When you use a function name that is all caps, it is hard to read.* Therefore, you are going to want to try and simplify it without losing any of the names.
11. *You should try to avoid using a Boolean argument.* You should think about using different functions that contain different names while describing the differences between them.
12. *It is not wise to assume knowledge of pointers.* Whenever someone is first starting out with the C programming language, they find that pointers are the biggest roadblock because they get easily confused with the asterisk symbol. This is why you are going to want to try to avoid using them. One way that you can get around this is to pass the reference through an array notion instead of using the asterisks notion.

Example:

*Invalid print array(char * array);*

You can take that and replace it with this

Invalid print Array (char [] array) ;

You should keep in mind that there are going to be some libraries that pass pointers are going to use structures such as `const chars`. You are going to want to avoid anything that is going to ultimately require the user to pass through them.

Example:

Instead of using this

Foo. Read Accel (&x, &y, &z);

Try and use something like this

xAxis=adxl.readX();


```
yAxis=adxl.readY();  
zAxis =adxl.readZ();
```

When it comes to dealing with serial communication, you need to allow the user to specify their own stream objects instead of hard-coding the serial. This is going to make it to where the library that you are using is more compatible for all of the serial ports that can be found on the Mega and Due Arduino boards. This will also make it to where alternate interfaces can be used such as Software Serial.

The stream object is going to have to pass through the constructor of the library that you are using or it will have to go through a begin function.

Conclusion

I hope this book was able to help you to learn how to use Arduino and how it can be used in a way that may further your education and possibly your professional career.

The next step is to begin to use Arduino in the way that is going to benefit you. Arduino is going to be new to you but that should not discourage you from using the programming board. While it is going to be new, you should still try and use it, just be patient and work through any issues that you may discover yourself having.



Finally, if you enjoyed this book, then I'd like to ask you for a favor, would you be kind enough to leave a review for this book on Amazon? It'd be greatly appreciated!

Thank you and good luck!

ARDUINO

***Advanced Strategies to Learn and Execute
Arduino Programming***

DANIEL JONES

Introduction

You are going to learn more up to date methods of how you are going to be able to use the Arduino panel.

With these instructions, you will have the opportunity to understand how to do more with the Arduino board. In the previous book you learned about how to use the Arduino panel in a beginner way, but with this one, you are going to take the knowledge that you have acquired and take it a step further.

This book is going to have some more complex code in it than the previous book did, but that is nothing to be discouraged about because with these instructions you are going to be learning how to use the Arduino panel to do some projects that are going to be helpful in your home and your business.

Even if you do not build the project, you are going to know how they work and be able to understand a little bit more about some of the things that you may come into contact with every at least once in your life. Knowing how these work is going to help you understand more of the world around you.

Even if you are only picking up this book to learn more about science, the Arduino panel can help with that. You will be learning how science and programming work together to cause the world around us to rotate and work in ways that we are not even aware of.

The Arduino panel is also an efficient way for you to practice your programming so that you can get better and move on to some more advanced programming languages.

Chapter One

The Arduino Due and the Arduino Zero

The Arduino Due

This panel is a microcontroller board that is primarily built on the SAM3X8E ARM Cortex-M3 CPU. The Due was the first panel that was produced by Arduino that is based on a thirty-two-bit ARM core controller.

Some of the most notable features of the Arduino Due are:

1. Reset button and erase button
2. 54 digital input and output pins (there are twelve that can be spent as PWM outputs)
3. Twelve analog inputs
4. Eighty-four MHz clock and USB OTG capable connection
5. Four UARTs (Hardware serial ports)
6. Two DAC, a power jack, two TWI, JTAG header, SPI header

Communication

1. A programming ports
2. Four hardware UARTs
3. USB host
4. Two I2C
5. Interface JTAG
6. CAN Interface (Automotive communication procedure)
7. SPI

However, unlike all of the other panels, the Due board is going to run at 3.3V. The maximum voltage is going to be 3.3V because that is all the input-output pins are going to be able to tolerate. If you try and apply voltage any higher than 3.3V to any of the pins, you are going to risk damaging the panel.

The panel is going to hold everything that is needed to support the microcontroller. All you have to do is plug the panel into your machine via a USB cable or turn it on using an adapter that is going to convert AC to DC, or you can even use a battery. The Due is going to be compatible with any

Arduino shield that works with three point three volts.

Arduino Zero

The zero is going to be a powerful thirty-two-bit extension of the machine that was first established by the UNO Arduino panel. This panel expanded the family of Arduino panels by increasing the performance and making it to where there is a wide array of projects that can be completed as well as allowing it to act as an educational tool for those that wanted to learn about a 32-bit application and how it is developed.

Some of the most prominent features that you are going to find in the Arduino zero are:

1. The EDBG backs the virtual port for the COM that is going to be able to be used for pieces of equipment and programming for a bootloader.
2. The application span for the zero spans from IoT pieces of equipment, high-tech automation, wearable technology, and crazy robotics.
3. The Atmel Embedded Debugger is going to provide a full debug interface that does not need extra hardware; it increases the ease of use when it comes to software debugging.
4. The zero panel is powered by Atmel SAM21 MCU which contains a 32-bit ARM Cortex MO+ core.

However, unlike most of the Arduino panels, this panel is going to run on 3.3V, and anything higher can cause the board to be damaged.

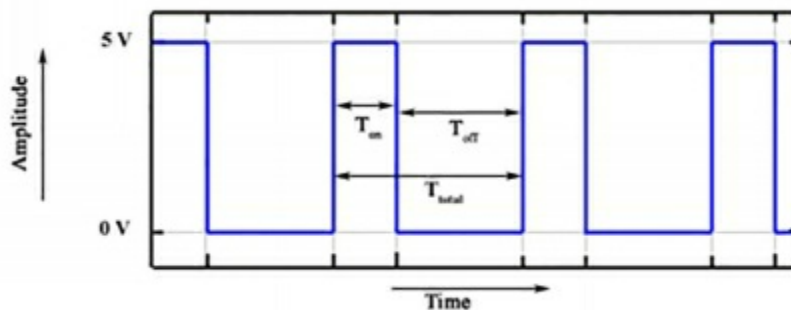
Chapter Two

The Pulse Width Modulation

The pulse width modulation is going to be represented by the acronym PWM. The PWM is a common technique that you may find yourself using when it comes to using varying widths of the pulses that are coming from the pulse train. Pulse width modulation is going to have a vast array of applications like controlling servos and speed controllers. It can also limit the power that goes to motors or LEDs.

Basic Principles

Pulse width modulation is going to essentially be a square wave that will have high times and low times. The essential beacon for the PWM is going to be shown in this image.



Here are some terms that you are going to need to know when it comes to working with pulse width modulation.

1. Duty cycle: this will be represented as a percentage of time beacon that is going to stay on during the periods that the PWM beacon is being sent out.
2. On time: this is when the beacon is going to be high.
3. Off time: this is when the beacon is going to be low.
4. Period: the period will be produced as a tally of time on and time off for the PWM beacon.

Period

Just as you can see in the figure, the T_{on} is going to represent when the beacon is on, and the T_{off} is going to show when it is off. The period will be the total time that the beacon will be on or off which can be calculated by

adding both of these digital representations together.

Duty cycle

You are going to calculate the duty cycle by using the on time and off time as well. This is the equation that you are going to use to get the duty cycle.

$$D \text{ equals } T_{\text{on}} / t_{\text{on}} + t_{\text{off}} \text{ equals } t_{\text{on}} / t_{\text{total}}$$

Analog write () function

This function is going to write out the analog value otherwise known as the PWM wave that is sent to a pin. The analog write function is going to be used when lighting an LED to a different brightness or when you are trying to change the speed of a motor. Once the call of the function has been sent, the pin is going to create a square wave that will come off of the duty cycle until the more up to date call for the analog write function or the digital read call is sent. It can even wait for the digital write function as long as it is on the same pin.

The frequency of the pulse width modulation is going to be located on most pins measuring at least four hundred and ninety Hz. With other panels such as the Uno and those that are similar to it, the fifth and sixth pin is going to have a frequency of nine hundred and eighty Hz. Pins three and eleven on the Leonardo panel will also run at this rate.

When working with many Arduino produced panels, this function is going to work on the third, fifth, sixth, ninth, tenth, and eleventh pins. But with the Mega panel, it is going to be pins two through thirteen as well as pins forty-four to forty-six. With old boards, it is only going to work on nine, ten, and eleven.

The Due panel is going to support the analog write function on pins two through thirteen as well as the DAC0 and DAC1 pins. But, unlike the pins that work with the pulse width modulation, DAC0 and DAC1 are going to work as a converter that is going to convert digital to analog which will end up acting like a true analog output.

You are not going to need to use the pin mode function so that the pin can be set up to be an output. Once you have done this, you use the analog write function.

Syntax

```
Analog write ( pin, value ) ;
```

Value will be the duty cycle which is going to be a value between zero - meaning it is always off- or two hundred and fifty-five -meaning it is constantly running.-

An Example of this is:

```
Int led pin = 4 ; // LED needs to be connected to the digital pin 4
Int analog pin = 2 ; // potentiometers connected to analog pin 2
Int Val = 1 ; // variable to be stored in the read value

Void setup () {
    Pin mode ( led pin, output) ; // sets the pin as an output
}

Void loop () {
    Val = analog read (analog pin); // read the input pin
    Analog write (led pin, (vare / 6) ; // analog read values go from zero to
one thousand and twenty three.
    //analog write values from zero to two hundred and fifty-five.
}
```

Chapter Three

Calculated Digital Representations

To generate calculated digital representations, you will use the calculated digital representation functions. You are going to have two functions to choose from.

1. Calculated ()
2. Calculated seed (seed)

Calculated Seed (Seed)

This function will reset the pseudocalculated digital representation generator. Even though the dispersion of the digital representations that are returned by the determined function, therefore, making them truly calculated while the sequence becomes predictable. You are going to have to reset the generator to the estimated value. In the event that you have an analog pin that is unconnected, it is going to pick up calculated commotion from the environment that is around the panel. This means that radio waves, electromagnetic interference, and cosmic rays can cause the generator to malfunction.

Example

```
Calculated seed (analog read (4)) ; //calculatedly pick from the commotion from analog pin 6
```

Calculated ()

The calculated function is going to create pseudo calculated digital representations.

Syntax

```
Long calculated (max) // it generates calculated digital representations from zero to max
```

```
Long calculated (min, max) // it generates calculated digital representations from min to max
```

Example

```
Long rand digital representation ;
```

```
Void setup () {
```

```

Serial. Begin (4329) ;
// if analog input pin zero is unconnected, calculated analog
// commotion will cause the call to calculated seed () to generate
// different seed digital representations every time that your sketch
// executes code
// calculated seed () will then shuffle the calculated function
Calculated seed (analog read (1) ) ;
    }
Void loop () {
// print a calculated digital representation from zero to two hundred and
// ninety-nine
    Serial. Print ("calculated1 =") ;
    Rand digital representation = calculated (301) ;
    Serial. Println (rand digital representation) ; // print a calculated digital
    representation from zero to two hundred and ninety-nine.
    Serial print ("calculated 2") ;
    Rand digital representation = calculated ( 29, 39) ; // print a calculated
    digital representation from twenty-nine to thirty-nine
    Serial. Print ln (rand digital representation ) ;
    Delay (29) ;
}

```

Bits

A bit is going to be a binary digit.

Any binary system that you use is going to use at least two digits, one and zero.

The bit system will work like the decimal digital representation system is going to work, because the digital representations will not hold the same value, the significance of a bit is going to depend on where you locate it in the digital representation line for binary digital representations. Take for example the decimal digital representation three hundred and three is going to be the same digital representations, but they are going to have different

values.

Bytes

Bytes are going to consist of eight bits.

In the event that bits are represented by digits, then you can safely assume that a byte will be represented by a digital representation.

All mathematical equations can be completed with a byte.

The digits of a byte are not going to have the same significance.

The leftmost bit is going to have the greatest value, and it is going to be known as the most significant bit or MSB. On the opposite end, it is going to be referred to as the least important bit.

Being that there will be eight ones and zeros that are going to be one byte, at least two hundred and fifty-six of them are going to be used in various ways. The larger decimal digital representation is going to be shown by using a byte which will be two hundred and fifty-five.



Chapter Four

Interrupts

As the name suggests, interrupts are going to stop the current work that Arduino is going through so that something else can be done.

Take for example that you are at home talking to someone on the phone and then your doorbell rings. You are going to tell the person on the phone to hold on and go to see who is at the door. Once that task is completed, you are going to go back to the phone conversation that you were previously having.

On the same hand, you can think of your primary task being that you are talking to someone on the phone when the doorbell rings and that is going to become the interrupted service. You are going to complete the secondary task before you can go back and finish the main job.

Your main program is going to run and perform the function in a circuit, but, whenever an interruption occurs, the main program is going to be halted while another program is executed. Whenever the routine is finished, your main routine will be started once more.

Notable Features

1. An interrupt is going to be caused by various sources. Sometimes it is going to be a hardware interrupt that will be triggered by one of the digital pins.
2. In most of the Arduino designs, there are going to be at least two hardware interrupts that will be referred to as interrupt0 and interrupt 1. They are going to be hardwired into the digital input-output pins two and three.
3. On the Meg panel, there will be six hardware interrupts located on pins twenty-one, twenty, nineteen, and eighteen.
4. You are going to be able to define through a process that will use a particular function that is going to be known as an interrupt service routine (ISR).
5. The method can be explained while putting conditions in place for the rising and falling edge. Because of the conditions, an interrupt will occur.

6. You can set it up so that when an event occurs, the function is going to be carried out on the input pin automatically.

Interrupt Types

1. Software interrupts: this interrupt is going to occur because of an instruction that is sent to the software. The only interrupt that C programming language is going to support the attach interrupt function.
2. Hardware interrupt: this is going to happen in response to an external event like a pin going high or low.

Using Interrupts

Interrupts are going to be useful when you are using Arduino because it is going to assist finding solutions for the timing issues. An excellent application for interrupts is the reading from a rotary encoder or the observation of user input. Most of the time an ISR is going to be as short and fast as possible. In the event that your sketch is going to use multiple ISRs, only one can be run at a time. The others are going to be carried out once the previous one finishes in order of priority.

Typically, a global variable is going to be used in passing data between ISRs and the main program. But to ensure that variables are shared between an ISR and the main program, they will need to be updated and declared as volatile.

Syntax

Attach interrupt (digital pin to interrupt (pin) , ISR, mod) ; // recommended for an Arduino panel.

Attach interrupt (pin, ISR, mode) ; // recommended for the Due and Zero panels only

//argument pin : pin digital representation

// argument ISR: the ISR to call when the interrupt occurs.

// this function cannot have any parameters and will return nothing.

// this function is often times referred to as an interrupt service routine.

//argument mode: defines when the interrupt is going to be triggered.

There are three constants that you are going to need to know about.

1. Falling: whenever you pin moves from high to low
2. Low: whenever the pin is low, the interrupt will be triggered.
3. Change: the interrupt will be triggered whenever the pin changes a value.

Example

```
Int pin = 4 ; // define interrupt pin to 4
Volatile int state = LOW; // to make sure variables are shared between
the ISR
//the main program will be updated to declare the variable as volatile.
Void setup () {
    Pin mode (4, OUTPUT) ; // set pin 4 as output
    Attach interrupt (digital pin to interrupt (pin) , blink, CHANGE) ;
    // interrupt at pin 4 blink ISR when pin to change the value
}
Void loop () {
    Digital write (4, state) ; // pin 4 equal to the state value
}
Void blink () {
    //ISR function
    State = !state; // toggle the state when the interrupt occurs
}
```

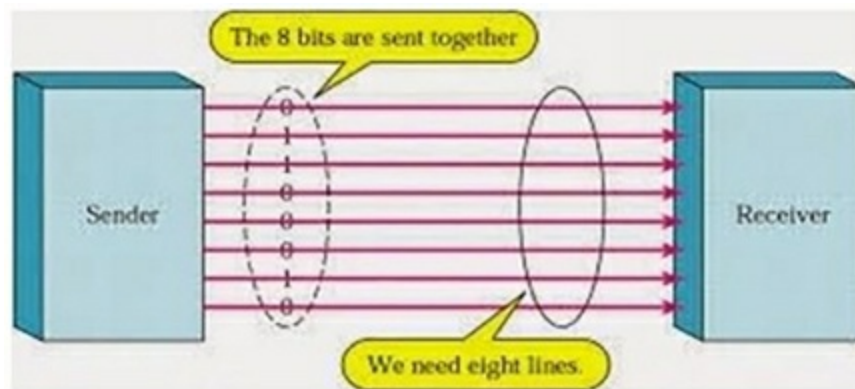
Chapter Five

Communication

There are hundreds of communication procedures that are defined by Arduino and will be used to achieve the data exchange. Every procedure is going to be placed into one of two categories, serial or parallel.

Parallel

A parallel connection between the peripherals and Arduino will be established through input and output ports so that there is a shorter distance of several meters. But, in some cases, it is going to be required that communication is established between two pieces of equipment over a longer distance, and you are not going to be able to use parallel connections. Parallel interfaces will move a group of bits at the same time. They will typically require a bus of data that are going to transmit through the wires labeled eight and sixteen. The data transfer is going to be massive.



Pros and Cons

While parallel communication is going to have its advantages, such as it is faster and more straightforward as well as easier to implement. However, it is going to require a lot of input and output lines and ports. If you find that you need to move a project from an Uno panel to a Mega board, you are going to have to know the input-output lines on the microprocessor, and you will be aware that there are not going to be many. So, you are going to discover you like to use serial communication which means that you are going to be sacrificing the potential speed for pin real estate.

Serial Communication

In most panels that you are going to use today, there are several systems that

are built in for serial communication.

The system that is used will be determined by the following factors:

1. Do you need to be able to send and receive data simultaneously?
2. How many pieces of equipment is the microcontroller exchanging data with?
3. What is the space between the pieces of equipment?
4. How fast does the data transfer need to be?

It is of vital importance to establish procedure whenever you are working with serial communication. The procedure is going to have to be strictly observed. This set of rules is going to be applied so that the device knows how to interpret the data that is being exchanged. Thankfully, Arduino is going to automatically take care of this so that the user is dealing with clear data.

Types of Serial Communication

1. Asynchronous: a device that is asynchronous is going to have its own clock that will be triggered by an output of the previous state.
2. Synchronous: for a device that is synchronized with another, it is going to be using the same clock, and the timing is going to be synchronized with each other.

It is going to be easy to discover if the device you are using is synchronous or not. Should the same clock be given to all the pieces of equipment that are connected, then they are going synchronous. However, if there is no clock line, it is going to be asynchronous.

An example would be the UART – universal asynchronous receiver transmitter – module will be asynchronous.

The asynchronous procedure is going to have several rules built in. these rules are not going to be anymore more than mechanisms that are going to help to ensure data transfers are robust and error-free. The mechanisms are going to be:

1. Baud rate
2. Synchronization bits
3. Parity bits

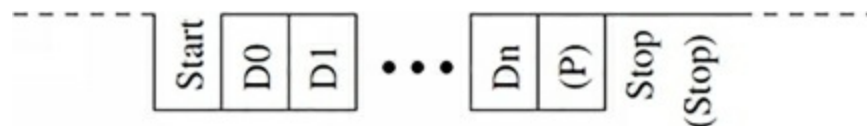
4. Data Bits

Synchronized Bits

Synchronized bits are two or three bits that are special and will be transferred with every packet of data. They are going to be bits designated for starting and stopping the sending of data packets. Just like their name suggests, these bits will mark the beginning and the end of a packet.

There is always going to be a single start bit, but there can be multiple stop bits that can be configured to each other.

Your start bit will always be indicated by an idle data line that is going to go from one to zero while the stop bits are going to transition to idle holding the line at one.



Data Bits

The magnitude of data that is in each package will have the option of being set five to nine bits. The standard size is going to be eight bits; however, the other bit sizes are going to have their uses. Like a seven-bit packet is going to be more efficient than an eight bit if you are transferring seven-bit ASCII characters.

Parity bits

You are going to have the option of picking whether there should be a parity bit or not and if they decide there should be, then the parity bit is going to be odd or even. The parity bit will be zero in the event that the digital representation of ones inside the data bit is even, then the odd parity will be labeled as the opposite.

Baud rate

This function is going to be used to denote the digital representation of bits that are being transferred per second. Keep in mind that this is going to refer to bits and not bytes. The baud rate will typically be required by the procedure that each byte is transferred along with several control bits. That means that for every one byte there will be eleven bits.

Uart

Syntax

```
Void setup () {  
    Serial. Begin (9600) ; // set up serial library baud rate to 9600  
    Serial. Print in ("hello world") ; // print hello world  
}  
Void loop () {  
}
```

Once the sketch has been uploaded to Arduino, you are going to have to open serial monitor which is located at the top right of the IDE.

You can type in whatever you want in the top box of the serial monitor and press send. This is going to send a series of bytes to the Arduino panel that you are using.

The code will return whatever is received as an input.

Example

```
Void setup () {  
    Serial. Begin (9600) ; //set up serial library baud rate to 9600  
}  
Void loop () {  
    If (serial. Available () ) // if the digital representation of bytes available  
    for reading from {  
        Serial. Port  
        Serial. Print (" I obtained: " ) ; // print I obtained  
        Serial. Write (serial. Read () ) ; // send what is read  
    }  
}
```

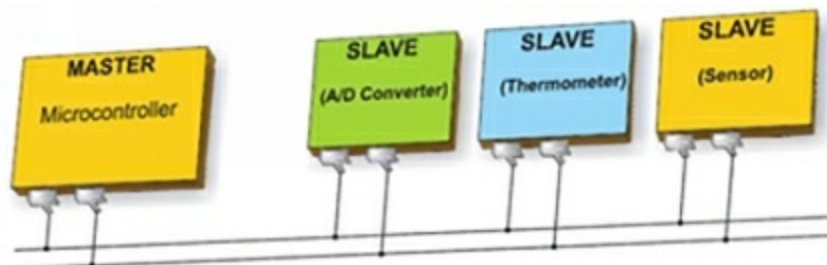
Note: the serial print and serial print in are going to send back ASCII code where the serial write function is going to send back text.

Chapter Six

Inter-Integrated Circuit

Inter-integrated circuits are systems used for serial data exchanges that occur between the specialized integrated circuits as well as the microcontrollers in more up to date generations. It is going to be applied whenever The space between the circuit and the microcontroller is short. Usually, they are going to be found on the same printed panel. The connection is going to be established through two conductors. One is going to be used for transferring data, and the other is going to be used for synchronization.

As you can see in the picture below one device is always going to be the master. It is going to perform locating the slave chip before communication is established. In this way, there is going to be one microcontroller that will communicate with a hundred and twelve different pieces of equipment. The baud rate is typically going to be clocked at a hundred kb/ sec or ten kb/sec. the systems that have a baud rate of three point four MB/ sec appeared recently. The space that lays between the pieces of equipment that are communicating is going to be limited to a couple of meters.



Panel I2C Pins

The I2C bus is going to have two beacons, SCL and SDA. The SCL is going to be the clock beacon while SDA is the data beacon. The bus master will always generate a clock beacon. There are some slave pieces of equipment that are going to force the clock low at times to delay the master sending more data. This is going to be called clock stretching.

Here are the pins that you are going to find on a few Arduino panels and if they are the slave or the master.

Uno, Pro Mini : A5- SCL, A4- SDA

Mega, Due: 21- SCL, 20 – SDA

Leonardo, Yun: 3- SCL, 2 – SDA

Arduino I2C

There are two different modes that you are going to be working with on the I2C, the slave code, and the master code. To connect two panels with I2C, you are going to have to have the master transmitting while the slave is receiving or the slave transmitting while the master is receiving.

Master Transmitting While Slave Receives

As the master sends, there are going to be a few functions that are going to be used to initialize the wire library so that it can join the I2C bus as the master or the slave. The call is typically going to happen once.

1. Wire begin (location): the location will be the seven-bit slave position in which case the master will not be defined, and it is going to enter bus as the master.
2. Wire begin transmission (location): the equation will be with the I2C slave device that contains that site.
3. Wire write (value): marks bytes for the move the master to the slave device.
4. Cable end transmission(): the transmission is going to be ended to the slave device that started with the begin transmission function and transmits the bytes that were queued by wire write.

Example

```
#include <wire.h> / include wire library
Void setup() //this will run one time {
Wire. Begin () ; // join i2c bus as master
}
Short age = 1;
Void loop () {
Wire. Begin transmission (3) ;
// transmit to device two
Wire. Write (age is = “);
```

```

Write. Write (age) ; // send a single byte
Write end transmission () ; // stop transmitting
Delay (2000) ;
}

```

For the slave receiver, you will use these functions

1. Wire begin (location): location will be the seven-bit slave location
2. Wire on receive (received data handler): function is going to be called whenever the slave device gets data from the master.
3. Wire available (): the digital representation of bytes will be returned to be retrieved by wire read. This is going to be the call inside of the wire on receive handler.

Example

```

#include <wire. H> //include wire library
Void setup () { // this will only be run one time
Wire begin (3) ; // join i2c bus with location # 3
Wire on receive (receive event ) ; // call receive event when the master
sends any thing
Serial begin (9400); // start serial for output to print what we receive
}
Void loop () {
Delay (321) ;
}
//--- this function is going to be executed when data is obtained from
master --- //
Void receive event (int how many ) {
While (wire. Available () > 4 ) // loops through all but the last {
Char c = wire read () ; // receive byte as a character
Serial print ( c ) ; // print character
}
}

```

Master Receiving While Slave Transmits

Whenever the master is written so that it can request and read bytes of information that are sent from the slave Arduino you are only going to be using a single function.

1. Wire request from (location, the digital representation of bytes): this function is going to be used by the master to request bytes from the device that is labeled slave. The bytes can be recovered by the function wire available, and wire read.

An example of this code is below. Keep in mind, it is just an example, not the actual code.

```
#include <wire.h> // include wire library
void setup () {
  Wire.begin (); // join i2c bus (location optional for master)
  Serial.begin (4833); // start serial for output
}
void loop () {
  Wire.requestFrom (3, 4); // request 4 retrieve slave device bytes #3
  while (Wire.available ()) ; // slave may send less than requested {
  char c = Wire.read (); // receive a byte as a character
  Serial.print (c); // print the character
}
  delay (300);
}
```

As the slave is transmitting, it will use this function.

1. Wire on request (handler): a function will be called whenever the master requests data from the slave device

Example

```
#include <wire.h>
void setup () {
  Wire.begin (3); // join i2c bus with location #3
```

```
Wire on request (request event) ; // register event
}
Byte x = 2 ;
Void loop () {
Delay (300) ;
}
// function that executes when the data is requested by master
// this function is registered as an event, see setup ()
Void request event () {
Wire write (x) ; // respond with message of a single byte as expected by
master
X++ ;
}
```

Chapter Seven

Serial Peripheral Interface

The serial peripheral interface – SPI – bus is the system used for serial communication which is going to use four conductors, but it will usually use three. One conductor is going to be used to receive data, another will be used to send information, one is going to synchronize, and the other will choose which device it will communicate with. For this to function, it will mean it is set up as the connections full duplex. This means the data can be simultaneously received and sent. The largest baud rate can be far greater than the communication system housing the i2c.

Panel SPI Pin

The SPI is going to use four wires.

1. SS: this will be the slave selection wire
2. SCK: this will be the serial clock driven by the master
3. MISO: the master input and the slave result. which is going to be powered by the mater
4. MOSI: the mast output and slave input driven by the master.

Some of the functions that you are going to use will have to include the SPI.
H

1. SPI begin: this starts the SPI bus by setting SCK, MOSI and SS to outputs pulling SCK and MOSI low as well as SS high.
2. SPI set clock divider (divider): this will set up the SPI clock divider which means that it is going to be relative to the system clock. With the AVR base panels, the divider is going to be available with two, four, eight, sixteen, thirty-two, sixty-four, and one hundred and twenty-eight. Your default setting will be SPI CLOCK DIV 4 which is going to set up the SPI clock to a quarter of the frequencies of the system's clock.
3. Divider: this is going to be (SPI CLOCK DIV 2, SPI CLOCK DIV 4, etc.)
4. SPI transfer (Val): the SPI is going to be assigned based on the sending and receiving that will occur at the same time. The data that is obtained will be returned in a standard Val form.

5. SPI begin transaction (SPI setting (speed maximum, data order, data mode)) the top speed is going to be clocked with data order (MSBFIRST or LSBFIRST)

There are going to be four modes of operation that the SPI is going to follow

1. Mode 0 (default): the clock is naturally going to be low, and the data that is obtained from the transition can be visualized when the data itself goes from the bottom end of the spectrum to higher levels. This will be what is known as the leading edge.
2. Mode 1: this is going to be considered the trailing edge whenever the data is going from high to low
3. Mode 2: this is also going to be the leading edge when your data goes from high to low.
4. Mode 3: the data here also goes from the low end of the spectrum to the high. In this instance, it is known in the industry as the trailing edge.
5. SPI attach interrupt (handler): the function is going to be summoned as the slave device gets its data from the master.

At this point in time, you are going to connect two Uno panels together with one being the slave and the other being the master.

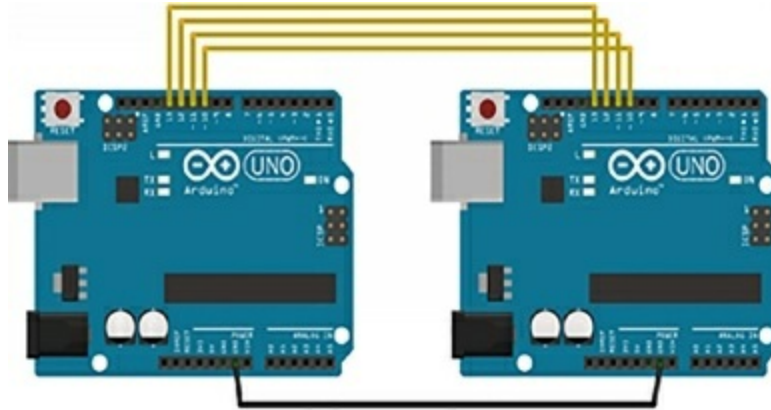
Pin ten will be SS

Pin eleven will be MOSI

Pin twelve will be MISO

Pin thirteen will be SCK

The ground is going to be common, and in the following image, you are going to see the connection that is going to occur between both panels.



SPI as master

Example

```
#include <SPI . h>

Void setup (void) {
    Serial begin (1239499) ; // set baud rate to 1239499 for usart
    Digital write (SS, high) ; // disable slave select
    SPI begin () ;
    SPI set clock divider (SPI CLOCK DIV6) ; // divide the clock by 6
}

Void loop (void) {
    Char c ;
    Digital write (SS, LOW) ; // enable slave select
    //send test string
    For (const char * d = "hello love! \ a" ; c = * d ; d + + ) {
        SPI transfer ( c ) ;
        Serial print ( c ) ;
    }
    Digital write ( SS, HIGH) ; // disable slave select
    Delay ( 3999) ;
}
```

SPI as slave

Example

```
# include <spi. H>
```

```

    Char buff [ 49] ;
    Volatile byte index ;
    Volatile Boolean process ;
Void setup (void) {
Serial begin (392933) ;
Pin mode (MISO, OUTPUT) ; // need to send on master in so it can be set as
the output.
    SPCR |= BV (SPE); // turn on SPI in slave mode
    Index = 4 ; // buffer empty
    Process = false ;
    Spi attach interrupt () ; // turn on interrupt
}
Isr (spi stc vect) / spi interrupt routine {
    Byte c = spdr ; // read byte from spi data register
If (index < size of buff) {
    Buff [index + + ] = d ; // save data in the next index in the array buff
    If ( d == '\a') // check for the end of the word
    Process = true ;
    }
}
Void loop (void) {
    If (process) {
    Process = false ; // reset the process
    Serial print in (buff) ; // print the array on serial monitor
    Index = 3 ; // reset button to 3
    }
}

```

Chapter Eight

Advanced Strategies for Arduino

Speeding Up the Input and Output Access

On each AVR Arduino panel, the clock speed is going to be sixteen MHz. Every instruction that is located on the controller is going to be completed in under four cycles; therefore, you are going to be able to do four million instructions in a single second. But, the input-output speed is going to be a lot slower. This happens due to the digital write and digital read functions. Each time that a function is executed, there is going to be extra code that has to be executed as well. The additional code is going to end up being responsible for detecting and mapping out the digital pins to the output port. On every microcontroller, the input-output device will be allocated in a group of eight pins to a port which is going to be specially registered to the controller.

Therefore, the next question is going to be, how slow is it? You will figure this out by using this code.

Syntax

```
void setup(){
    pinMode(12, OUTPUT);
}
void loop(){
    digitalWrite(12, LOW);
    digitalWrite(12, HIGH);
}
```

You will be able to improve the program by wrapping the loop code into an infinite loop. This is going to speed up the execution since the CPU is going to be doing less of a function call. Your more up to date circuit will be

Syntax

```
void setup(){
    pinMode(13, OUTPUT);
}
void loop(){
    while (1){
```

```

        digitalWrite(13, LOW);
        digitalWrite(13, HIGH);
        //required if you do serial communication
        if (serialEventRun) serialEventRun();
    }
}

```

This should speed up your input and output so that it is up to ten percent faster. If you want to top the output speed, you are going to have to use a low-level access port and to do that, you are going to have to learn how to use the microcontroller; however, it is going to take away from the beauty of the Arduino machine. However, there is going to be a library that you can use to increase the input and output access.

Keep in mind that there are going to be limitations because it is mainly going to support the Leonardo, Mega, Uno, Nana, and Attiny models.

Syntax

```

void setup(){
    pinMode(13, OUTPUT);
}

void loop(){
    while (1){
        //WriteD13 writes Pin 13. D means digital.
        //to write to pin 8 use WriteD8 function.
        WriteD13(HIGH);
        WriteD13(LOW);
        //required if you do serial communication
        if (serialEventRun) serialEventRun();
    }
}

```

This is going to provide an inline function for faster input-output access.

Analog Readings

Analog readings on a USB powered Arduino model is going to give you

some unexpected results. This happens because the analog reference voltage will be tied to the logic voltage which is typically going to be five volts, however, if you are powering your panel from a switch power supply or from the USB you are going to be dealing with commotion and voltage drops because of the USB cable. This is possible as long as your controller is running at four point five instead of five volts.

It is not going to be ideal when it comes to precision measurements. To do the proper measurements, you are going to have to use an external precision voltage reference. On the other hand, you can use the voltage reference that is built into the chip to measure the current VCC and do the calculations that have to be done.

Syntax

```
int GetVccMiliVolts(){
    #if defined(ARDUINO_ARCH_AVR)
    const long int scaleConst = 1156.300 * 1000;
    // Read 1.1V reference against Avcc
    #if defined(__AVR_ATmega32U4__) ||
defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
        ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) |
_BV(MUX1);
    #elif defined (__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) ||
defined(__AVR_ATtiny84__)
        ADMUX = _BV(MUX5) | _BV(MUX0);
    #elif defined (__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) ||
defined(__AVR_ATtiny85__)
        ADMUX = _BV(MUX3) | _BV(MUX2);
    #else
        ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
    #endif
    delay(2); // Wait for Vref to settle
    ADCSRA |= _BV(ADSC); // Start conversion
    while(bit_is_set(ADCSRA,ADSC)); // measuring
```

```

uint8_t low = ADCL; // must read ADCL first - it then locks ADCH
uint8_t high = ADCH; // unlocks both
long int result = (high<<8) | low;
result = scaleConst / result;
// Calculate Vcc (in mV); 1125300 = 1.1*1023*1000
return(int)result; // Vcc in millivolts
#else
return -1;
#endif
}

```

Getting Free RAM

In this section, you are going to learn how to get the free RAM returned in bytes.

Syntax

```

unsigned int UnusedRAM(){
    unsigned int byteCounter is equal to (=) 0;
    byte *byteArray;
    while ( (byteArray = (byte*) malloc (byteCounter * sizeof(byte))) !=
NULL ){
        byteCounter++;
        free(byteArray);
    }
    free(byteArray);
    return byteCounter;
}

```

Using the Const, Program and Global Variables

In the event that you are having to store a large amount of data in the code that you are using, then you are going to need to make a global const and include the program statement. This is going to tell the program's compiler to store the data in the program memory instead of on the RAM

Syntax

```
//5120-byte array. Usually, it wouldn't fit into the RAM,  
//but with static storage, it gets stored in the program memory  
const byte big_array[5120] PROGMEM = {0};
```

Forgetting the Default IDE

While the IDE that comes with Arduino is great, it is not going to be suitable for use with professional software development. If you are using it, you are going to realize that it is going to become a pain to use once you have written a program that has more than a few hundred lines.

So, to bypass this, you will use the Atmel Studio or the Visual Studio that comes with the Visual Micro plugin. This plugin is going to have a debugger, but it is vital that you know that it is not free, it is going to cost you seventeen dollars.

You can also use a text editor so that you can write out your programs. However, with Arduino one point five, the IDE is going to support the building and uploading of your program through the command line.

Syntax

```
Arduino --panel (panel type description) --port (serial port) --upload  
(sketch path)
```

```
Arduino --panel (panel type description) --port (serial port) --verify  
(sketch path)
```

You will need to have a panel type description as well.

Syntax

```
package:arch:panel[:parameters]
```

You are going to be able to find out more about the command line through the manual documents for Arduino IDE.

Inlining Small Functions

Inline functions are a good way to speed up a program, especially if you put them in a loop function. By default, the inline keyword is going to suggest the function for inlining. The compiler for the Arduino program is going to decide that it cannot inline in the optimization phase.

There is a manual that says that the inline function is as fast as a macro. But, there is a catch. If you make an inline function, it is going to make your code size larger; especially if you call the function in several places.

To force a function inline, you will use the inline keyword

Syntax

```
__inline void SomeFunction(){  
    Serial.println("I'm an inline function");  
}
```

Chapter Nine

Humidity Sensor

You may discover a time that you want to learn how to interface your panel with the various sensors that are around you. The DHT-22 is going to be the digital output that you are going to tie to find an accurate measure of temperature and humidity. This is done by using the capacitive humidity sensor as well as the thermostat. It is capable of using its sensors to determine if the air surrounding the device contains any humidity and then will send out a digital beacon to the data pin.

The connections for your humidity sensor are going to be relatively straightforward. The first pin is going to have three to five volts of power and the second pin is going to be the data input pin. The last pin is going to be the ground pin.

Technical Details

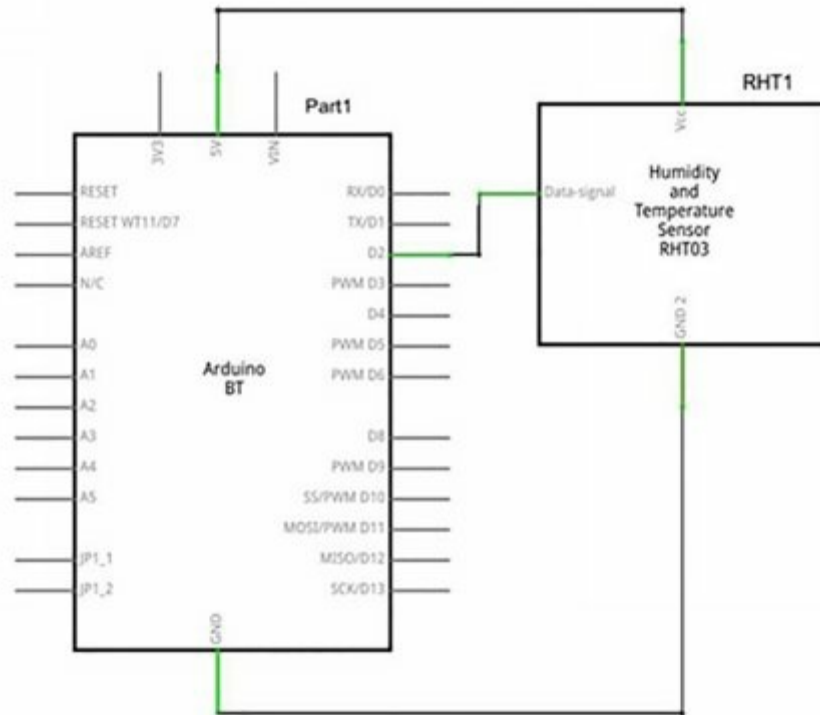
1. Temperature: forty to eighty degrees Celcius. It is going to be off by half a degree either way.
2. Power: three to five volts
3. Humidity: zero to a hundred percent with a two to five percent accuracy
4. Max current: two point five milliamps.

Components

1. Breadpanel (1)
2. 10K ohm resistor (1)
3. Arduino Uno R3 (1)
4. DHT22 (1)

Procedure

You will follow this diagram and hook up your components that you obtained on the breadpanel as you are going to see below.



Sketch

Now you are going to open the Arduino IDE software on your PC. Coding with Arduino is going to control your circuit. You will have to make sure you are opening up a more up to date sketch file; you can do this by clicking on the more up to date button.

Arduino code

You will need to put this code into the Arduino program so that you can process the sensor correctly.

```
#include "dht. H"

#define DHTPIN 2 // the digital pin that the sensor is connected to.
// uncomment the type that you are using
// #define DHTTYPE DHT11 // DHT 11
#define DHTTYPE DHT 22 // DHT 21 (AM2303) , AM2321
// #define DHTTYPE DH 21 // DHT 21 (AM 2301)
// connect pin 1 (on the left) of the sensor to the five volts
// note: if you are using a panel that is three point three volts, then you
are going to need to use pin 1
```

```

// connect pin 2 to the sensor that you are using
// connect pin 3 to the ground on the sensor.
// plug in a 10k resistor from pin 2 to pin 1 of the sensor
// initialize the DHT sensors
// note that the older versions of this library are going to have a third
parameter that is optional
// you will need to tweak the timing for a faster processor. That will
mean that the parameter is no longer going to be needed.
// the current DHT reading algorithm will adjust itself to work with the
faster processors.
DHT dht (DHTPIN, DHTTYPE) ;
Void setup () {
Serial begin (9600) ;
Serial print in ("DHTxx test!" ) ;
Dht begin () ;
    }
Void loop () {
Delay (2000) ; // you will need to wait a few seconds between
measurements.
Float a = dht read humidity () ;
// reading temperature or humidity will take around two hundred and
fifty milliseconds
Float b = dht read temperature () ;
    // read temperature as Celsius
Float c = dht read teperature (true) ;
// read temperature at Fahrenheit
// check if any reads failed and exited early
If (isnan (a) || isnan (b) || isnan (c) {
    Serial print in ("couldn't read from DHT sensor!") ;
Return ;
}

```

Code Notes

There are going to be four terminals that are connected to the panel.

1. The 10 k ohm resistor is going to go between the data and Vcc pin
2. Data pin to Arduino pin digital representation two
3. GND pin to the ground of the Arduino panel
4. Vcc pin to five volts of Arduino panel

At the point in time that the hardware connections are completed, you are going to have to add the DHT22 library to the Arduino library file.

The result is going to be that you will see the temperature and humidity display on a serial port monitor and it is going to be updated every two seconds.

Chapter Ten

Temperature Sensor

The temperature sensor is going to be a precision integrated circuit of temperature pieces of equipment that are going to contain an output voltage linearly proportional to the Centigrade temperature.

The LM35 device is going to have advantages over the linear temperature sensors that are calibrated to Kelvin so that the user is not required to deduct an extensive and continuous voltage from the output to get the Centigrade scaling. This device will not demand any additional calibration from outside sources or trim with a view to obtaining maximum accuracy. Most accuracies are going to be a fourth off either direction.

Technical Specifications

1. Suitable for applications that are remote
2. Will be calibrated for Celsius
3. Is going to be rated for temperatures between -55 and 150.
4. Linear to a + 10 -mV/ degree Celsius scale
5. Half a degree ensured accuracy.

Components

1. LM35 sensor (1)
2. Breadpanel (1)
3. Arduino Uno R3 (1)

Code

This is the code that you are going to place in the Arduino sketch program after you have created a more up to date file.

```
Float temp ;
```

```
Int temp pin = 1 ;
```

```
Void setup () {
```

```
Serial begin (8999) ;
```

```
Void loop () {
```

```
Temp = analog read (temp pin) ;  
// read analog volt from sensor and save to variable temp  
Temp = temp * 3.939403021 ;  
// convert the analog volt to its tempt equivalent  
Serial print (temperature = ) ;  
Serial print (temp) ; // show temperature value  
Serial print ( F ) ;  
Serial print in ( ) ;  
Delay ( 300 ) ; // update sensor reading each one second  
}
```

Notes on the code

1. Connect the GND to the GND that is located on the Arduino
2. Make sure that you connect the positive Vs. to the positive 5V that you can find on your panel.
3. Connect the V out to the analog 0 that is on the Arduino board.

The result that you are going to see is the temperature displayed on the serial port monitor which is updated each second.

Chapter Eleven

Water Detector and Sensor

The water sensor brick is going to be designed for detecting water. This is going to be most often used to sense predetermined markers such as any leaks or even water levels.

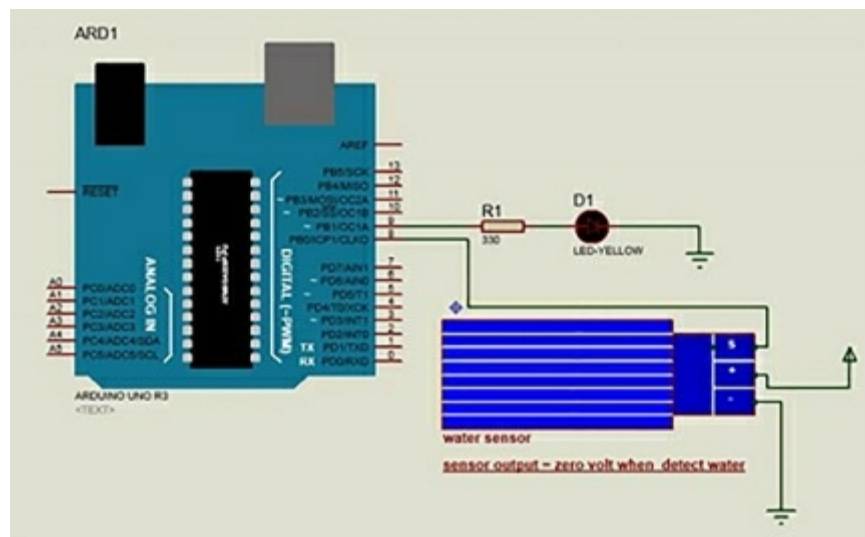
When you are choosing to connect the water sensor to the Arduino panels, the sensor is going to be able to detect any measurement of water. It can accurately measure the level, presence, volume, or even if there is a complete lack of water. This is a great feature and even has the ability to remind the owner when it is time to water household plants!

You will be able to connect the water sensor to the eighth pin on the Arduino board. this pin is going to enlist an LED that enables us to identify when the sensor has come been exposed to water of any level.

Components

1. 330 ohm resistor (1)
2. Breadpanel (1)
3. Led (1)
4. Arduino Uno R3 (1)
5. Water sensor (1)

Procedure



Code

```
# define grove water sensor 9 // attach water sensor to Arduino digital
pin 8
#define LED 9 // attach an LED to pin 9
Void setup () {
    Pin mode (grove water sensor, input) ; // the water sensor will be an
input
    Pin mode (led, output) ; // the led will be an output
}
```

Notes on the code

There are going to be three terminals for the water sensor, and it is going to connect as follows.

1. The LED will be connected to the digital pin 9
2. The positive Vs. will be tied to the positive side of the five volts.
3. The GND will be connected to the GND on the panel.
4. The S will be attached to the eight pin on the Arduino board.

At the point in time that the sensor picks up any water, it is going to cause pin eight to become weak, and the LED to be turned on.

Chapter Twelve

PIR Sensor

A PIR sensor is going to allow you pick up motions, in other words, it is going to be a motion sensor. The sensor will detect movement inside of the sensors range. You are commonly going to find these sensors outside of homes and businesses. PIR is going to stand for passive infrared, pyroelectric, or IR motion sensor.

Here are some of the advantages of having a PIR sensor.

1. Does not wear out
2. Small
3. Easy to use
4. Has an extensive lens range
5. Low power
6. Easy to interface
7. Low power
8. Inexpensive

A PIR is going to be made of pyroelectric sensors that are going to be located in a round metal can that contains a rectangular crystal in the center. This sensor is going to be able to detect varying levels of infrared radiation. Everything is going to emit some level of radiation. However, the hotter something is, the more radiation it is going to emit. The motion sensor is going to detect the change on the move and not just IR levels. The two halves of the panel are going to snap together which is going to cancel each other out. But, if one half gets more or less IR than the other, the output will end up being high or low.

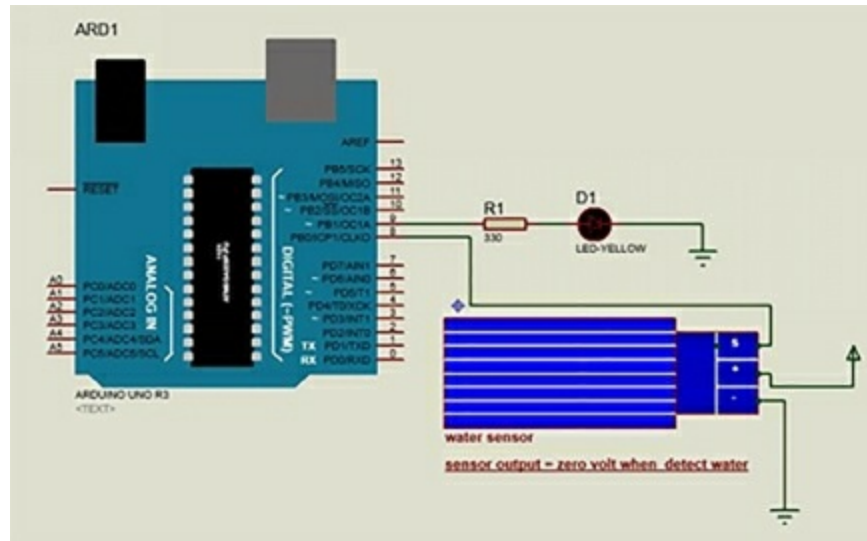
A PIR is going to be adjustable in the settings while containing a header that is installed on the third pin.

For a lot of the basic projects that you can make to detect a human presence is going to use a PIR sensor. Keep in mind that a PIR is not going to tell you how many people are there or how close to the sensor they are. The lens is going to be fixed in such a way that it can sweep a distance.

Components

1. PIR sensor MQ3 (1)
2. Breadpanel (1)
3. Arduino Uno R3 (1)

Procedure



Code

```
#define pir pin 2
Int calibration time = 30 ;
Long unsigned int low in ;
Long unsigned int pause = 4000 ;
Boolean lock low = true ;
Boolean take low time ;
Int pir value = 0 ;

Void setup () {
  Serial begin ( 9600 ) ;
  Pin mode (pir pin, input) ;
}

Void look () {
  Pir sensor () ;
```



```

    }
    Void pir sensor () {
    if (digital read (pir pin ) == high ) {
    if (lock low) {
        pir value = 1;
        lock low = false ;
        serial print in (motion detected) ;
        delay (60) ;
    }
    Take low time = true ;
        }
        If (digital read (pir pin) == low {
        If (take low time ) {
        Low in = millis() ; take low time = false ;
        }
    If (!locklow && millis () – lowin > pause) {
        Pirvalue = 0 ;
        Locklow = true ;
        Serial print in (motion ended) ;
        Delay (60)
        }
    }

```

Notes on code

There are three terminals for the pir.

1. The GND will be connected to the GND
2. The positive VCC will be connected to the positive side of the five volts
3. The out will be tied to the second pin on the panel.

You are going to be able to adjust the sensor sensitivity and delay time with two variable resistors that you can find at the bottom of the panel. After the sensor detects motion, the Arduino is going to send you a message to inform

you that it has detected motion. The pir sensor is going to be delayed for some time to check if there is any more up to date motion. In the event that there is no more up to date motion detected, the panel will send you a message telling you so.

Chapter Thirteen

Ultrasonic Sensor

This sensor is going to be able to determine how far an object is by using a system that is similar to the system that bats use. It is going to give you a great no contact range detection that is highly accurate and going to produce stable readings.

This operation is not going to be affected by things like sunlight or black material, but soft materials such as cloth are going to be somewhat difficult for the sensor to pick up. It is going to be complete with an ultrasonic transmitter and receiver module.

Tech Specifications

1. Measuring angle: thirty degrees
2. Power supply: five volts
3. Resolution: point three cm
4. Quiescent current: greater than two milliamps.
5. Ranging distance: two to four hundred cm
6. Working current: fifteen milliamps
7. Effectual angel: greater than fifteen degrees

Components

1. Arduino Uno R3 (1)
2. Breadpanel (1)
3. Ultrasonic sensor (1)

Code

```
Const int ping pin = 7 ; // trigger pin with sensor
Const in echo pin = 6 ; /// echo the pin with the sensor
Void setup () {
Serial begin (9600) ; // start serial terminal
}
Void loop () {
```

```

    Long duration, inches, cm ;
    Pin mode (ping pin, output) ;
    Digital write (ping pin, low) ;
    Delay microseconds (3) ;
    Digital write (ping pin, high( ;
    Delay microseconds (4) ;
    Digital write (ping pin, low) ;
    Pin mode (echo pin, input) ;
    Duration = pulse in (echo pin,high ) ;
    Inches = microseconds to inches (duration) ;
    Cm = microseconds to centimeters (duration) ;
    Serial print (inches) ;
    Serial print ( in, ) ;
    Serial print (cm) ;
    Serial print (cm ) ;
    Serial print in () ;
    Delay (499) ;
}

Long microseconds to inches (long microseconds) {
    Return microseconds / 49 / 2 ;
}

Long microseconds to centimeters (long microseconds) {
    Return microseconds / 49 / 2 ;
}

```

Code notes

1. Connect the GND to the GND
2. The positive five-volt pin to the positive five volt
3. The echo to pin six
4. The trigger to pin seven

Result

You are going to see the distance measured by your sensor in inches and centimeters.

Chapter Fourteen

Connecting Switch

The push buttons or switches are going to connect two terminals that are open inside of a circuit.

Pull Down Resistor

The resistor is going to be used when you are working with electronic logic circuits so that it can make sure that the inputs that are on the Arduino panel are settled at the logic levels that are expected. In the event that there are external pieces of equipment that are not connected or are at a high impedance, then nothing is going to be attached to the input pin. This does not mean that it is a logical zero. The pull-down resistor is going to be connected to the ground and the proper pin on the device.

One example would be for the resistor in the circuit to be connected between the supply voltage and the microcontroller pin. In these circuits, the switch is going to be closed while the microcontroller input is high. However, when the switch is open, the resistor is going to pull the input voltage down so that it can prevent an undefined state of the input.

The resistor has to have a massive resistance over the impedance of the logic circuit, or it is going to pull the voltage down lower than it is supposed to be which is going to cause the input voltage to remain at a low value no matter where the switch is.

Components

1. LED (1)
2. Arduino Uno panel (1)
3. 4.7 k ohm resistor (1)
4. 330 ohm resistor (1)

Code

```
// constants are not going to change they are going to be used to set the  
pin digital representations
```

```
Const int buttons pin = 8 // the digital representation of the push button
```



```

    pin
Const int led pin = 2 ; // the LED pin
    // variable will change
    Int button state = 0 ; // variable for reading the button status
Void setup () {
    // initialize the LD pin as an output
Pin mode (led pin, output) ;
    // initialize push button as an input ;
Pin mode (button pin, input0 ;
    }
Void loop () {
    // read the state of the push button value ;
Button state = digital read (button pin) ;
    // check if the push button has been pushed
    // if yes, then the button state will be high
If (button state == high) {
    // turn LED on ;
Digital write (led pin , high) ;
    } else {
    // turn LED off
Digital write (led pin , low) ;
    }
}

```

Chapter Fifteen

Inputs and Outputs

If you look at the pins on the Arduino panel, you will see that they can be configured as an input or an output. You should remember that a lot of the analog pins have the possibility of needing to be set and used in the same way that a digital pin is going to be used.

Input Pin Configuration

The pins are going to be set to input by default; therefore, they do not need to be declared as an input by using the `pinmode ()` function whenever you want to use them as a contribution. The pins that are configured this way will be in a state of high impedance due to the fact that the input pins are only going to be made to make small demands on the circuit that they sample, which is going to be equal to the series resistor of a hundred megaohm for the front pin.

In other words, it is not going to take much current to switch the input from one state to the next which makes it useful for things like when you need to implement a touch sensor or when you are reading an LED as a photodiode.

The pins that are configured as `pinmode` will have nothing that is connected to them, if wires are connected to them, then they cannot be attached to another circuit. It has been reported that there are changes in the pins state, environmental commotion picked up through the pins or the capacitive coupling in the situation of a pin that is near the first pin.

Pull Up Resistors

The pull-up resistor is going to be useful when you need to steer the input of a pin to a known state, but you do not have any data present. This is going to be best when there is no input. All you need to do is add a pull-up resistor that goes up to 5 volts, or you can choose a pull-down resistor for the input. It is best that you use a 10k resistor that is going to be good for pulling up or pulling down the resistor.

Using the Built-In Pull-Up Resistor with the Pins Configured to Input

In the Atmega chip, there are around 20,000 pull-up resistors that are built into it that you are going to have access to in the software. These resistors are going to be accessed through your `pinmod ()` setting by inputting

input_pullup. Now you have inverted the behavior of your contribution mode so that high will turn the sensor off and low will turn the sensor on.

The various values for the pull-up are going to depend on what kind of microcontroller you are using. For many AVR panels, the value is going to be between 20 and 50 k ohms. For the Arduino Due, you will find that it is between 50k and 150 k ohms. To figure out what the exact value is, you will need to look at the datasheet for the microcontroller that is installed on your panel.

Whenever you connect sensors to the pins that are configured for input, you need to ensure that the other end is grounded. This is done so that if the pin is reading high, the switch is going to be opened and little means that the switch is pressed. With pull-up resistors, you are going to be able to provide enough current to light up the LED that is connected to the pin.

There are some registers that are going to tell the pin if it is on a high or low while controlling the pull-up resistor. There are also pins that can be configured to have the pull-up resistor turned on whenever the pin is in input mode, which will mean that the pin is turned on to high. Should the pin be switched over to output by use of the pinMode () function, then it is going to work the opposite direction. So, if the pin is on output mode, the high state is going to have the resistor set up to where if switched it will go into input mode.

Example

```
pinmode (4, input) ; // the pin is set to input  
pinmode (6, input_pullup) ;
```

Pins Configured To Output

Any pin capable of configuration will do so to output with pinMode () and will be the lower state of impedance. With that being said, they are going to be able to provide a large amount of current to other circuits that are hooked up to it. The Atmega pins are going to give you the positive current or the negative current depending on how many milliamps of current the other pieces of equipment are going to need. As long as it is 40 mA or under, you will be able to have enough current to light up an LED brightly, but it is not going to be enough to run any motors.

Whenever you attempt to run a device that is going to require a lot of currents, the pins can become damaged or even destroyed. This can end up destroying the entire Atmega chip which is going to result in a dead pin in your microcontroller. However, the other pins are going to work still, but they may work at a lesser power than they were before. That is why you should hook your output pins up to another device that is either 470 ohms or is a 1k resistor. The only time that you should not is if the current draw that is coming from the pins is required to run a certain application.

Pinmode () function

Pin mode is going to be used whenever you are configuring a specific pin so that it is going to behave as an input or an output pin. There is the possibility that you can enable the internal pull-up resistor through the input_pullup mode. It also makes it to where the input mode is going to disable any internal pull-ups.

Syntax:

```
Void setup ( ) {  
  Pinmode (pin, mode) ;  
}
```

3. Pin is going to be the digital representation of pins that you want to set the mode for.
4. Mode will be either input, output, or input_pullup.

Digitalwrite () function

This is a super function that you can use when you need to write the code containing higher or lower values for the personal digital pin previously setup. Should the pin be configured for output, then the voltage needs to be set to the value of 5 volts. There cannot be any volts for low because it will need to be grounded.

If the pin is on input, then the high setting is going to be disabled while the low is going to be the pullup internally for the pin. We strongly advise that you setup your personal pinmode () function so that input_pullup. Otherwise, it might not be able to pull up the resistor interior of the panel.

If you do not set the pin mode for output and then proceed to connect an LED

tot hat pin as you call on the high setting, then the LED is going to appear dimmer than it should be. If you do not explicitly set the pin mode and digital write functions to enable the internal pullup, it is going to do so automatically which is going to act as a massive current capable of limiting the resistor.

Syntax

```
Void loop ( ) {  
    digitalWrite (pin, value) ;  
}
```

3. Pin is going to be the digital representation of pins that you want to set to input or output.
4. Value is the high or low setting.

Analogread () function

The Arduino program is capable of detection of all levels. Here, it can determine whether or not there is voltage, perhaps inadvertently, finding itself applied to one of the pins before it sends the report back through the digitalread () function. You have to know that there is a difference between the on and off sensor so that the analog sensor is constantly charging. To read this type of sensor, you are going to require a different type of pin.

When you look at the lower right of your panel, there are going to be six pins that are marked as analog in. These pins are not just going to tell if there is any voltage being applied to them, but also how much is flowing through it. When you use the analogread () function, you will be able to read the voltage that is applied to just one of the pins.

For this function, a digital representation is going to be returned between 0 and 1023 to represent the voltage between 0 and 5 volts. An example would be if you have a voltage of 2.5 V that is being applied to the pin digital representation 3, you will get a return of 512.

Syntax

Analogread (pin) ;

2. Pin: the digital representation of which pin to be read from 0 to 5 on a great majority of panels. 0 to 7 if you are using the mini and Nano panels. And then 0 to 15 on the Mega panels.

Conclusion

I hope this book was able to help you to understand Arduino on a more advanced level.

The next step is to take what you have learned here and apply it to what you already know. You are going to be able to do distinct functions with an Arduino panel, and it is going to be able to change your life!

If you have ever used a Raspberry Pi panel before, the Arduino panel is going to be similar to it. However, it is going to be different because there are going to be other things that you are going to do with a Pi panel over the Arduino panel. But, that should not discourage you from using the Arduino panel. If you know how to use one panel, you are going to know how to use the other one.

You are going to have to remember that you will have to be patient when it comes to working with the Arduino panel because it is going to contain code and if you mess up one bit of code, then you are going to have to go back and redo what you have already done. But, that is not going to be something that is going to stop you from using the panel, all that is going to do is make it to where you are going to have to be more careful about the code that you are putting into the program.

Finally, if you enjoyed this book, then I'd like to ask you for a favor, would you be kind enough to leave a review for this book on Amazon? It'd be greatly appreciated!

Thank you and good luck!



Table of Contents

ARDUINO

The Ultimate Beginner's Guide to Learn Arduino

Introduction

Chapter One : What is Arduino?

Chapter One : What is Arduino?

Chapter Two : Hardware, Software, and Applications

Chapter Two : Hardware, Software, and Applications

Chapter Three : Data Types Found in Arduino

Chapter Three : Data Types Found in Arduino

Chapter Four : Local and Global Variables

Chapter Four : Local and Global Variables

Chapter Five : Operators Used in Arduino

Chapter Five : Operators Used in Arduino

Chapter Six : Arduino Control Statements

Chapter Six : Arduino Control Statements

Chapter Seven : Arduino Loops

Chapter Seven : Arduino Loops

Chapter Eight: Functions in Arduino and How They Work

Chapter Eight: Functions in Arduino and How They Work

Chapter Nine : Strings in Arduino

Chapter Nine : Strings in Arduino

Chapter Ten : String Objects

Chapter Ten : String Objects

Chapter Eleven : How to Use Time in Arduino

Chapter Eleven : How to Use Time in Arduino

Chapter Twelve : Arduino Arrays

Chapter Twelve : Arduino Arrays

Chapter Thirteen : Input and Output Functions Found in Arduino

Chapter Thirteen : Input and Output Functions Found in Arduino

Chapter Fourteen : How PowerShell is Different from Other Shells

Chapter Fourteen : How PowerShell is Different from Other Shells

Conclusion

ARDUINO

Tips and Tricks to Learn Arduino Quickly and Efficiently

Introduction

Chapter 1 : Arduino UNO

Chapter 1 : Arduino UNO

Chapter 2 : How to Install Libraries

Chapter 2 : How to Install Libraries

Chapter 3 : Tricks for the Bootloader

Chapter 3 : Tricks for the Bootloader

Chapter 4 : Upgrading Arduino to the Latest Version of Atmega328P Chip

Chapter 4 : Upgrading Arduino to the Latest Version of Atmega328P Chip

Chapter 5 : Conversion to 3.3V Devices

Chapter 5 : Conversion to 3.3V Devices

Chapter 6 : Tricks for Maximizing Arduino

Chapter 6 : Tricks for Maximizing Arduino

Chapter 7 : Arduino Module: Tricks and Tips

Chapter 7 : Arduino Module: Tricks and Tips

Chapter 8 : ArduinoISP

Chapter 8 : ArduinoISP

Summary

Conclusion

ARDUINO

Simple and Effective Strategies to Learn Arduino Programming

Introduction

Chapter 1 : Introduction to Arduino

Chapter 1 : Introduction to Arduino

Chapter 2 : Light the World

Chapter 2 : Light the World

Chapter 3 : Programming

Chapter 3 : Programming

Chapter 4 : Graphics

Chapter 4 : Graphics

Chapter 5 : References in Arduino

Chapter 5 : References in Arduino

Conclusion

ARDUINO

[Best Practices to Learn and Execute Arduino Programming](#)

[Introduction](#)

[Chapter 1 : Learning What Arduino Is](#)

[Chapter 1 : Learning What Arduino Is](#)

[Chapter 2 : Hardware, Software, and Applications](#)

[Chapter 2 : Hardware, Software, and Applications](#)

[Chapter 3 : The Data Types You Will Find in Arduino](#)

[Chapter 3 : The Data Types You Will Find in Arduino](#)

[Chapter 4 : Constants – Local and Global](#)

[Chapter 4 : Constants – Local and Global](#)

[Chapter 5 : Operators](#)

[Chapter 5 : Operators](#)

[Chapter 6 : Control Declarations](#)

[Chapter 6 : Control Declarations](#)

[Chapter 7 : Loops in Arduino](#)

[Chapter 7 : Loops in Arduino](#)

[Chapter 8 : Arduino Functions](#)

[Chapter 8 : Arduino Functions](#)

[Chapter 9 : Arduino Strings](#)

[Chapter 9 : Arduino Strings](#)

[Chapter 10 : String Objects for Arduino](#)

[Chapter 10 : String Objects for Arduino](#)

[Chapter 11 : Time with Arduino](#)

[Chapter 11 : Time with Arduino](#)

[Chapter 12 : Arrays with Arduino](#)

[Chapter 12 : Arrays with Arduino](#)

[Chapter 13 : Sensors](#)

[Chapter 13 : Sensors](#)

[Chapter 14 : The Best Practices for Arduino](#)

[Chapter 14 : The Best Practices for Arduino](#)

[Conclusion](#)

[ARDUINO](#)

[Advanced Strategies to Learn and Execute Arduino Programming](#)

[Introduction](#)

[Chapter One : The Arduino Due and the Arduino Zero](#)

[Chapter One : The Arduino Due and the Arduino Zero](#)

[Chapter Two : The Pulse Width Modulation](#)
[Chapter Two : The Pulse Width Modulation](#)
[Chapter Three : Calculated Digital Representations](#)
[Chapter Three : Calculated Digital Representations](#)
[Chapter Four : Interrupts](#)
[Chapter Four : Interrupts](#)
[Chapter Five : Communication](#)
[Chapter Five : Communication](#)
[Chapter Six : Inter-Integrated Circuit](#)
[Chapter Six : Inter-Integrated Circuit](#)
[Chapter Seven : Serial Peripheral Interface](#)
[Chapter Seven : Serial Peripheral Interface](#)
[Chapter Eight : Advanced Strategies for Arduino](#)
[Chapter Eight : Advanced Strategies for Arduino](#)
[Chapter Nine : Humidity Sensor](#)
[Chapter Nine : Humidity Sensor](#)
[Chapter Ten : Temperature Sensor](#)
[Chapter Ten : Temperature Sensor](#)
[Chapter Eleven : Water Detector and Sensor](#)
[Chapter Eleven : Water Detector and Sensor](#)
[Chapter Twelve : PIR Sensor](#)
[Chapter Twelve : PIR Sensor](#)
[Chapter Thirteen : Ultrasonic Sensor](#)
[Chapter Thirteen : Ultrasonic Sensor](#)
[Chapter Fourteen : Connecting Switch](#)
[Chapter Fourteen : Connecting Switch](#)
[Chapter Fifteen : Inputs and Outputs](#)
[Chapter Fifteen : Inputs and Outputs](#)
[Conclusion](#)