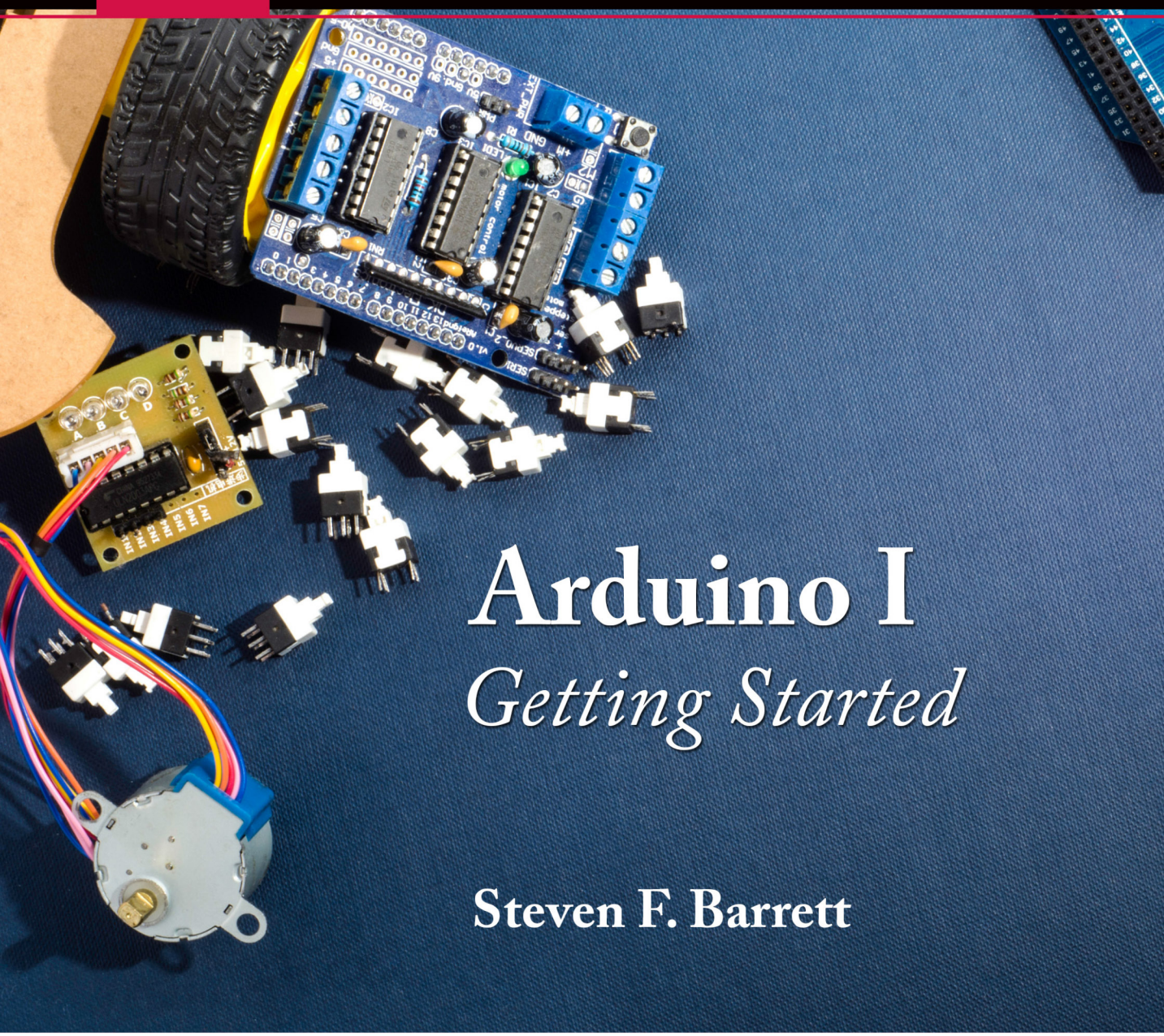
MORGAN & CLAYPOOL PUBLISHERS



Arduino I

Getting Started

Steven F. Barrett

SYNTHESIS LECTURES ON DIGITAL CIRCUITS AND SYSTEMS

Mitchell A. Thornton, *Series Editor*

Arduino I

Getting Started

Synthesis Lectures on Digital Circuits and Systems

Editor
Mitchell A. Thornton, ***Southern Methodist University***

The *Synthesis Lectures on Digital Circuits and Systems* series is comprised of 50- to 100-page books targeted for audience members with a wide-ranging background. The Lectures include topics that are of interest to students, professionals, and researchers in the area of design and analysis of digital circuits and systems. Each Lecture is self-contained and focuses on the background information required to understand the subject matter and practical case studies that illustrate applications. The format of a Lecture is structured such that each will be devoted to a specific topic in digital circuits and systems rather than a larger overview of several topics such as that found in a comprehensive handbook. The Lectures cover both well-established areas as well as newly developed or emerging material in digital circuits and systems design and analysis.

Arduino I: Getting Started
Steven F. Barrett
2020

Index Generation Functions
Tsutomu Sasao
2019

Microchip AVR® Microcontroller Primer: Programming and Interfacing, Third Edition
Steven F. Barrett and Daniel J. Pack
2019

Microcontroller Programming and Interfacing with Texas Instruments MSP430FR2433 and MSP430FR5994 – Part II, Second Edition
Steven F. Barrett and Daniel J. Pack
2019

Microcontroller Programming and Interfacing with Texas Instruments MSP430FR2433 and MSP430FR5994 – Part I, Second Edition
Steven F. Barrett and Daniel J. Pack
2019

Synthesis of Quantum Circuits vs. Synthesis of Classical Reversible Circuits
Alexis De Vos, Stijn De Baerdemacker, and Yvan Van Rentergen
2018

Boolean Differential Calculus
Bernd Steinbach and Christian Posthoff
2017

Embedded Systems Design with Texas Instruments MSP432 32-bit Processor
Dung Dang, Daniel J. Pack, and Steven F. Barrett
2016

Fundamentals of Electronics: Book 4 Oscillators and Advanced Electronics Topics
Thomas F. Schubert and Ernest M. Kim
2016

Fundamentals of Electronics: Book 3 Active Filters and Amplifier Frequency
Thomas F. Schubert and Ernest M. Kim
2016

Bad to the Bone: Crafting Electronic Systems with BeagleBone and BeagleBone Black, Second Edition
Steven F. Barrett and Jason Kridner
2015

Fundamentals of Electronics: Book 2 Amplifiers: Analysis and Design
Thomas F. Schubert and Ernest M. Kim
2015

Fundamentals of Electronics: Book 1 Electronic Devices and Circuit Applications
Thomas F. Schubert and Ernest M. Kim
2015

Applications of Zero-Suppressed Decision Diagrams
Tsutomu Sasao and Jon T. Butler
2014

Modeling Digital Switching Circuits with Linear Algebra
Mitchell A. Thornton
2014

Arduino Microcontroller Processing for Everyone! Third Edition
Steven F. Barrett
2013

Boolean Differential Equations
Bernd Steinbach and Christian Posthoff
2013

Bad to the Bone: Crafting Electronic Systems with BeagleBone and BeagleBone Black
Steven F. Barrett and Jason Kridner
2013

Introduction to Noise-Resilient Computing
S.N. Yanushkevich, S. Kasai, G. Tangim, A.H. Tran, T. Mohamed, and V.P. Shmerko
2013

Atmel AVR Microcontroller Primer: Programming and Interfacing, Second Edition
Steven F. Barrett and Daniel J. Pack
2012

Representation of Multiple-Valued Logic Functions
Radomir S. Stankovic, Jaakko T. Astola, and Claudio Moraga
2012

Arduino Microcontroller: Processing for Everyone! Second Edition
Steven F. Barrett
2012

Advanced Circuit Simulation Using Multisim Workbench
David Báez-López, Félix E. Guerrero-Castro, and Ofelia Delfina Cervantes-Villagómez
2012

Circuit Analysis with Multisim
David Báez-López and Félix E. Guerrero-Castro
2011

Microcontroller Programming and Interfacing Texas Instruments MSP430, Part I
Steven F. Barrett and Daniel J. Pack
2011

Microcontroller Programming and Interfacing Texas Instruments MSP430, Part II
Steven F. Barrett and Daniel J. Pack
2011

Pragmatic Electrical Engineering: Systems and Instruments
William Eccles
2011

Pragmatic Electrical Engineering: Fundamentals
William Eccles
2011

Introduction to Embedded Systems: Using ANSI C and the Arduino Development Environment
David J. Russell
2010

Arduino Microcontroller: Processing for Everyone! Part II
Steven F. Barrett
2010

Arduino Microcontroller Processing for Everyone! Part I
Steven F. Barrett
2010

Digital System Verification: A Combined Formal Methods and Simulation Framework
Lun Li and Mitchell A. Thornton
2010

Progress in Applications of Boolean Functions
Tsutomu Sasao and Jon T. Butler
2009

Embedded Systems Design with the Atmel AVR Microcontroller: Part II
Steven F. Barrett
2009

Embedded Systems Design with the Atmel AVR Microcontroller: Part I
Steven F. Barrett
2009

Embedded Systems Interfacing for Engineers using the Freescale HCS08 Microcontroller II: Digital and Analog Hardware Interfacing
Douglas H. Summerville
2009

Designing Asynchronous Circuits using NULL Convention Logic (NCL)
Scott C. Smith and JiaDi
2009

Embedded Systems Interfacing for Engineers using the Freescale HCS08 Microcontroller I: Assembly Language Programming
Douglas H.Summerville
2009

Developing Embedded Software using DaVinci & OMAP Technology
B.I. (Raj) Pawate
2009

Mismatch and Noise in Modern IC Processes
Andrew Marshall
2009

Asynchronous Sequential Machine Design and Analysis: A Comprehensive Development of the Design and Analysis of Clock-Independent State Machines and Systems
Richard F. Tinder
2009

An Introduction to Logic Circuit Testing
Parag K. Lala
2008

Pragmatic Power
William J. Eccles
2008

Multiple Valued Logic: Concepts and Representations
D. Michael Miller and Mitchell A. Thornton
2007

Finite State Machine Datapath Design, Optimization, and Implementation
Justin Davis and Robert Reese
2007

Atmel AVR Microcontroller Primer: Programming and Interfacing
Steven F. Barrett and Daniel J. Pack
2007

Pragmatic Logic
William J. Eccles
2007

PSpice for Filters and Transmission Lines
Paul Tobin
2007

PSpice for Digital Signal Processing
Paul Tobin
2007

PSpice for Analog Communications Engineering
Paul Tobin
2007

PSpice for Digital Communications Engineering
Paul Tobin
2007

PSpice for Circuit Theory and Electronic Devices
Paul Tobin
2007

Pragmatic Circuits: DC and Time Domain
William J. Eccles
2006

Pragmatic Circuits: Frequency Domain
William J. Eccles
2006

Pragmatic Circuits: Signals and Filters
William J. Eccles
2006

High-Speed Digital System Design
Justin Davis
2006

Introduction to Logic Synthesis using Verilog HDL
Robert B.Reese and Mitchell A.Thornton
2006

Microcontrollers Fundamentals for Engineers and Scientists
Steven F. Barrett and Daniel J. Pack
2006

Arduino I: Getting Started

Steven F. Barrett

www.morganclaypool.com

ISBN: 9781681738185 paperback
ISBN: 9781681738192 ebook
ISBN: 9781681738208 hardcover

DOI 10.2200/S01001ED1V01Y202003DCS058

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON DIGITAL CIRCUITS AND SYSTEMS

Lecture #58
Series Editor: Mitchell A. Thornton, *Southern Methodist University*
Series ISSN
Print 1932-3166 Electronic 1932-3174

Arduino I

Getting Started

Steven F. Barrett
University of Wyoming, Laramie, WY

SYNTHESIS LECTURES ON DIGITAL CIRCUITS AND SYSTEMS #58

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

This book is about the Arduino microcontroller and the Arduino concept. The visionary Arduino team of Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis launched a new innovation in microcontroller hardware in 2005, the concept of open-source hardware. Their approach was to openly share details of microcontroller-based hardware design platforms to stimulate the sharing of ideas and promote innovation. This concept has been popular in the software world for many years. In June 2019, Joel Claypool and I met to plan the fourth edition of *Arduino Microcontroller Processing for Everyone!* Our goal has been to provide an accessible book on the rapidly changing world of Arduino for a wide variety of audiences including students of the fine arts, middle and senior high school students, engineering design students, and practicing scientists and engineers. To make the book more accessible to better serve our readers, we decided to change our approach and provide a series of smaller volumes. Each volume is written to a specific audience. This book, *Arduino I: Getting Started* is written for those looking for a quick tutorial on the Arduino environment, platforms, interface techniques, and applications. Arduino II will explore advanced techniques, applications, and systems design. Arduino III will explore Arduino applications in the Internet of Things (IoT). *Arduino I: Getting Started* covers three different Arduino products: the Arduino UNO R3 equipped with the Microchip ATmega328, the Arduino Mega 2560 equipped with the Microchip ATmega2560, and the wearable Arduino LilyPad.

KEYWORDS

Arduino microcontroller, Arduino UNO R3, LilyPad, Arduino Mega 2560, microcontroller interfacing

To Mom and Dad.

– Steven

Contents

Preface

This book is about the Arduino microcontroller and the Arduino concept. The visionary Arduino team of Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis launched a new innovation in microcontroller hardware in 2005: the concept of open-source hardware. Their approach was to openly share details of microcontroller-based hardware design platforms to stimulate the sharing of ideas and promote innovation. This concept has been popular in the software world for many years. In June 2019, Joel Claypool and I met to plan the fourth edition of *Arduino Microcontroller Processing for Everyone!* Our goal has been to provide an accessible book on the rapidly changing world of Arduino for a wide variety of audiences including students of the fine arts, middle and senior high school students, engineering design students, and practicing scientists and engineers. To make the book more accessible to better serve our readers, we decided to change our approach and provide a series of smaller volumes. Each volume is written to a specific audience. This book, *Arduino I: Getting Started* is written for those looking for a quick tutorial on the Arduino environment, platforms, interface techniques, and applications. Arduino II will explore advanced techniques, applications, and systems design. Arduino III will explore Arduino applications in the Internet of Things (IoT). *Arduino I: Getting Started* covers three different Arduino products: the Arduino UNO R3 equipped with the Microchip ATmega328, the Arduino Mega 2560 equipped with the Microchip ATmega2560, and the wearable Arduino LilyPad.

APPROACH OF THE BOOK

Chapters 1 and 2 are intended for novice microcontroller users. Chapter 1 provides an introduction to programming and introduces the Arduino Development Environment and how to program sketches. Chapter 2 provides an introduction to the Arduino concept, a description of the Arduino UNO R3, the Arduino Mega 2560, and the LilyPad development boards. Chapter 3 introduces the extremely important concept of the operating envelope for a microcontroller. The voltage and current electrical parameters for the Arduino microcontrollers are presented and applied to properly interface input and output devices to the Arduino UNO R3, the Arduino Mega 2560, and the LilyPad. Chapter 4 provides several detailed small system examples including a remote weather station and an underwater ROV.

Steven F. Barrett
March 2020

Acknowledgments

A number of people have made this book possible. I would like to thank Massimo Banzi of the Arduino design team for his support and encouragement in writing the first edition of this book. In 2005, Joel Claypool of Morgan & Claypool Publishers, invited Daniel Pack and me to write a book on microcontrollers for his new series titled "Synthesis Lectures on Digital Circuits and Systems." The result was the book *Microcontrollers Fundamentals for Engineers and Scientists.* Since then we have been regular contributors to the series. Our goal has been to provide the fundamental concepts common to all microcontrollers and then apply the concepts to the specific microcontroller under discussion. We believe that once you have mastered these fundamental concepts, they are easily transportable to different processors. As with many other projects, he has provided his publishing expertise to convert our final draft into a finished product. We thank him for his support on this project and many others. He has provided many novice writers the opportunity to become published authors. His vision and expertise in the publishing world made this book possible. We also thank Dr. C.L. Tondo of T&T TechWorks, Inc. and his staff for working their magic to convert our final draft into a beautiful book.

I would also like to thank Sparkfun, Adafruit, DFRobot, Mikroe, and Microchip for their permission to use images of their products and copyrighted material throughout the text. Several Microchip acknowledgments are in order.

- This book contains copyrighted material of Microchip Technology Incorporated replicated with permission. All rights reserved. No further replications may be made without Microchip Technology Inc's prior written consent.
- *Arduino I: Getting Started* is an independent publication and is not affiliated with, nor has it been authorized, sponsored, or otherwise approved by Microchip.

I would like to dedicate this book to my close friend Dr. Daniel Pack, Ph.D., P.E. In 2000, Daniel suggested that we might write a book together on microcontrollers. I had always wanted to write a book but I thought that's what other people did. With Daniel's encouragement we wrote that first book (and several more since then). Daniel is a good father, good son, good husband, brilliant engineer, a work ethic second to none, and a good friend.

To you, good friend, I dedicate this book. I know that we will do many more together. It is hard to believe we have been writing together for 20 years. Finally, I would like to thank my wife and best friend of many years, Cindy.

Steven F. Barrett
March 2020

CHAPTER 1

Getting Started

Objectives: After reading this chapter, the reader should be able to do the following:

- successfully download and execute a simple program using the Arduino Development Environment;
- describe the key features of the Arduino Development Environment;
- list the programming support information available at the Arduino home page; and
- write programs for use on the Arduino UNO R3, Mega 2560 R3, and the LilyPad Arduino processing boards.

1.1 OVERVIEW

Welcome to the world of Arduino! The Arduino concept of open-source hardware was developed by the visionary Arduino team of Massimo Banzi, David Cuartilles, Tom Igoe, Gianluca Martino, and David Mellis in Ivrea, Italy. The team's goal was to develop a line of easy-to-use microcontroller hardware and software such that processing power would be readily available to everyone.

Chapters 1 and 2 provide a tutorial on the Arduino programming environment and some of the Arduino hardware platforms. As you begin your Arduino adventure, you will find yourself going back and forth between Chapters 1 and 2. Chapter 1 concentrates on the Arduino programming environment. To the novice, programming a microcontroller may appear mysterious, complicated, overwhelming, and difficult. When faced with a new task, one often does not know where to start. The goal of this chapter is to provide a tutorial on how to begin programming. We will use a top-down design approach. We begin with the "big picture" of the chapter. We then discuss the Ardunio Development Environment and how it may be used to quickly develop a program (sketch) for the Arduino UNO R3, the Arduino Mega 2560 R3, and the LilyPad Arduino processor boards. Throughout the chapter, we provide examples and also provide pointers to a number of excellent references. To get the most out of this chapter, it is highly recommended to work through each example.

1.2 THE BIG PICTURE

We begin with the big picture of how to program different Arduino development boards, as shown in Figure 1.1. This will help provide an overview of how chapter concepts fit together.

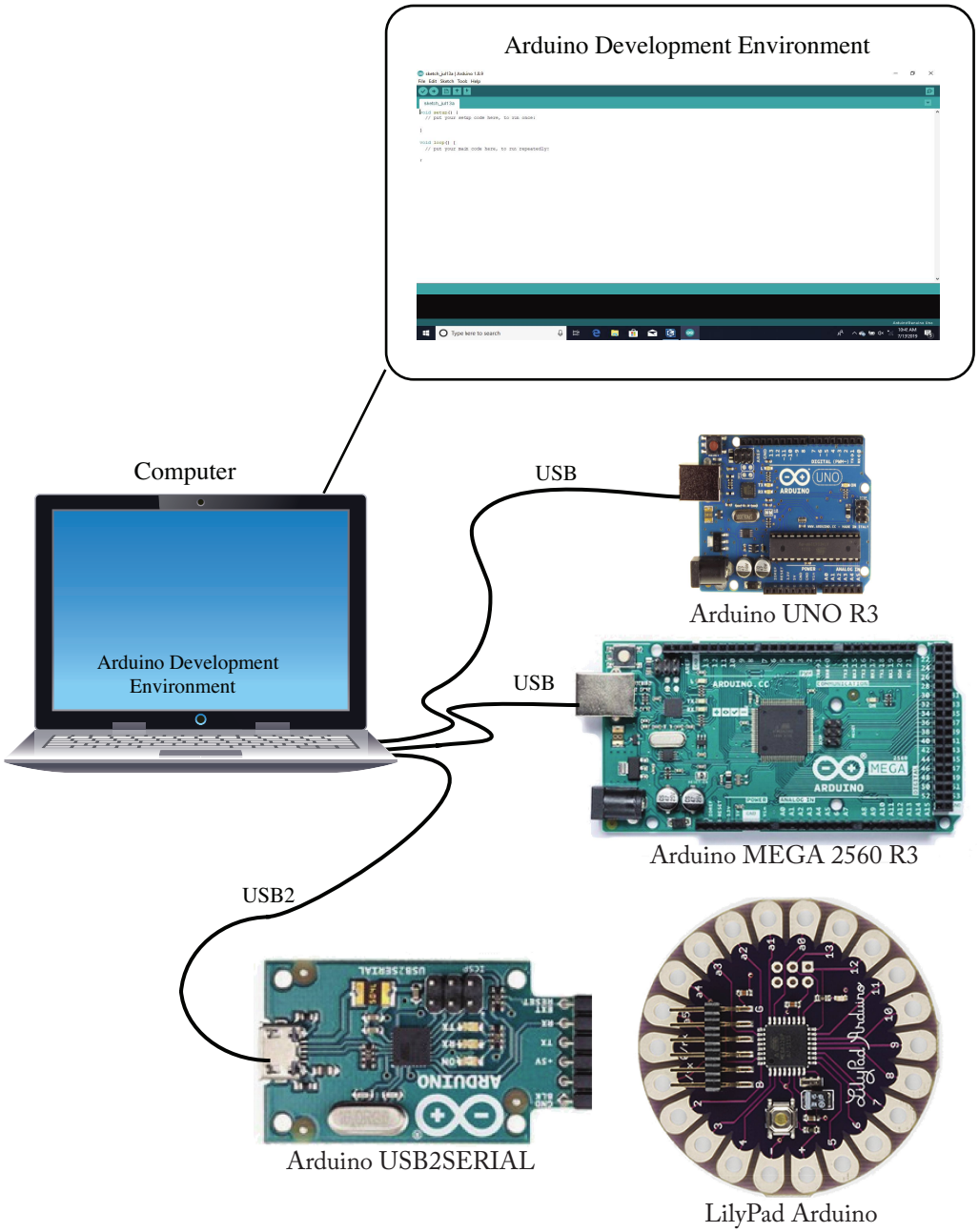


Figure 1.1: Programming the Arduino processor board. (Arduino illustrations used with permission of the Arduino Team (CC BY-NC-SA), www.arduino.cc.)

Most microcontrollers are programmed with some variant of the C programming language. The C programming language provides a nice balance between the programmer's control of the microcontroller hardware and time efficiency in program writing. As an alternative, the Arduino Development Environment (ADE) provides a user-friendly interface to quickly develop a program, transform the program to machine code, and then load the machine code into the Arduino processor in several simple steps. We use the ADE throughout the book.

The first version of the ADE was released in August 2005. It was developed at the Interaction Design Institute in Ivrea, Italy to allow students the ability to quickly put processing power to use in a wide variety of projects. Since that time, updated versions incorporating new features have been released on a regular basis (`www.arduino.cc`).

At its most fundamental level, the ADE is a user-friendly interface to allow one to quickly write, load, and execute code on a microcontroller. A barebones program need only consist of a setup() and loop() function. The ADE adds the other required pieces such as header files and the main program construct. The ADE is written in Java and has its origins in the Processor programming language and the Wiring Project (`www.arduino.cc`).

1.3 ARDUINO QUICKSTART

To get started using an Arduino-based platform, you will need the following hardware and software:

- an Arduino-based hardware processing platform,
- the appropriate interface cable from the host PC or laptop to the Arduino platform,
- an Arduino compatible power supply, and
- the Arduino software.

Interface Cable. The UNO and the MEGA connect to a host PC via a USB cable (Type A male to Type B female). The LilyPad requires a USB 2 serial converter (Arduino USB2SERIAL) and an USB 2 cable (Type A male to mini Type B female).

Power Supply. The Arduino processing boards may be powered from the USB port during project development. However, it is highly recommended that an external power supply be employed. This will allow developing projects beyond the limited electrical current capability of the USB port. For the UNO and the MEGA platforms, Arduino (`www.arduino.cc`) recommends a power supply from 7–12 VDC with a 2.1-mm center positive plug. A power supply of this type is readily available from a number of electronic parts supply companies. For example, the Jameco #133891 power supply is a 9 VDC model rated at 300 mA and equipped with a 2.1-mm center positive plug. It is available for under US$10. Both the UNO and MEGA have onboard voltage regulators that maintain the incoming power supply voltage to a stable 5 VDC.

1.3.1 QUICK START GUIDE

The ADE may be downloaded from the Arduino website's front page at www.arduino.cc. Versions are available for Windows, Mac OS X, and Linux. Provided below is a quick start step-by-step approach to blink an onboard LED.

- Download the ADE from www.arduino.cc.
- Connect the Arduino UNO R3 processing board to the host computer via a USB cable (A male to B male).
- Start the ADE.
- Under the Tools tab select the evaluation **Board** you are using and the **Port** that it is connected to.
- Type the following program.

```
//*****************************************************************

#define LED_PIN 13

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

void loop()
{
digitalWrite(LED_PIN, HIGH);
delay(500);                    //delay specified in ms
digitalWrite(LED_PIN, LOW);
delay(500);
}

//*****************************************************************
```

- Upload and execute the program by asserting the "Upload" (right arrow) button.
- The onboard LED should blink at one second intervals.

With the ADE downloaded and exercised, let's take a closer look at its features.

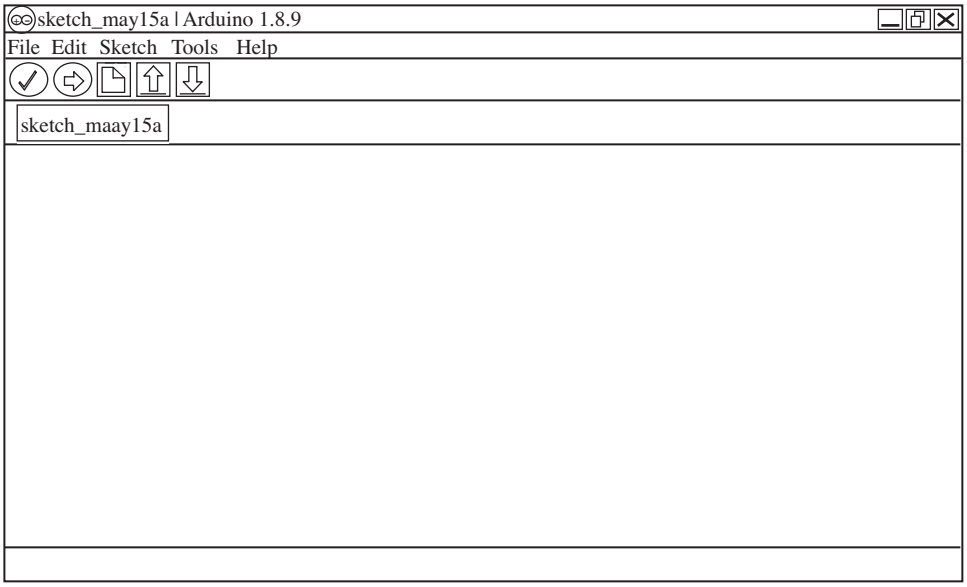


Figure 1.2: Arduino Development Environment (`www.arduino.cc`).

1.3.2 ARDUINO DEVELOPMENT ENVIRONMENT OVERVIEW

The ADE is illustrated in Figure 1.2. The ADE contains a text editor, a message area for displaying status, a text console, a tool bar of common functions, and an extensive menuing system. The ADE also provides a user-friendly interface to the Arduino processor board which allows for a quick upload of code. This is possible because the Arduino processing boards are equipped with a bootloader program.

A close up of the Arduino toolbar is provided in Figure 1.3. The toolbar provides single button access to the more commonly used menu features. Most of the features are self-explanatory. As described in the previous section, the "Upload" button compiles your code and uploads it to the Arduino processing board. The "Serial Monitor" button opens the serial monitor feature. The serial monitor feature allows text data to be sent to and received from the Arduino processing board.

1.3.3 SKETCHBOOK CONCEPT

In keeping with a hardware and software platform for students of the arts, the Arduino environment employs the concept of a sketchbook. An artist maintains their works in progress in a sketchbook. Similarly, we maintain our programs within a sketchbook in the Arduino environment. Furthermore, we refer to individual programs as sketches. An individual sketch within the sketchbook may be accessed via the Sketchbook entry under the file tab.

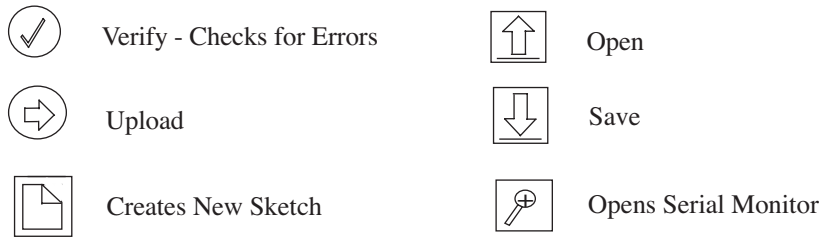


Figure 1.3: Arduino Development Environment buttons.

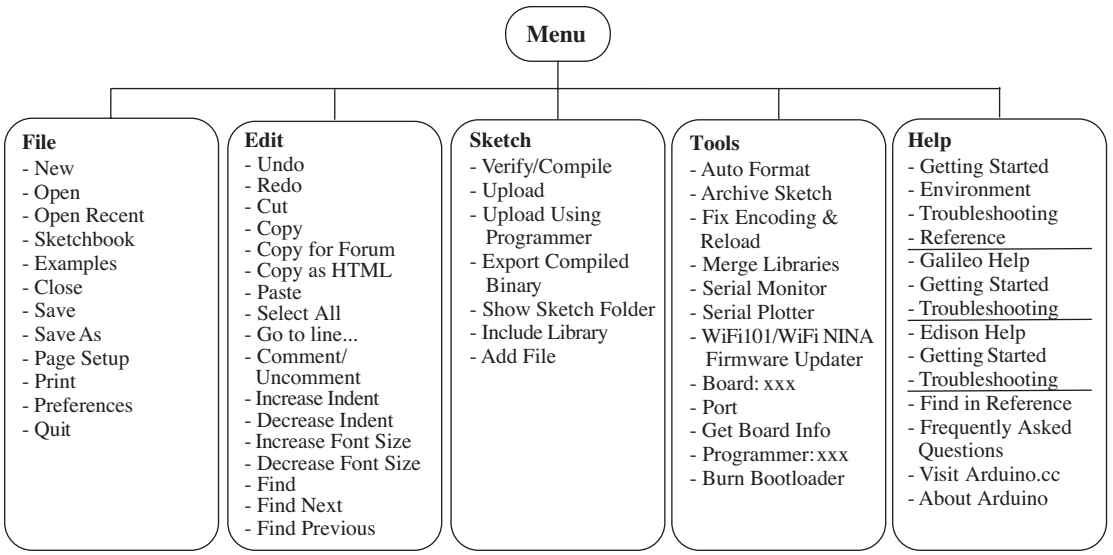


Figure 1.4: Arduino Development Environment menu (`www.arduino.cc`).

1.3.4 ARDUINO SOFTWARE, LIBRARIES, AND LANGUAGE REFERENCES

The ADE has a number of built-in features. Some of the features may be directly accessed via the ADE drop down toolbar illustrated in Figure 1.2. Provided in Figure 1.4 is a handy reference to show the available features. The toolbar provides a wide variety of features to compose, compile, load, and execute a sketch.

1.3.5 WRITING AN ARDUINO SKETCH

The basic format of the Arduino sketch consists of a "setup" and a "loop" function. The setup function is executed once at the beginning of the program. It is used to configure pins, declare variables and constants, etc. The loop function will execute sequentially step-by-step. When

the end of the loop function is reached it will automatically return to the first step of the loop function and execute again. This goes on continuously until the program is stopped.

```
//*************************************************************

void setup()
  {
  //place setup code here
  }

void loop()
  {
  //main code steps are provided here
  :
  :

  }

//*************************************************************
```

Example: Let's revisit the sketch provided earlier in the chapter.

```
//*************************************************************

#define LED_PIN 13                    //name pin 13 LED_PIN

void setup()
{
pinMode(LED_PIN, OUTPUT);             //set pin to output
}

void loop()
{
digitalWrite(LED_PIN, HIGH);          //write pin to logic high
delay(500);                           //delay specified in ms
digitalWrite(LED_PIN, LOW);           //write to logic low
delay(500);                           //delay specified in ms
}

//*************************************************************
```

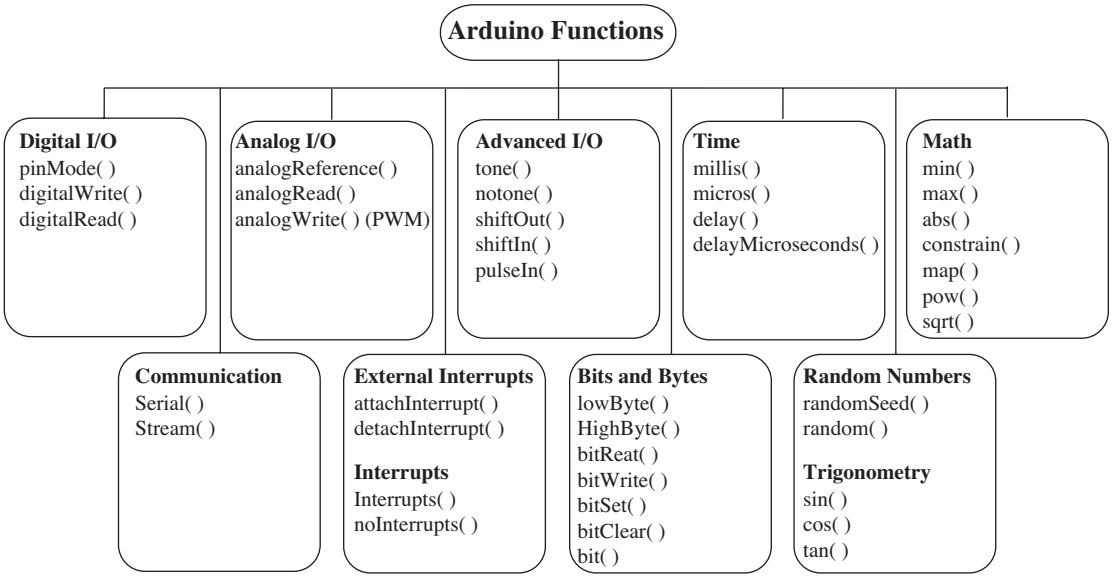


Figure 1.5: Arduino Development Environment functions (www.arduino.cc).

In the first line the #define statement links the designator "LED_PIN" to pin 13 on the Arduino processor board. In the setup function, LED_PIN is designated as an output pin. Recall the setup function is only executed once. The program then enters the loop function that is executed sequentially step-by-step and continuously repeated. In this example, the LED_PIN is first set to logic high to illuminate the LED onboard the Arduino processing board. A 500 ms delay then occurs. The LED_PIN is then set low. A 500-ms delay then occurs. The sequence then repeats.

Even the most complicated sketches follow the basic format of the setup function followed by the loop function. To aid in the development of more complicated sketches, the ADE has many built-in features that may be divided into the areas of structure, variables and functions. The structure and variable features follow rules similar to the C programming language which is discussed in the text *Arduino II: Systems*. The built-in functions consists of a set of pre-defined activities useful to the programmer. These built-in functions are summarized in Figure 1.5.

There are many program examples available to allow the user to quickly construct a sketch. These programs are summarized in Figure 1.6. Complete documentation for these programs is available at the Arduino homepage www.arduino.cc. This documentation is easily accessible via the Help tab on the ADE toolbar. This documentation will not be repeated here. Instead, we refer to these features at appropriate places throughout the remainder of the book. With the Arduino open-source concept, users throughout the world are constantly adding new built-in

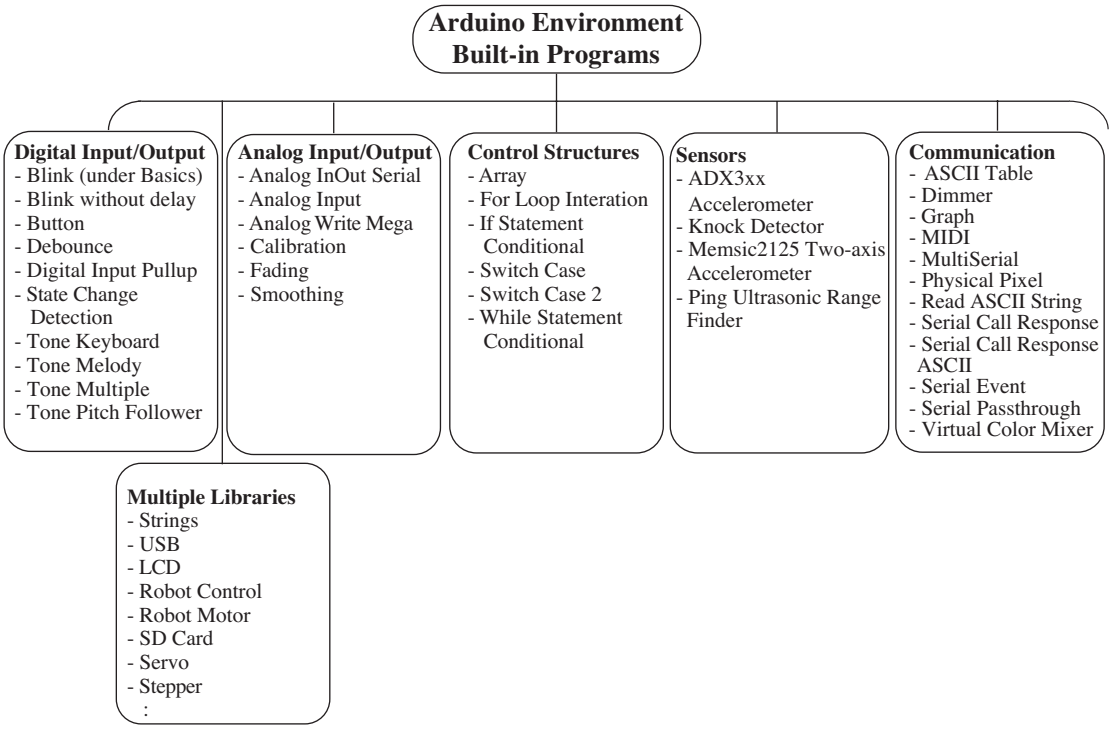


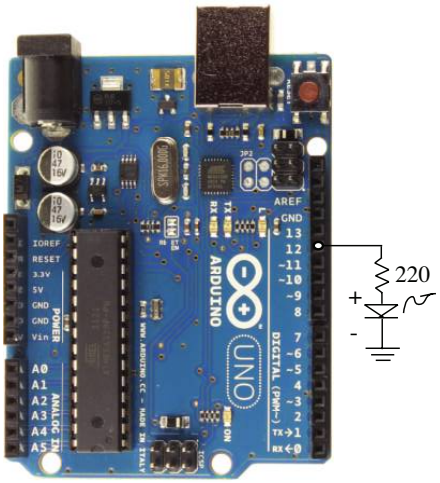
Figure 1.6: Arduino Development Environment built-in features (www.arduino.cc).

features. As new features are added, they will be released in future ADE versions. As an Arduino user, you too may add to this collection of useful tools. We continue with another example.

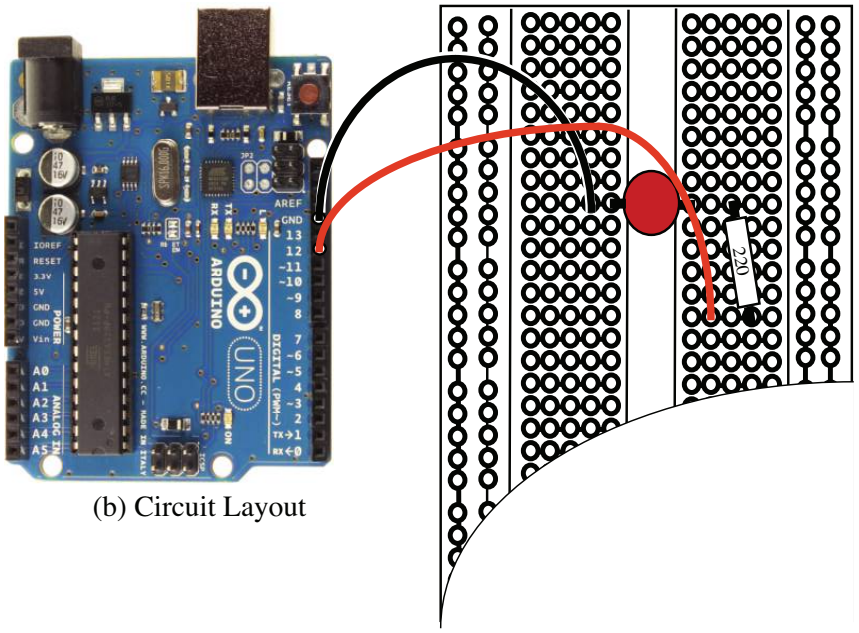
Example: In this example we connect an external LED to Arduino UNO R3 pin 12. The on-board LED will blink alternately with the external LED. The external LED is connected to the Arduino UNO R3 as shown in Figure 1.7. The LED has a positive (anode) and negative (cathode) lead. As you look down from above, a round LED has a flat side. The lead closest to the flat side is the cathode.

In the bottom-right portion of the figure, a cutaway is provided of a prototype board. Prototype boards provide a convenient method of interconnecting electronic components. The boards are configured to accept dual inline package (DIP) integrated circuits (ICs or chips). The ICs are placed over the center channels. Other components may also be placed on the boards. The boards are covered with multiple connection points. Typically, the board's connection points are arranged in columns and five connection point rows. The connection points in a column are connected at the board's base by a conductor. Also, connection points in a board row are connected.

The connection points readily accept insulated, solid 22 AWG insulated wire. This wire type is available in variety of colors (www.jameco.com). To connect two circuit points together,



(a) Schematic



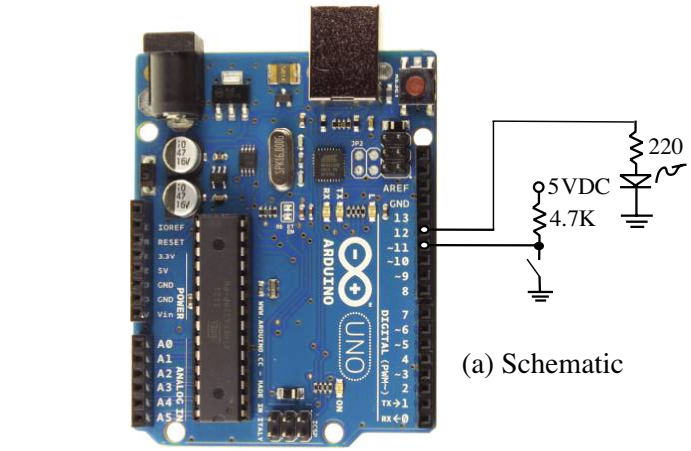
(b) Circuit Layout

Figure 1.7: Arduino UNO R3 with an external LED. (UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA) www.arduino.cc).

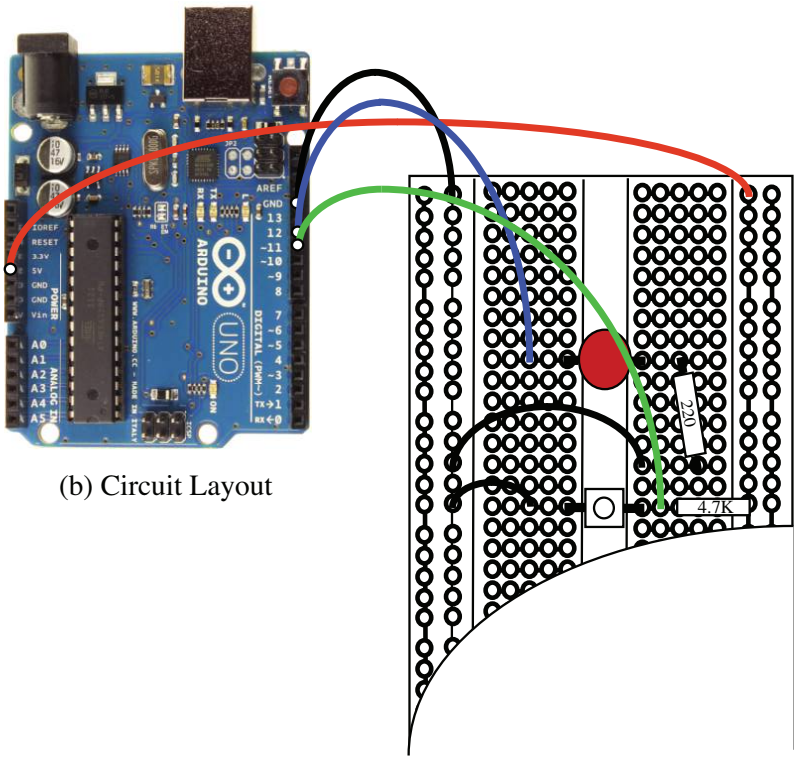
estimate the length of wire needed to connect the points. Cut the solid 22 AWG insulated wire to that length plus an additional one-half inch (approximately 13 mm). Using a wire stripper (e.g., Jameco #159291), strip off approximately one-quarter inch (7.5 mm) from each end of the wire. The conductor exposed ends of the wire can then be placed into appropriate circuit locations for a connection. Once a wire end is placed in a breadboard hole, the exposed conductor should not be visible. Circuit connections may also be made using prefabricated jumper wires. These are available from a number of sources such as Jameco www.jameco.com, Adafruit www.adafruit.com, and SparkFun Electronics (www.sparkfun.com).

```
//*********************************************************

#define int_LED 13                      //name pin 13 int_LED
#define ext_LED 12                      //name pin 12 ext_LED

void setup()
{
pinMode(int_LED, OUTPUT);               //set pin to output
pinMode(ext_LED, OUTPUT);               //set pin to output
}

void loop()
{
digitalWrite(int_LED, HIGH);            //set pin logic high
digitalWrite(ext_LED, LOW);             //set pin logic low
delay(500);                             //delay specified in ms
digitalWrite(int_LED, LOW);             //set pin logic low
digitalWrite(ext_LED, HIGH);            //set pin logic high
delay(500);
}

//*********************************************************
```

Example: In this example we connect an external LED to Arduino UNO R3 pin 12 and an external switch attached to pin 11. The onboard LED will blink alternately with the external LED when the switch is depressed. The external LED and switch is connected to the Arduino UNO R3, as shown in Figure 1.8.



(a) Schematic



(b) Circuit Layout

Figure 1.8: Arduino UNO R3 with an external LED. (UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA) www.arduino.cc).

```
//*************************************************************

#define int_LED 13                      //name pin 13 int_LED
#define ext_LED 12                      //name pin 12 ext_LED
#define ext_sw  11                      //name pin 11 ext_sw

int switch_value;                       //integer variable to
                                        //store switch status

void setup()
{
pinMode(int_LED, OUTPUT);              //set pin to output
pinMode(ext_LED, OUTPUT);              //set pin to output
pinMode(ext_sw,  INPUT);               //set pin to input
}

void loop()
{
switch_value = digitalRead(ext_sw);    //read switch status
if(switch_value == LOW)                //if switch at logic low,
  {                                    //do steps with braces
  digitalWrite(int_LED, HIGH);         //set pin logic high
  digitalWrite(ext_LED, LOW);          //set pin logic low
  delay(50);                           //delay 50 ms
  digitalWrite(int_LED, LOW);          //set pin logic low
   digitalWrite(ext_LED, HIGH);        //set pin logic high
   delay(50);                          //delay 50ms
   }
else                                   //if switch at logic high,
   {                                   //do steps between braces
   digitalWrite(int_LED, LOW);         //set pins low
   digitalWrite(ext_LED, LOW);         //set pins low
   }
}

//*************************************************************
```

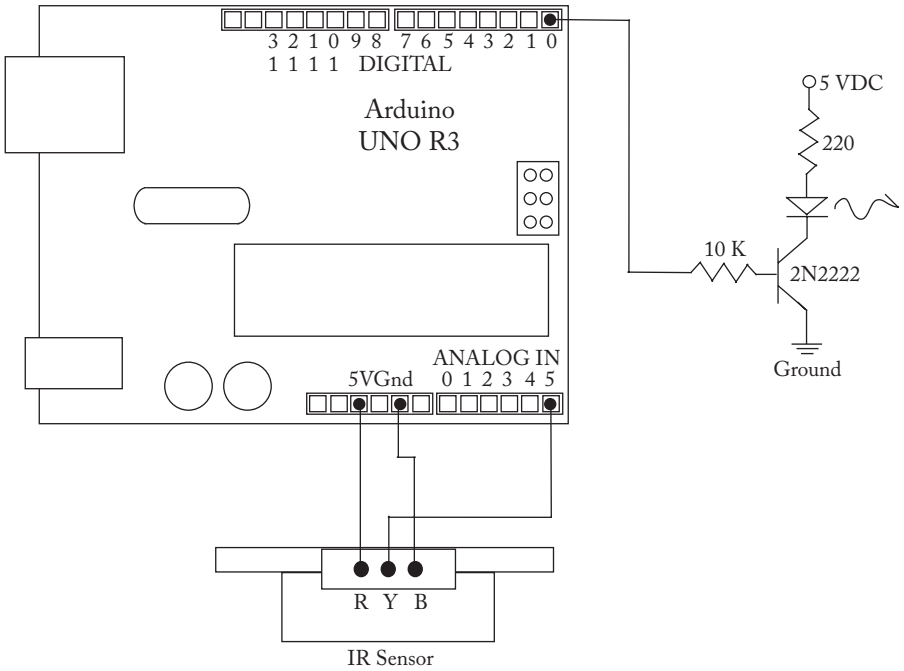


Figure 1.9: IR sensor interface.

1.4 APPLICATION: ROBOT IR SENSOR

In this example we investigate a sketches's interaction with the Arduino UNO R3 processing board and external sensors and indicators. We will use the robot project as an ongoing example.

In Chapter 2, we equip a robot platform with three Sharp GP2Y0A41SK0F (we abbreviate as GP2Y) infrared (IR) sensors. The IR sensor provides a voltage output that is inversely proportional to the sensor distance from the maze wall. It is desired to illuminate the LED if the robot is within 10 cm of the maze wall. The sensor provides an output voltage of 2.5 VDC at the 10-cm range. The interface between the IR sensor and the Arduino UNO R3 board is provided in Figure 1.9.

The IR sensor's power (red wire) and ground (black wire) connections are connected to the 5V and Gnd pins on the Arduino UNO R3 board, respectively. The IR sensor's output connection (yellow wire) is connected to the ANALOG IN 5 pin on the Arduino UNO R3 board. The LED circuit shown in the top-right corner of the diagram is connected to the DIGITAL 0 pin on the Arduino UNO R3 board. We discuss the operation of this circuit in Chapter 3.

```
//*********************************************************************
#define LED_PIN   0                 //digital pin - LED connection
#define IR_sensor_pin 5             //analog pin - IR sensor
```

```
int IR_sensor_value;                //declare variable for IR sensor value

void setup()
  {
   pinMode(LED_PIN, OUTPUT);        //configure pin 0 for digital output
  }

void loop()
  {
                                    //read analog output from IR sensor
   IR_sensor_value = analogRead(IR_sensor_pin);

   if(IR_sensor_value > 512)        //0 to 1023 maps to 0 to 5 VDC
     {
     digitalWrite(LED_PIN, HIGH);  //turn LED on
     }
   else
     {
     digitalWrite(LED_PIN, LOW);   //turn LED off
     }
  }
//*************************************************************************
```

The sketch begins by providing names for the two Arduino UNO R3 board pins that will be used in the sketch. This is not required but it makes the code easier to read. We define the pin for the LED as "LED_PIN." Any descriptive name may be used here. Whenever the name is used within the sketch, the number "0" will be substituted for the name by the compiler.

After providing the names for pins, the next step is to declare any variables required by the sketch. In this example, the output from the IR sensor will be converted from an analog to a digital value using the built-in Arduino "analogRead" function. A detailed description of the function may be accessed via the Help menu. It is essential to carefully review the support documentation for a built-in Arduino function the first time it is used. The documentation provides details on variables required by the function, variables returned by the function, and an explanation on function operation.

The "analogRead" function requires the pin for analog conversion variable passed to it and returns the analog signal read as an integer value (int) from 0–1023. So, for this example, we need to declare an integer value to receive the returned value. We have called this integer variable "IR_sensor_value."

Following the declaration of required variables are the two required functions for an Arduino UNO R3 program: setup and loop. The setup function calls an Arduino built-in function, pinMode, to set the "LED_PIN" as an output pin. The loop function calls several functions to read the current analog value on pin 5 (the IR sensor output) and then determine if the reading is above 512 (2.5 VDC). If the reading is above 2.5 VDC, the LED on DIGITAL pin 0 is illuminated, else it is turned off.

After completing writing the sketch with the ADE, it must be compiled and then uploaded to the Arduino UNO R3 board. These two steps are accomplished using the "Sketch—Verify/Compile" and the "File—Upload to I/O Board" pull down menu selections.

As an exercise, develop a range vs. voltage plot for the IR sensor for ranges from 0–25 cm.

1.5 APPLICATION: BLINK LED FIBER

For the next two applications we use optical fibers coupled to light-emitting diodes (LED) for visual display effects. We start with some background information on optical fibers. Optical fibers are used to link two devices via light rather than an electronic signal. The flexible optical links are used in electronically noisy environments because the optical link is not susceptible to electronic noise. In a typical application an electronic signal is converted to light, transmitted down the optical fiber, and converted back to an electronic signal.

As shown in Figure 1.10a an optical fiber consists of several concentric layers of material including the core where light is transmitted, the cladding, the buffer, and the protective outer jacket. Light is transmitted through the fiber via the concept of total internal reflection. The core material is more optically dense than the cladding material. At shallow entry angles the light reflects from the core/cladding boundary and stays within the fiber core, as shown in Figure 1.10b. This allows for the transmission of light via fiber for long distances with limited signal degradation.

To provide an interface between an electronic signal and the fiber, an optical emitter is used as shown in Figure 1.10c. The optical emitter contains an LED as the light source. At the far end of the optical fiber an optical detector is used to convert the light signal back to an electronic one.

It is important to note that optical emitters, detectors, and fibers are available in a variety of wavelengths as shown in Figure 1.10d. It is important that the emitter, detector, and fiber are capable at operating at the same optical wavelengths.

In this example we use a transistor (PN2222) to boost the current from the Arduino so it is compatible with the voltage and current requirements of the red LED (660 nm, IF-E97, $V_f = 1.7$, I_f 40 mA), as shown in Figure 1.11. We describe this interface circuit in more detail in Chapter 3.

We reuse the code example of flashing an external LED.

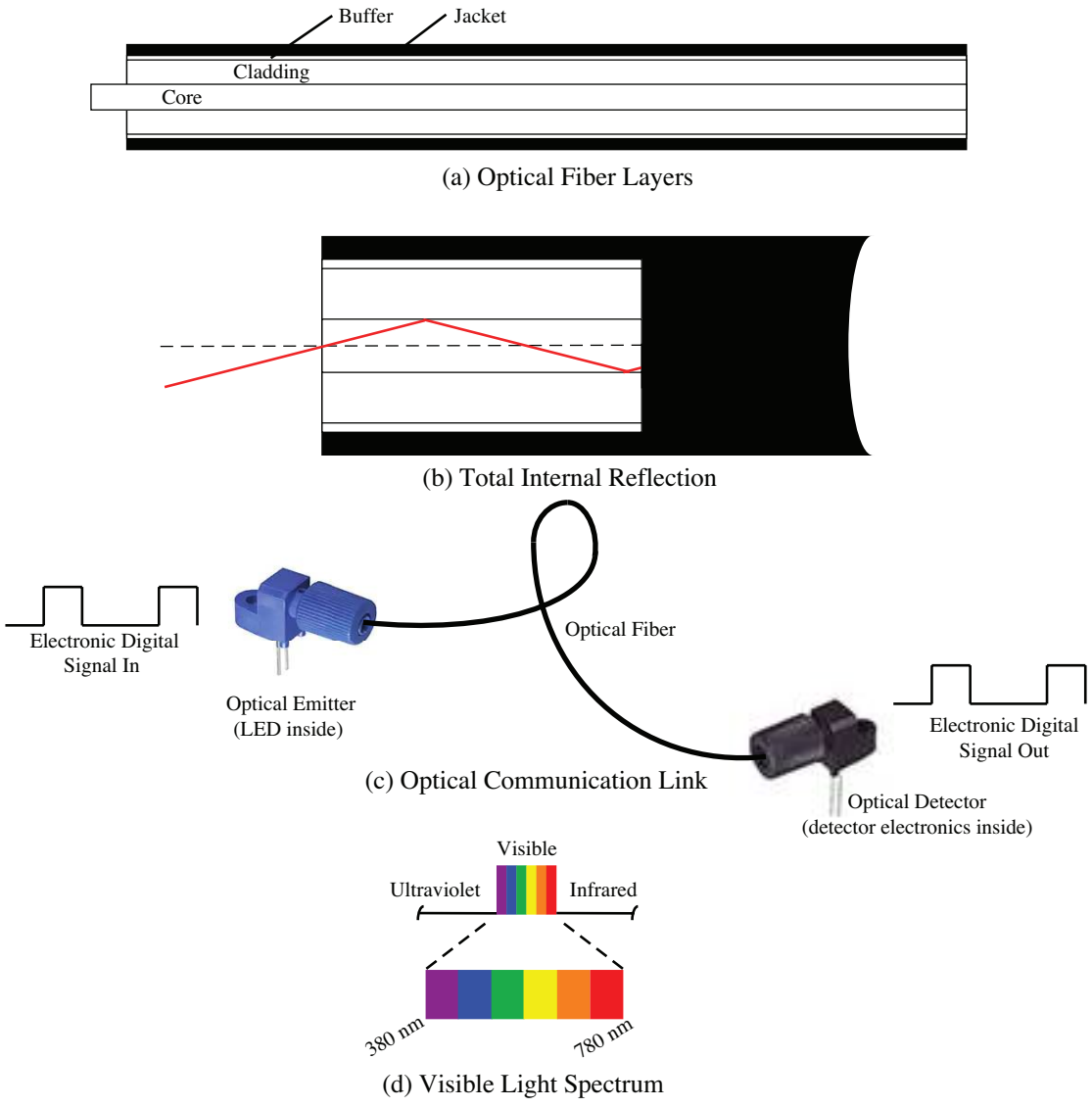


Figure 1.10: Optical fibers.

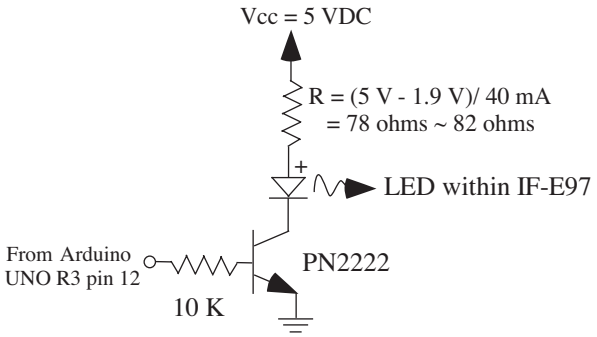
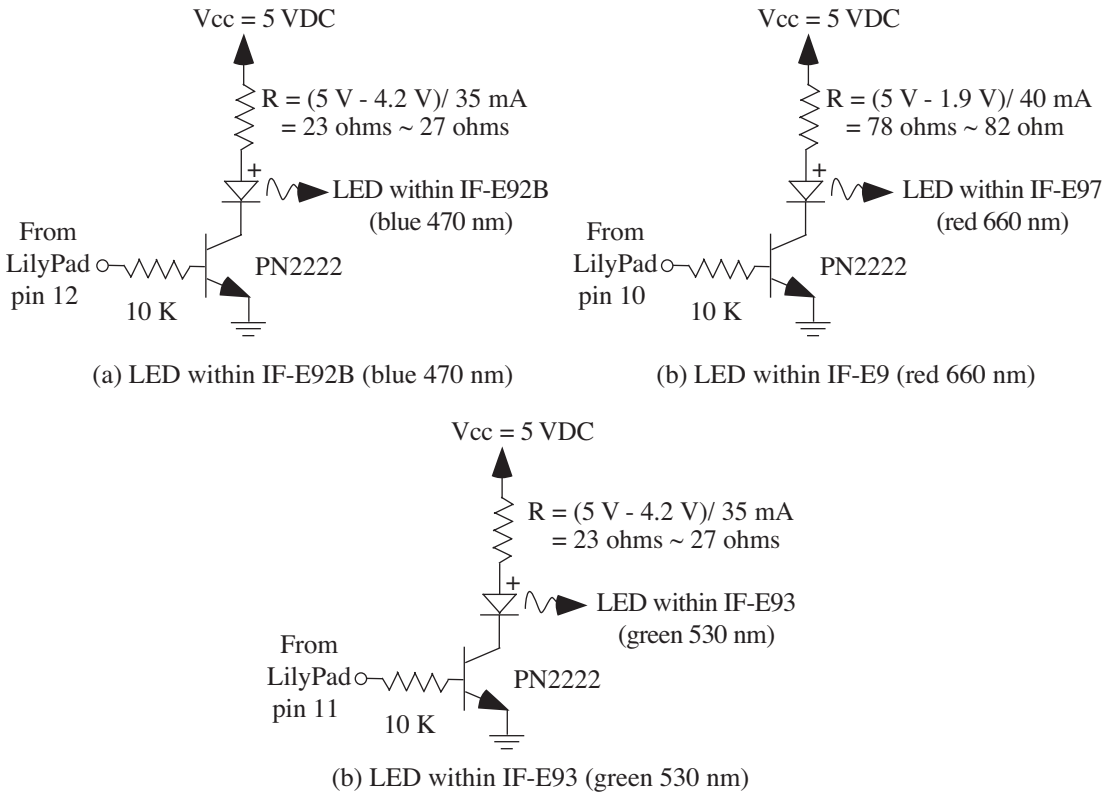


Figure 1.11: Interface for optical fibers.

```
//*********************************************************

#define ext_LED 12                      //name pin 12 ext_LED
#define int_LED 13                      //name pin 13 int_LED

void setup()
{
pinMode(int_LED, OUTPUT);               //set pin to output
pinMode(ext_LED, OUTPUT);               //set pin to output
}

void loop()
{
digitalWrite(int_LED, HIGH);            //set pin to logic high
digitalWrite(ext_LED, LOW);             //set pin to logic low
delay(500);                             //delay specified in ms
digitalWrite(int_LED, LOW);             //set pin to logic low
digitalWrite(ext_LED, HIGH);            //set pin to logic high
delay(500);                             //delay specified in ms
}

//*********************************************************
```

1.6 APPLICATION: LILYPAD WITH LED FIBERS

The LilyPad is a wearable Arduino. In this example, we use the LilyPad Arduino 328 Main Board (Sparkfun #DEV-13342) equipped with the Microchip Mega 328P. More information



(a) LED within IF-E92B (blue 470 nm)

(b) LED within IF-E9 (red 660 nm)

(b) LED within IF-E93 (green 530 nm)

Figure 1.12: LilyPad interface for optical fibers.

about this board is provided in the next chapter. The LilyPad is connected to three different fibers sourced with different LEDs (red, blue, and green) via pins 10, 11, and 12. As in the previous example, transistors are used to interface the LilyPad to the fibers as shown in Figure 1.12.

The example code sequentially illuminates each fiber.

```
//*********************************************************

#define ext_fiber_red   10        //name pin 10 ext_fiber_red
#define ext_fiber_green 11        //name pin 11 ext_fiber_green
#define ext_fiber_blue  12        //name pin 12 ext_fiber_blue

void setup()
{
pinMode(ext_fiber_red,   OUTPUT); //set pin to output
pinMode(ext_fiber_green, OUTPUT); //set pin to output
```

```
pinMode(ext_fiber_blue,  OUTPUT); //set pin to output
}

void loop()
{
digitalWrite(ext_fiber_red,  HIGH);//set pin logic high
digitalWrite(ext_fiber_green, LOW);//set pin logic low
digitalWrite(ext_fiber_blue,  LOW);//set pin logic low
delay(500);                        //delay specified in ms

digitalWrite(ext_fiber_red,   LOW);//set pin logic low
digitalWrite(ext_fiber_green,HIGH);//set pin logic high
digitalWrite(ext_fiber_blue,  LOW);//set pin logic low
delay(500);                        //delay specified in ms

digitalWrite(ext_fiber_red,   LOW);//set pin logic low
digitalWrite(ext_fiber_green, LOW);//set pin logic low
digitalWrite(ext_fiber_blue, HIGH);//set pin logic high
delay(500);                        //delay specified in ms
}

//*********************************************************
```

1.7 APPLICATION: FRIEND OR FOE SIGNAL

In aerial combat a "friend or foe" signal is used to identify aircraft on the same side. The signal is a distinctive pattern only known by aircraft on the same side. In this example, we generate a friend signal on the internal LED on pin 13 that consists of 10 pulses of 40 ms each followed by a 500-ms pulse and then 1000 ms when the LED is off.

In this example we use a **for** loop. The for loop provides a mechanism for looping through the same portion of code a fixed number of times. The for loop consists of three main parts:

- loop initialization,
- loop termination testing, and
- the loop increment.

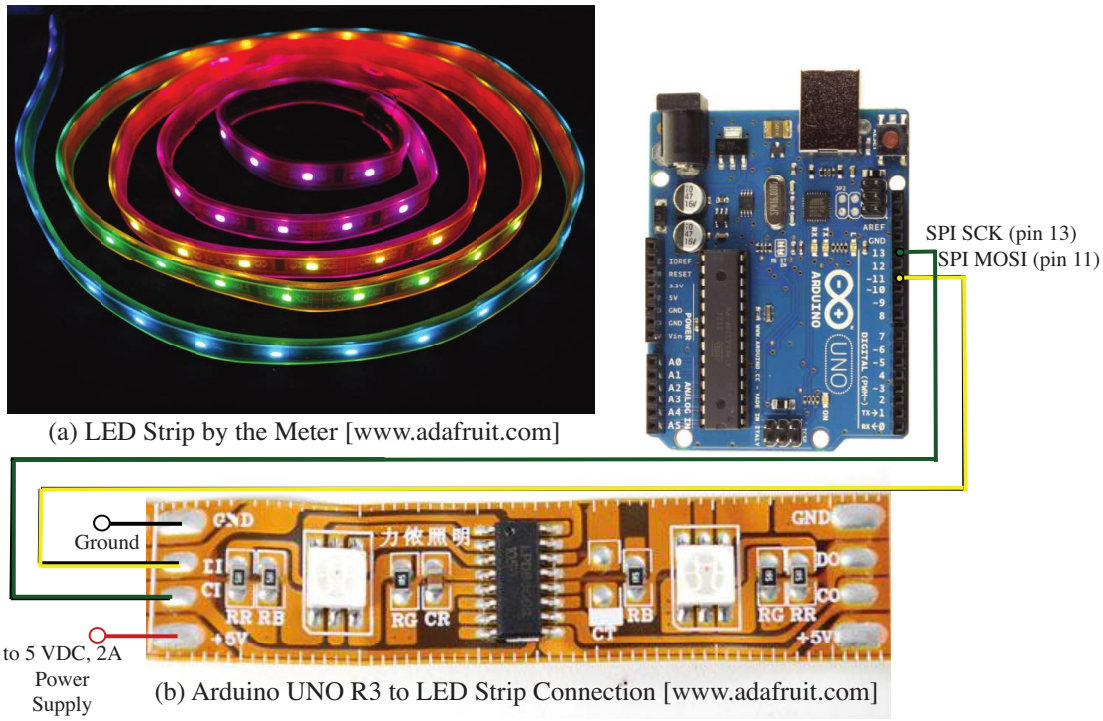
In the following code fragment the for loop is executed ten times.

```
//*************************************************************

#define LED_PIN 13                      //name pin 13 LED_PIN

void setup()
{
pinMode(LED_PIN, OUTPUT);               //set pin to output
}

void loop()
{
int i;                                  //for loop counter

for(i=0; i<=9; i++)                     //for loop control
  {                                     //execute loop 10 times
  digitalWrite(LED_PIN, HIGH);          //set pin high
  delay(20);                            //delay specified in ms
  digitalWrite(LED_PIN, LOW);           //set pin low
  delay(20);                            //delay specified in ms
  }                                     //exit loop

digitalWrite(LED_PIN, HIGH);            //set pin high
delay(500);                             //delay specified in ms
digitalWrite(LED_PIN, LOW);             //set pin low
delay(1000);                            //delay specified in ms
}

//*************************************************************
```

1.8 APPLICATION: LED STRIP

Example: LED strips may be used for motivational (fun) optical displays, games, or for instrumentation-based applications. In this example we control an LPD8806-based LED strip using the Arduino UNO R3. We use a one meter, 32 RGB LED strip available from Adafruit (#306) for approximately $30 USD (www.adafruit.com).

The red, blue, and green component of each RGB LED is independently set using an eight-bit code. The most significant bit (MSB) is logic one followed by seven bits to set the LED intensity (0–127). The component values are sequentially shifted out of the Arduino UNO R3



(a) LED Strip by the Meter [www.adafruit.com]

(b) Arduino UNO R3 to LED Strip Connection [www.adafruit.com]

Figure 1.13: UNO R3 controlling LED strip. LED strip illustration used with permission of Adafruit (www.adafruit.com). UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA) (www.arduino.cc).

using the Serial Peripheral Interface (SPI) features. The first component value shifted out corresponds to the LED nearest the microcontroller. Each shifted component value is latched to the corresponding R, G, and B component of the LED. As a new component value is received, the previous value is latched and held constant. An extra byte is required to latch the final parameter value. A zero byte $(00)_{16}$ is used to complete the data sequence and reset back to the first LED (www.adafruit.com).

Only four connections are required between the UNO R3 and the LED strip, as shown in Figure 1.13. The connections are color coded: red-power, black-ground, yellow-data, and green-clock. It is important to note the LED strip requires a supply of 5 VDC and a current rating of 2 amps per meter of LED strip. In this example we use the Adafruit #276 5V 2A (2000 mA) switching power supply (www.adafruit.com).

In this example each RGB component is sent separately to the strip. The example illustrates how each variable in the program controls a specific aspect of the LED strip. Here are some important implementation notes.

- SPI must be configured for most significant bit (MSB) first.
- LED brightness is seven bits. Most significant bit (MSB) must be set to logic one.
- Each LED requires a separate R-G-B intensity component. The order of data is G-R-B.
- After sending data for all LEDs. A byte of (0x00) must be sent to return strip to first LED.
- Data stream for each LED is: 1-G6-G5-G4-G3-G2-G1-G0-1-R6-R5-R4-R3-R2-R1-R0-1-B6-B5-B4-B3-B2-B1-B0.

```
//*********************************************************************
//RGB_led_strip_tutorial: illustrates different variables within
//RGB LED strip
//
//LED strip LDP8806 - available from www.adafruit.com (#306)
//
//Connections:
// - External 5 VDC supply - Adafruit 5 VDC, 2A (#276) - red
// - Ground - black
// - Serial Data In - Arduino pin 11 (MOSI pin)- yellow
// - CLK - Arduino pin 13 (SCK pin)- green
//
//Variables:
// - LED_brightness - set intensity from 0 to 127
// - segment_delay  - delay between LED RGB segments
// - strip_delay    - delay between LED strip update
//
//Notes:
// - SPI must be configured for Most significant bit (MSB) first
// - LED brightness is seven bits.  Most significant bit (MSB)
//   must be set to logic one
// - Each LED requires a separate R-G-B intensity component.  The order
//   of data is G-R-B.
// - After sending data for all strip LEDs.  A byte of (0x00) must
//   be sent to return strip to first LED.
// - Data stream for each LED is:
//1-G6-G5-G4-G3-G2-G1-G0-1-R6-R5-R4-R3-R2-R1-R0-1-B6-B5-B4-B3-B2-B1-B0
//
//This example code is in the public domain.
//*********************************************************************
```

```
#include <SPI.h>

#define LED_strip_latch 0x00

const byte strip_length = 32;          //number of RGB LEDs in strip
const byte segment_delay = 100;        //delay in milliseconds
const byte strip_delay = 500;          //delay in milliseconds
unsigned char   LED_brightness;        //0 to 127
unsigned char   position;              //LED position in strip
unsigned char   troubleshooting = 0;   //allows printouts to serial
                                       //monitor

void setup()
  {
  SPI.begin();                          //SPI support functions
  SPI.setBitOrder(MSBFIRST);            //SPI bit order
  SPI.setDataMode(SPI_MODE3);           //SPI mode
  SPI.setClockDivider(SPI_CLOCK_DIV32);//SPI data clock rate
  Serial.begin(9600);                   //serial comm at 9600 bps
  }

void loop()
  {
  SPI.transfer(LED_strip_latch);        //reset to first segment
  clear_strip();                        //all strip LEDs to black
  delay(500);

  //increment the green intensity of the strip LEDs
  for(LED_brightness = 0; LED_brightness <= 60;
      LED_brightness = LED_brightness + 10)
    {
    for(position = 0; position<strip_length; position = position+1)
      {
      SPI.transfer(0x80 | LED_brightness);  //Green - MSB 1
      SPI.transfer(0x80 | 0x00);            //Red   - none
      SPI.transfer(0x80 | 0x00);            //Blue  - none

      if(troubleshooting)
        {
```

```
      Serial.println(LED_brightness, DEC);
      Serial.println(position, DEC);
      }
    delay(segment_delay);
    }
  SPI.transfer(LED_strip_latch);     //reset to first segment
  delay(strip_delay);
  if(troubleshooting)
      {
      Serial.println(" ");
      }
  }

clear_strip();                       //all strip LEDs to black
delay(500);

//increment the red intensity of the strip LEDs
for(LED_brightness = 0; LED_brightness <= 60;
    LED_brightness = LED_brightness + 10)
  {
  for(position = 0; position<strip_length; position = position+1)
    {
    SPI.transfer(0x80 | 0x00);             //Green - none
    SPI.transfer(0x80 | LED_brightness);   //Red   - MSB1
    SPI.transfer(0x80 | 0x00);             //Blue  - none

    if(troubleshooting)
      {
      Serial.println(LED_brightness, DEC);
      Serial.println(position, DEC);
      }
    delay(segment_delay);
    }
  SPI.transfer(LED_strip_latch);     //reset to first segment
  delay(strip_delay);
  if(troubleshooting)
    {
    Serial.println(" ");
    }
```

```
    }

  clear_strip();                        //all strip LEDs to black
  delay(500);

  //increment the blue intensity of the strip LEDs
  for(LED_brightness = 0; LED_brightness <= 60;
      LED_brightness = LED_brightness + 10)
    {
    for(position = 0; position<strip_length; position = position+1)
      {
      SPI.transfer(0x80 | 0x00);            //Green - none
      SPI.transfer(0x80 | 0x00);            //Red   - none
      SPI.transfer(0x80 | LED_brightness);  //Blue  - MSB1

      if(troubleshooting)
        {
        Serial.println(LED_brightness, DEC);
        Serial.println(position, DEC);
        }
      delay(segment_delay);
      }
    SPI.transfer(LED_strip_latch);     //reset to first segment
    delay(strip_delay);
    if(troubleshooting)
      {
      Serial.println(" ");
      }
   }

  clear_strip();                       //all strip LEDs to black
  delay(500);
 }

//****************************************************************

void clear_strip(void)
{
  //clear strip
```

```
  for(position = 0; position<strip_length; position = position+1)
      {
      SPI.transfer(0x80 | 0x00);            //Green - none
      SPI.transfer(0x80 | 0x00);            //Red - none
      SPI.transfer(0x80 | 0x00);            //Blue  - none

      if(troubleshooting)
        {
        Serial.println(LED_brightness, DEC);
        Serial.println(position, DEC);
        }
      }
  SPI.transfer(LED_strip_latch);            //Latch with zero
  if(troubleshooting)
    {
    Serial.println(" ");
    }
  delay(2000);                              //clear delay
}

//*****************************************************************
```

1.9 APPLICATION: EXTERNAL INTERRUPTS

The interrupt system onboard a microcontroller allows it to respond to higher priority events. Appropriate responses to these events may be planned, but we do not know when these events will occur. When an interrupt event occurs, the microcontroller will normally complete the instruction it is currently executing and then transition program control to interrupt event specific tasks. These tasks, which resolve the interrupt event, are organized into a function called an interrupt service routine (ISR). Each interrupt will normally have its own interrupt specific ISR. Once the ISR is complete, the microcontroller will resume processing where it left off before the interrupt event occurred.

The ADE has four built-in functions to support external the INT0 and INT1 external interrupts (www.arduino.cc).

These are the four functions.

- **interrupts()**. This function enables interrupts.
- **noInterrupts()**. This function disables interrupts.
- **attachInterrupt(interrupt, function, mode)**. This function links the interrupt to the appropriate interrupt service routine.

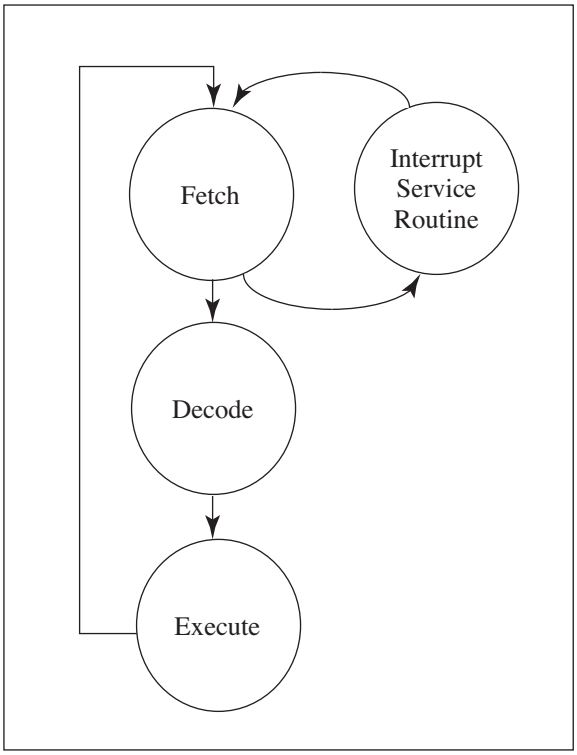


Figure 1.14: Microcontroller Interrupt Response.

- **detachInterrupt(interrupt)**. This function turns off the specified interrupt.

The Arduino UNO R3 processing board is equipped with two external interrupts: INT0 on pin 2 and INT1 on pin 3. The Arduino Mega 2560 processing board is equipped with six external interrupts: INT0 at pin 2, INT1 at pin 3, INT2 at pin 21, INT3 at pin 1, INT4 at pin 19, and INT5 at pin 18.

The **attachInterrupt(interrupt, function, mode)** function is used to link the hardware pin to the appropriate interrupt service pin. The three arguments of the function are configured as follows.

- **interrupt.** Interrupt specifies the INT interrupt number: either 0 or 1.
- **function.** Function specifies the name of the interrupt service routine.
- **mode.** Mode specifies what activity on the interrupt pin will initiate the interrupt: **LOW** level on pin, **CHANGE** in pin level, **RISING** edge, or **FALLING** edge.

Provided below is a template to configure an interrupt.

```
//****************************************************************

void setup()
{
attachInterrupt(0, int0_ISR, FALLING);
}

void loop()
{

//wait for interrupts

}

//****************************************************************
//int0_ISR: interrupt service routine for INT0
//****************************************************************

void int0_ISR(void)
{

//Insert interrupt specific actions here.

}
//****************************************************************
```

As an example, we connect an external tact switch to INT0 (pin 2) and measure the elapsed time between two switch presses. The switch configuration provided earlier in Figure 1.8 may be used.

```
//****************************************************************
//Program measures the elapsed time in ms between two switch
//closures.  A tact switch with a series 4.7K resistor is attached
//to INT0 (pin 2) of the UNO R3.
//****************************************************************

unsigned long first, second, elapsed_time;
unsigned int first_time_hack = 1;

void setup()
{
```

```
Serial.begin(9600);
pinMode(2, INPUT);
attachInterrupt(0, int0_ISR, FALLING);
}

void loop()
{

//wait for interrupts

}

//*****************************************************************
//int0_ISR: interrupt service routine for INT0
//*****************************************************************

void int0_ISR(void)
{
if(first_time_hack ==1)
  {
  first = millis();
  first_time_hack = 0;
  delay(5);
  }
else
  {
  second = millis();
  first_time_hack = 1;
  elapsed_time = second - first;
  Serial.print(elapsed_time);
  Serial.println(" ms");
  Serial.println();
  delay(5);
  }
}

//*****************************************************************
```

1.10 SUMMARY

The goal of this chapter was to provide a tutorial on how to begin programming. We used a top-down design approach. We began with the "big picture" of the chapter followed by an overview of the ADE. Throughout the chapter, we provided examples and also provided references to a number of excellent references.

1.11 REFERENCES

[1] Arduino homepage, www.arduino.cc.

[2] Duree G. (2011). *Optics for Dummies*, Wiley Publishing, Inc.

1.12 CHAPTER PROBLEMS

1.1. Describe the steps in writing a sketch and executing it on an Arduino UNO R3 processing board.

1.2. What is the serial monitor feature used for in the ADE?

1.3. Describe what variables are required and returned and the basic function of the following built-in Arduino functions: Blink, Analog Input.

1.4. Adapt Application: Robot IR sensor for the Arduino Mega 2560 processor board.

1.5. Adapt Application: Art Piece Illumination System for the Arduino Mega 2560 processor board.

1.6. Adapt Application: LED Strip to a light sequence of your creation.

CHAPTER 2

Arduino Platforms

Objectives: After reading this chapter, the reader should be able to the following:

- describe the Arduino concept of open-source hardware;
- diagram the layout of the Arduino UNO R3 processor board;
- name and describe the different features aboard the Arduino UNO R3 processor board;
- discuss the features and functions of the Microchip ATmega328;
- diagram the layout of the Arduino Mega 2560 processor board;
- name and describe the different features aboard the Arduino Mega 2560 R3 processor board;
- discuss the features and functions of the Microchip ATmega2560;
- diagram the layout of the Arduino LilyPad;
- name and describe the different features aboard the Arduino LilyPad;
- discuss the features and functions of the LilyPad; and
- describe how to extend the hardware features of the Arduino processor using Arduino Shields.

2.1 OVERVIEW

Throughout the book, we use three different Arduino processing boards: the Arduino UNO R3 board (UNO), the Arduino Mega 2560 REV3 board (MEGA), and the Arduino LilyPad. Starter kits for these platforms are available from a number of sources.

2.2 ARDUINO UNO R3 PROCESSING BOARD

The Arduino UNO R3 processing board is illustrated in Figure 2.1. Working clockwise from the left, the board is equipped with a USB connector to allow programming the processor from a host personal computer (PC) or laptop. The board may also be programmed using In System

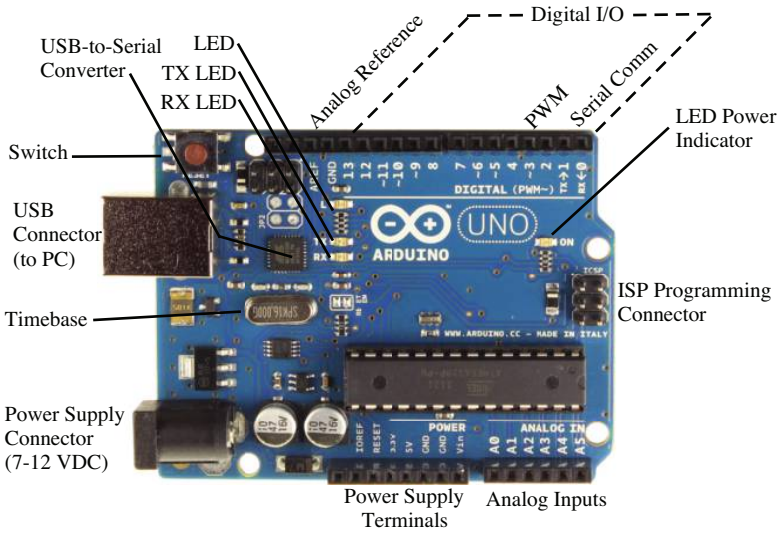


Figure 2.1: Arduino UNO R3 layout. (Figure adapted and used with permission of Arduino Team (CC BY-NC-SA) www.arduino.cc.)

Programming (ISP) techniques. A 6-pin ISP programming connector is on the opposite side of the board from the USB connector.

The board is equipped with a USB-to-serial converter to allow compatibility between the host PC and the serial communications systems aboard the Microchip ATmega328 processor. The UNO R3 is also equipped with several small surface-mount LEDs to indicate serial transmission (TX) and reception (RX) and an extra LED for project use. The header strip at the top of the board provides access for an analog reference signal, pulse width modulation (PWM) signals, digital input/output (I/O), and serial communications. The header strip at the bottom of the board provides analog inputs for the analog-to-digital (ADC) system and power supply terminals. Finally, the external power supply connector is provided at the bottom left corner of the board. The top and bottom header strips conveniently mate with an Arduino shield to extend the features of the Arduino host processor.

2.3 ADVANCED: ARDUINO UNO R3 HOST PROCESSOR – THE ATMEGA328

The host processor for the Arduino UNO R3 is the Microchip Atmega328. The "328" is a 28 pin, 8-bit microcontroller. The architecture is based on the Reduced Instruction Set Computer (RISC) concept which allows the processor to complete 20 million instructions per second (MIPS) when operating at 20 MHz. The "328" is equipped with a wide variety of features as shown in Figure 2.2. The features may be conveniently categorized into the following systems:

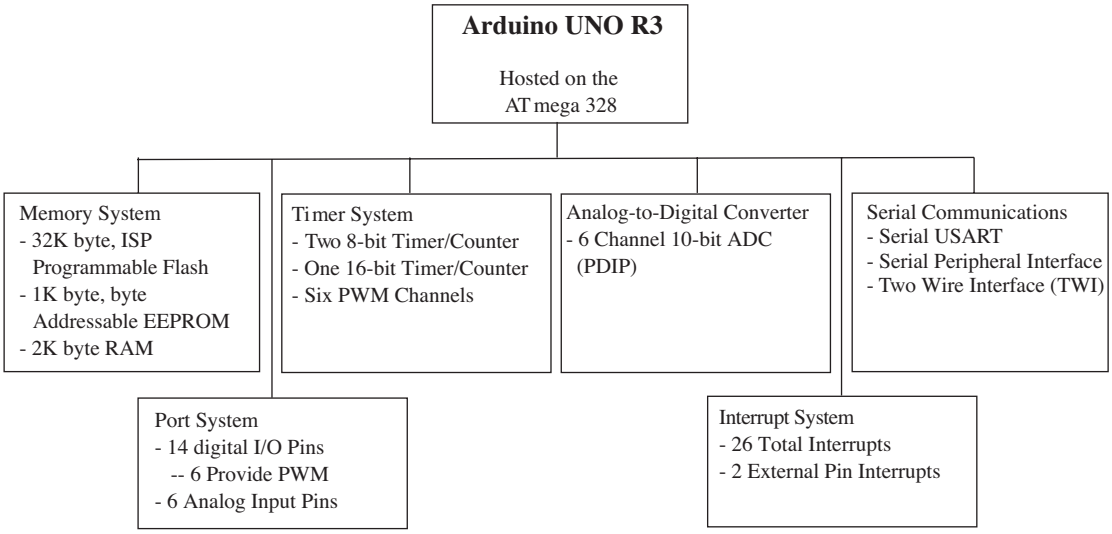


Figure 2.2: Arduino UNO R3 systems.

- memory system,
- port system,
- timer system,
- analog-to-digital converter (ADC),
- interrupt system, and
- serial communications.

2.3.1 ARDUINO UNO R3/ATMEGA328 HARDWARE FEATURES

The Arduino UNO R3's processing power is provided by the ATmega328. The pin out diagram and block diagram for this processor are provided in Figures 2.3 and 2.4. In this section, we provide a brief overview of the systems aboard the processor.

2.3.2 ATMEGA328 MEMORY

The ATmega328 is equipped with three main memory sections: flash electrically erasable programmable read-only memory (EEPROM), static random access memory (SRAM), and byte-addressable EEPROM. We discuss each memory component in turn.

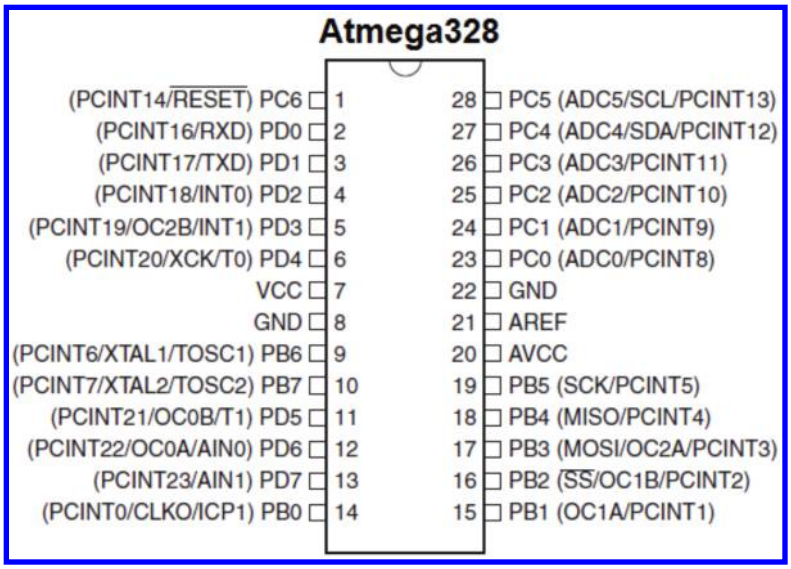


Figure 2.3: ATmega328 pin out. (Figure used with permission of Microchip, Incorporated.)

2.3.2.1 ATmega328 In-System Programmable Flash EEPROM

Bulk programmable flash EEPROM is used to store programs. It can be erased and programmed as a single unit. Also, should a program require a large table of constants, it may be included as a global variable within a program and programmed into flash EEPROM with the rest of the program. Flash EEPROM is nonvolatile meaning memory contents are retained even when microcontroller power is lost. The ATmega328 is equipped with 32K bytes of onboard reprogrammable flash memory. This memory component is organized into 16K locations with 16 bits at each location.

2.3.2.2 ATmega328 Byte-Addressable EEPROM

Byte-addressable EEPROM memory is used to permanently store and recall variables during program execution. It too is nonvolatile. It is especially useful for logging system malfunctions and fault data during program execution. It is also useful for storing data that must be retained during a power failure but might need to be changed periodically. Examples where this type of memory is used are found in applications to store system parameters, electronic lock combinations, and automatic garage door electronic unlock sequences. The ATmega328 is equipped with 1024 bytes of EEPROM.

2.3.2.3 ATmega328 Static Random Access Memory (SRAM)

Static RAM memory is volatile. That is, if the microcontroller loses power, the contents of SRAM memory are lost. It can be written to and read from during program execution. The

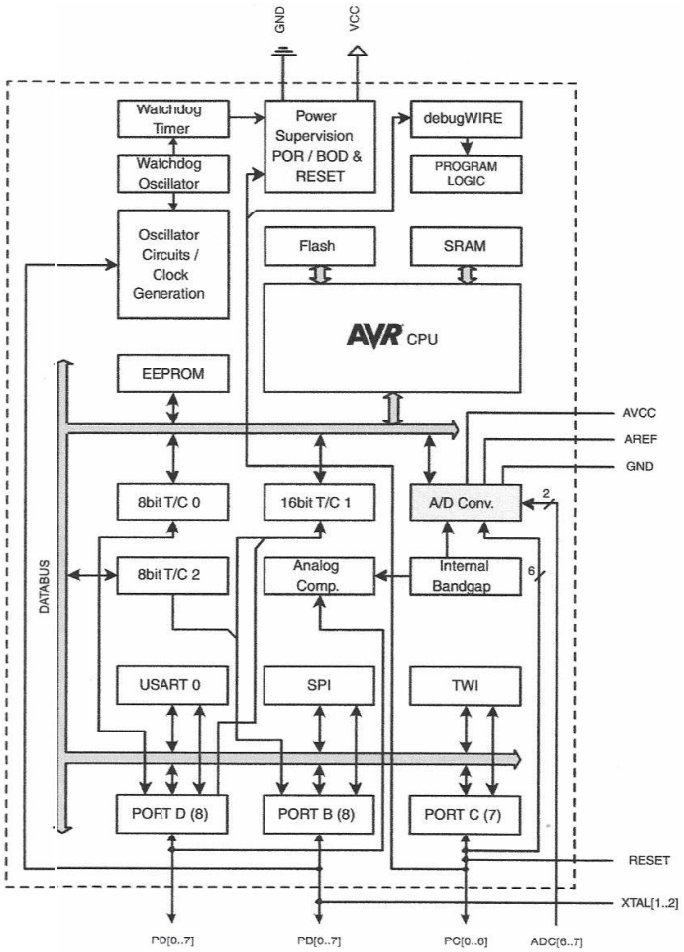


Figure 2.4: ATmega328 block diagram. (Figure used with permission of Microchip, Incorporated.)

ATmega328 is equipped with 2K bytes of SRAM. A small portion of the SRAM is set aside for the general-purpose registers used by the processor and also for the input/output and peripheral subsystems aboard the microcontroller. A header file provides the link between register names used in a program and their physical description and location in memory. During program execution, RAM is used to store global variables, support dynamic memory allocation of variables, and to provide a location for the stack.

2.3.3 ATMEGA328 PORT SYSTEM

The Microchip ATmega328 is equipped with four, 8-bit general purpose, digital input/output (I/O) ports designated PORTB (8 bits, PORTB[7:0]), PORTC (7 bits, PORTC[6:0]), and PORTD (8 bits, PORTD[7:0]). As shown in Figure 2.5, each port has three registers associated with it:

- Data Register PORTx—used to write output data to the port,
- Data Direction Register DDRx—used to set a specific port pin to either output (1) or input (0), and
- Input Pin Address PINx—used to read input data from the port.

Figure 2.5b describes the settings required to configure a specific port pin to either input or output. If selected for input, the pin may be selected for either an input pin or to operate in the high impedance (Hi-Z) mode. If selected for output, the pin may be further configured for either logic low or logic high.

Port pins are usually configured at the beginning of a program for either input or output and their initial values are then set. Usually all eight pins for a given port are configured simultaneously.

2.3.4 ATMEGA328 INTERNAL SYSTEMS

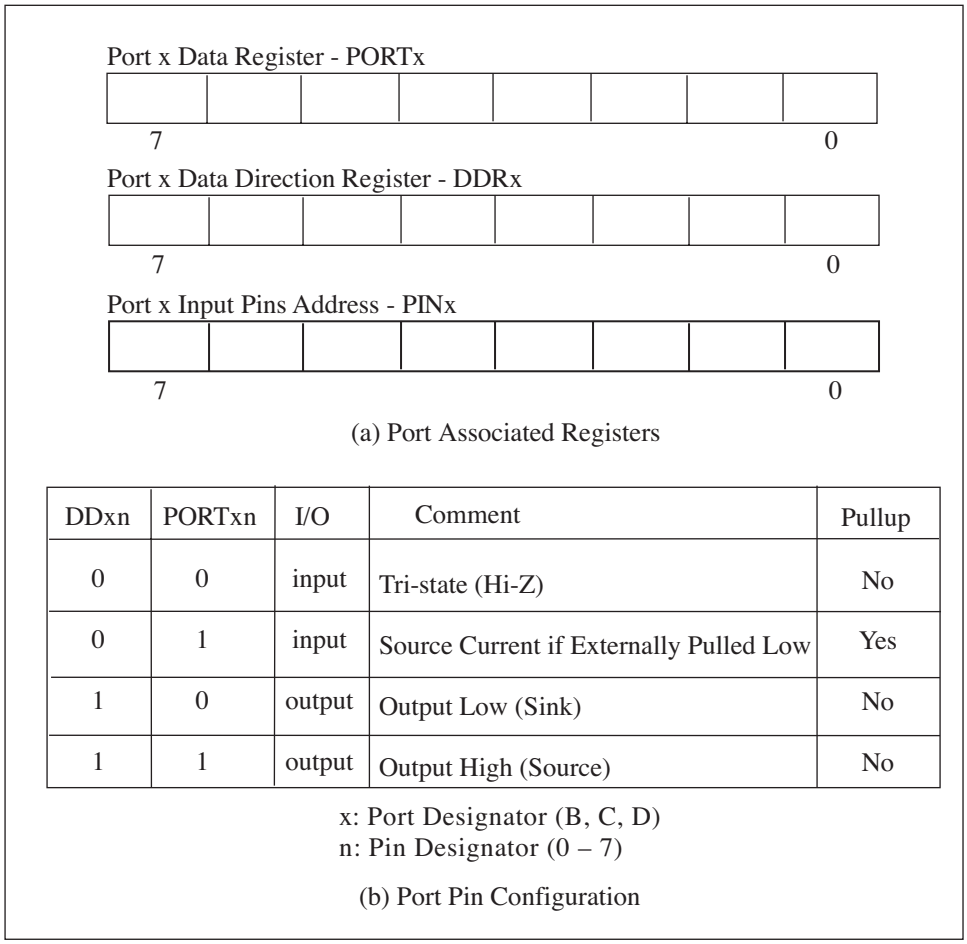
In this section, we provide a brief overview of the internal features of the ATmega328. It should be emphasized that these features are the internal systems contained within the confines of the microcontroller chip. These built-in features allow complex and sophisticated tasks to be accomplished by the microcontroller.

2.3.4.1 ATmega328 Time Base

The microcontroller is a complex synchronous state machine. It responds to program steps in a sequential manner as dictated by a user-written program. The microcontroller sequences through a predictable fetch-decode-execute sequence. Each unique assembly language program instruction issues a series of signals to control the microcontroller hardware to accomplish instruction related operations.

The speed at which a microcontroller sequences through these actions is controlled by a precise time base called the clock. The clock source is routed throughout the microcontroller to provide a time base for all peripheral subsystems. The ATmega328 may be clocked internally using a user-selectable resistor capacitor (RC) time base or it may be clocked externally. The RC internal time base is selected using programmable fuse bits. You may choose from several different internal fixed clock operating frequencies.

To provide for a wider range of frequency selections an external time source may be used. The external time sources, in order of increasing accuracy and stability, are an external RC net-



(a) Port Associated Registers

DDxn	PORTxn	I/O	Comment	Pullup
0	0	input	Tri-state (Hi-Z)	No
0	1	input	Source Current if Externally Pulled Low	Yes
1	0	output	Output Low (Sink)	No
1	1	output	Output High (Source)	No

x: Port Designator (B, C, D)
n: Pin Designator (0 – 7)

(b) Port Pin Configuration

Figure 2.5: ATmega328 port configuration registers.

work, a ceramic resonator, or a crystal oscillator. The system designer chooses the time base frequency and clock source device appropriate for the application at hand. Generally speaking, if the microcontroller will be interfaced to external peripheral devices either a ceramic resonator or a crystal oscillator should be used as a time base.

2.3.4.2 ATmega328 Timing Subsystem

The ATmega328 is equipped with a complement of timers which allows the user to generate a precision output signal, measure the characteristics (period, duty cycle, frequency) of an incoming digital signal, or count external events. Specifically, the ATmega328 is equipped with two 8-bit timer/counters and one 16-bit counter.

2.3.4.3 Pulse Width Modulation Channels

A pulse width modulated (PWM) signal is characterized by a fixed frequency and a varying duty cycle. Duty cycle is the percentage of time a repetitive signal is logic high during the signal period. It may be formally expressed as:

$$dutycycle[\%] = (on\ time/period) \times (100\%).$$

The ATmega328 is equipped with four PWM channels. The PWM channels coupled with the flexibility of dividing the time base down to different PWM subsystem clock source frequencies allows the user to generate a wide variety of PWM signals: from relatively high-frequency low-duty cycle signals to relatively low-frequency high-duty cycle signals.

PWM signals are used in a wide variety of applications including controlling the position of a servo motor and controlling the speed of a DC motor.

2.3.4.4 ATmega328 Serial Communications

The ATmega328 is equipped with a variety of different serial communication subsystems including the Universal Synchronous and Asynchronous Serial Receiver and Transmitter (USART), the serial peripheral interface (SPI), and the Two-wire Serial Interface. What these systems have in common is the serial transmission of data. In a serial communications transmission, serial data is sent a single bit at a time from transmitter to receiver.

ATmega328 Serial USART The serial USART may be used for full duplex (two-way) communication between a receiver and transmitter. This is accomplished by equipping the ATmega328 with independent hardware for the transmitter and receiver. The USART is typically used for asynchronous communication. That is, there is not a common clock between the transmitter and receiver to keep them synchronized with one another. To maintain synchronization between the transmitter and receiver, framing start and stop bits are used at the beginning and end of each data byte in a transmission sequence.

The ATmega328 USART is quite flexible. It has the capability to be set to different data transmission rates known as the Baud (bits per second) rate. The USART may also be set for data bit widths of 5–9 bits with one or two stop bits. Furthermore, the ATmega328 is equipped with a hardware generated parity bit (even or odd) and parity check hardware at the receiver. A single parity bit allows for the detection of a single bit error within a byte of data. The USART may also be configured to operate in a synchronous mode.

ATmega328 Serial Peripheral Interface – SPI The ATmega328 Serial Peripheral Interface (SPI) can also be used for two-way serial communication between a transmitter and a receiver. In the SPI system, the transmitter and receiver share a common clock source. This requires an additional clock line between the transmitter and receiver but allows for higher data transmission rates as compared to the USART.

The SPI may be viewed as a synchronous 16-bit shift register with an 8-bit half residing in the transmitter and the other 8-bit half residing in the receiver. The transmitter is designated the master since it is providing the synchronizing clock source between the transmitter and the receiver. The receiver is designated as the slave.

ATmega328 Two-Wire Serial Interface – TWI The TWI subsystem allows the system designer to network related devices (microcontrollers, transducers, displays, memory storage, etc.) together into a system using a two-wire interconnecting scheme. The TWI allows a maximum of 128 devices to be interconnected. Each device has its own unique address and may both transmit and receive over the two-wire bus at frequencies up to 400 kHz. This allows the device to freely exchange information with other devices in the network within a small area.

2.3.4.5 ATmega328 Analog to Digital Converter – ADC

The ATmega328 is equipped with an eight-channel ADC subsystem. The ADC converts an analog signal from the outside world into a binary representation suitable for use by the microcontroller. The ATmega328 ADC has 10-bit resolution. This means that an analog voltage between 0 and 5 V will be encoded into one of 1024 binary representations between $(000)_{16}$ and $(3FF)_{16}$. This provides the ATmega328 with a voltage resolution of approximately 4.88 mV.

2.3.4.6 ATmega328 Interrupts

The normal execution of a program follows a designated sequence of instructions. However, sometimes this normal sequence of events must be interrupted to respond to high priority faults and status both inside and outside the microcontroller. When these higher priority events occur, the microcontroller suspends normal operation and executes event specific actions contained within an interrupt service routine (ISR). Once the higher priority event has been serviced by the ISR, the microcontroller returns and continues processing the normal program.

The ATmega328 is equipped with a complement of 26 interrupt sources. Two of the interrupts are provided for external interrupt sources while the remaining interrupts support the efficient operation of peripheral subsystems aboard the microcontroller.

2.4 ARDUINO UNO R3 OPEN SOURCE SCHEMATIC

The entire line of Arduino products is based on the visionary concept of open-source hardware and software. That is, hardware and software developments are openly shared among users to stimulate new ideas and advance the Arduino concept. In keeping with the Arduino concept, the Arduino team openly shares the schematic of the Arduino UNO R3 processing board; see Figure 2.6.

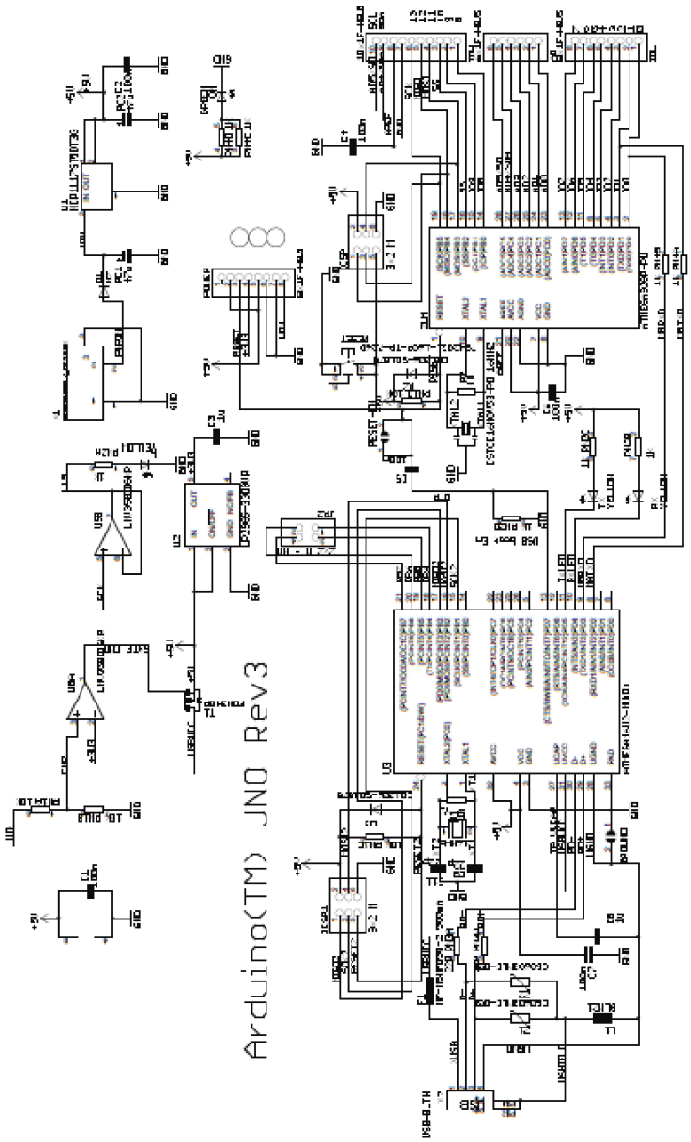


Figure 2.6: Arduino UNO R3 open-source schematic. (Figure adapted and used with permission of the Arduino Team (CC BY-NC-SA) www.arduino.cc.)

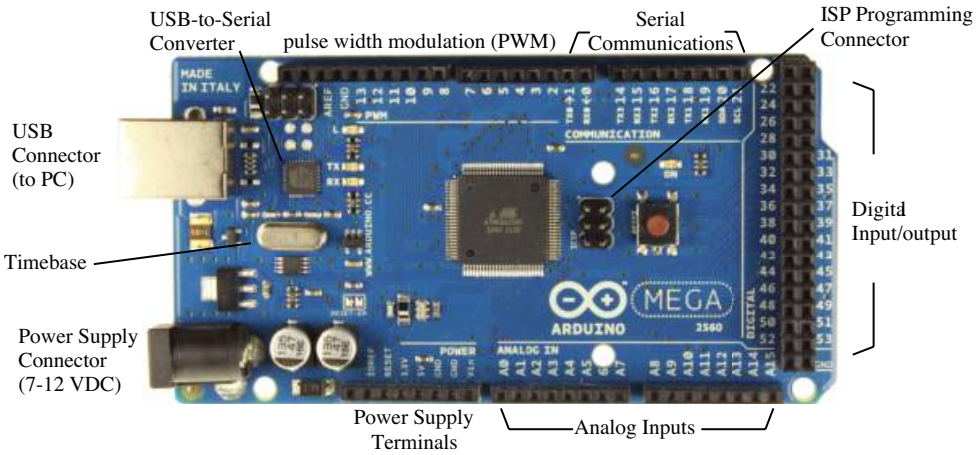


Figure 2.7: Arduino Mega2560 layout. (Figure adapted and used with permission of Arduino Team (CC BY-NC-SA) www.arduino.cc.)

2.5 ARDUINO MEGA 2560 R3 PROCESSING BOARD

The Arduino Mega 2560 REV3 (R3) processing board is illustrated in Figure 2.7. Working clockwise from the left, the board is equipped with a USB connector to allow programming the processor from a host PC. The board may also be programmed using In System Programming (ISP) techniques. A 6-pin ISP programming connector is on the opposite side of the board from the USB connector.

The board is equipped with a USB-to-serial converter to allow compatibility between the host PC and the serial communications systems aboard the ATmega2560 processor. The Mega 2560 R3 is also equipped with several small surface-mount LEDs to indicate TX and RX and an extra LED for project use. The header strip at the top of the board provides access to PWM signals and serial communications. The header strip at the right side of the board provides access to multiple digital input/output pins. The bottom of the board provides analog inputs for the ADC system and power supply terminals. Finally, the external power supply connector is provided at the bottom left corner of the board. The header strips conveniently mate with an Arduino shield (to be discussed shortly) to extend the features of the host processor.

2.6 ADVANCED: ARDUINO MEGA 2560 HOST PROCESSOR – THE ATMEGA2560

The host processor for the Arduino Mega 2560 is the Microchip Atmega2560. The “2560” is a 100 pin, surface-mount 8-bit microcontroller. The architecture is based on the Reduced Instruction Set Computer (RISC) concept which allows the processor to complete 16 million

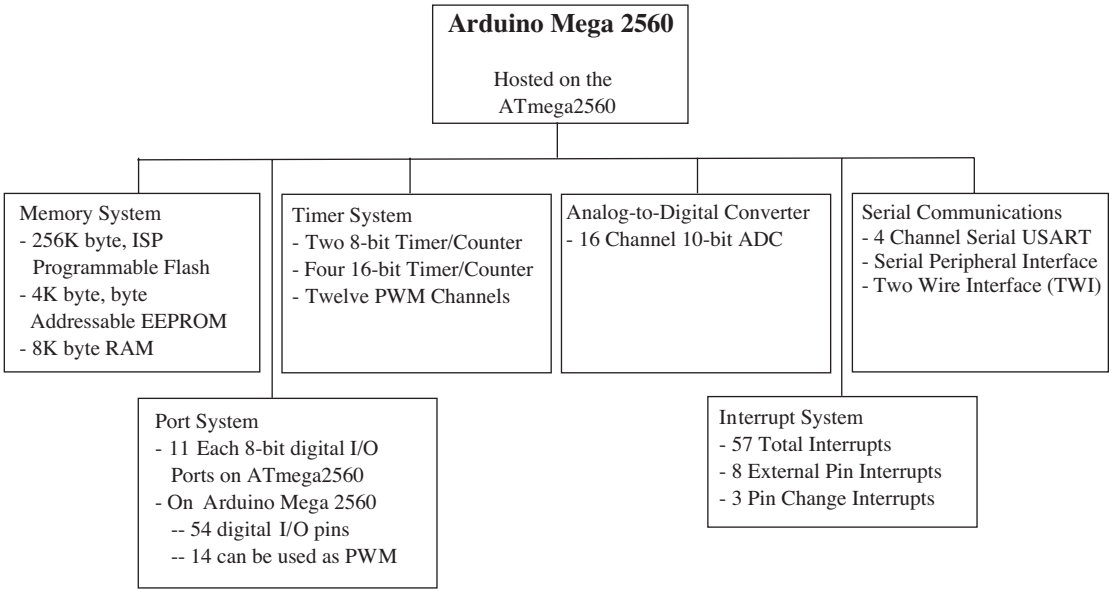


Figure 2.8: Arduino mega 2560 systems.

instructions per second (MIPS) when operating at 16 MHz. The "2560" is equipped with a wide variety of features as shown in Figure 2.8. The features may be conveniently categorized into the following systems:

- memory system,
- port system,
- timer system,
- analog-to-digital converter (ADC),
- interrupt system, and
- serial communications.

2.6.1 ARDUINO MEGA 2560 /ATMEGA2560 HARDWARE FEATURES

The Arduino Mega 2560's processing power is provided by the ATmega2560. The pin out diagram and block diagram for this processor are provided in Figures 2.9 and 2.10. In this section, we provide a brief overview of the systems onboard the processor.

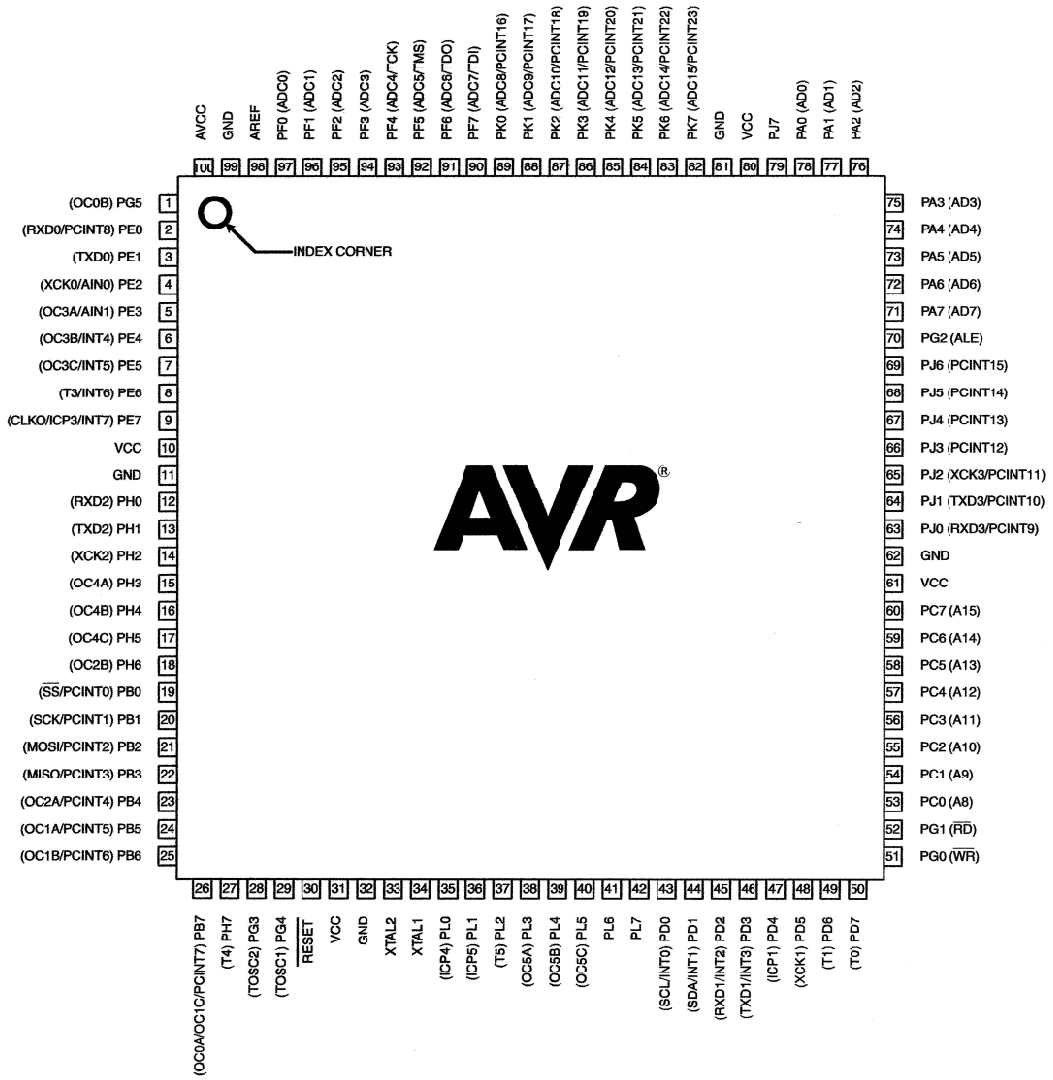


Figure 2.9: ATmega2560 pin out. (Figure used with permission of Microchip, Incorporated.)

2.6.2 ATMEGA2560 MEMORY

The ATmega2560 is equipped with three main memory sections: flash electrically erasable programmable read only memory (EEPROM), static random access memory (SRAM), and byte-addressable EEPROM for data storage.

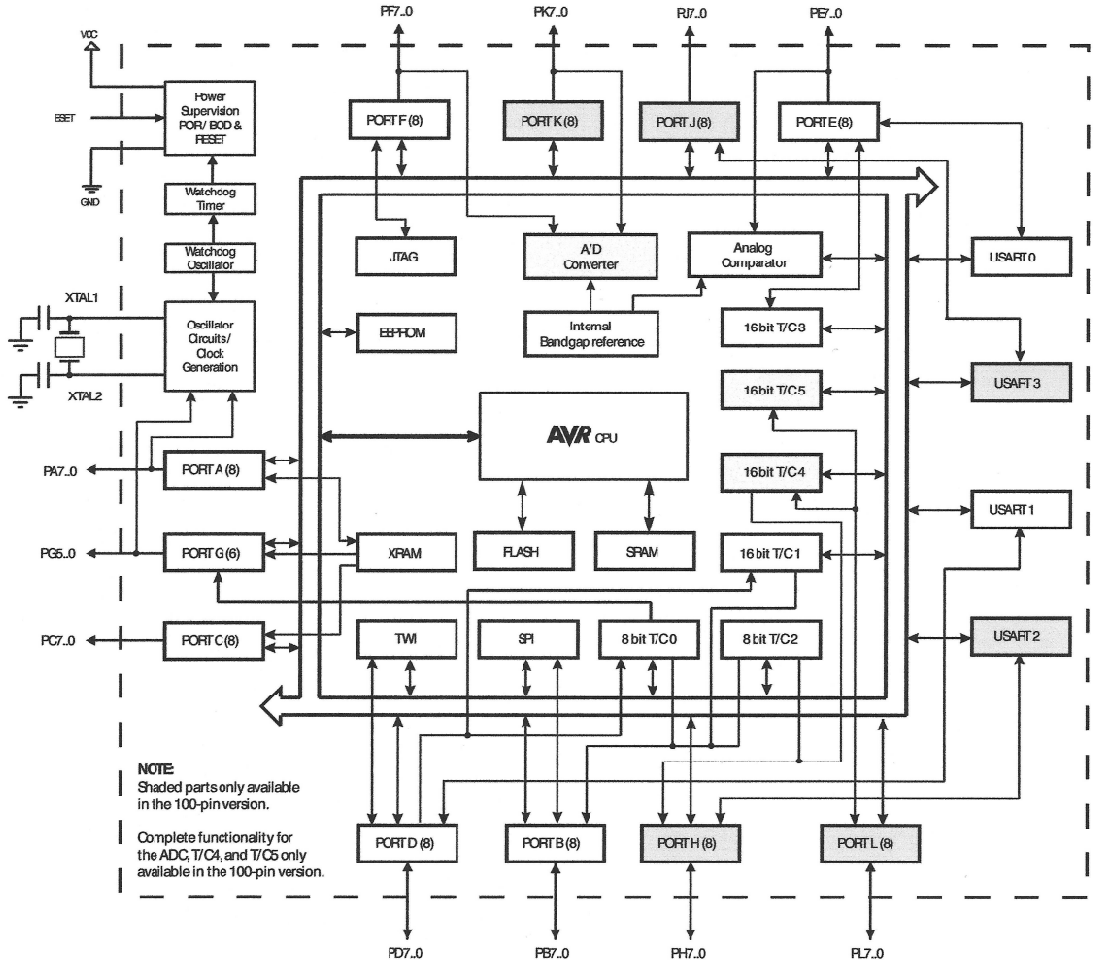


Figure 2.10: ATmega2560 block diagram. (Figure used with permission of Microchip, Incorporated.)

2.6.2.1 ATmega2560 In-System Programmable Flash EEPROM

Bulk programmable flash EEPROM is used to store programs. It can be erased and programmed as a single unit. Also, should a program require a large table of constants, it may be included as a global variable within a program and programmed into flash EEPROM with the rest of the program. Flash EEPROM is nonvolatile meaning memory contents are retained when microcontroller power is lost. The ATmega2560 is equipped with 256K bytes of onboard reprogrammable flash memory.

2.6.2.2 ATmega2560 Byte-Addressable EEPROM

Byte-addressable EEPROM memory is used to permanently store and recall variables during program execution. It too is nonvolatile. It is especially useful for logging system malfunctions and fault data during program execution. It is also useful for storing data that must be retained during a power failure but might need to be changed periodically. Examples where this type of memory is used are found in applications to store system parameters, electronic lock combinations, and automatic garage door electronic unlock sequences. The ATmega2560 is equipped with 4096 bytes of byte-addressable EEPROM.

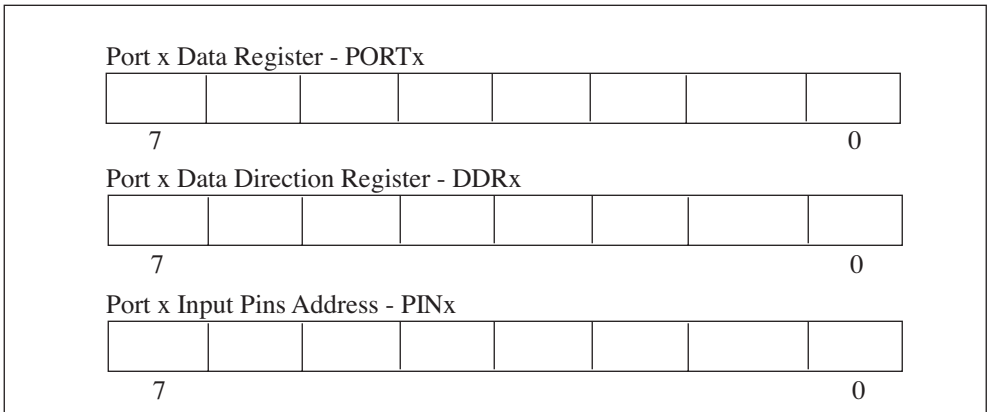
2.6.2.3 ATmega2560 Static Random Access Memory (SRAM)

Static RAM memory is volatile. That is, if the microcontroller loses power, the contents of SRAM memory are lost. It can be written to and read from during program execution. The ATmega2560 is equipped with 8 K bytes of SRAM. A small portion of the SRAM is set aside for the general-purpose registers used by the processor and also for the input/output and peripheral subsystems aboard the microcontroller. During program execution, RAM is used to store global variables, support dynamic memory allocation of variables, and to provide a location for the stack.

2.6.3 ATMEGA2560 PORT SYSTEM

The Microchip ATmega2560 is equipped with eleven, 8-bit general purpose, digital input/output (I/O) ports designated:

- PORTA (8 bits, PORTA[7:0])
- PORTB (8 bits, PORTB[7:0])
- PORTC (7 bits, PORTC[7:0])
- PORTD (8 bits, PORTD[7:0])
- PORTE (8 bits, PORTE[7:0])
- PORTF (8 bits, PORTF[7:0])
- PORTG (7 bits, PORTG[7:0])
- PORTH (8 bits, PORTH[7:0])
- PORTJ (8 bits, PORTJ[7:0])
- PORTK (8 bits, PORTK[7:0])
- PORTL (7 bits, PORTL[7:0])



(a) Port Associated Registers

DDxn	PORTxn	I/O	Comment	Pullup
0	0	input	Tri-state (Hi-Z)	No
0	1	input	Source Current if Externally Pulled Low	Yes
1	0	output	Output Low (Sink)	No
1	1	output	Output High (Source)	No

x: Port Designator (B, C, D)
n: Pin Designator (0 – 7)

(b) Port Pin Configuration

Figure 2.11: ATmega2560 port configuration registers.

All of the ports also have alternate functions which will be described later. In this section, we concentrate on the basic digital I/O port features.

As shown in Figure 2.11, each port has three registers associated with it:

- Data Register PORTx—used to write output data to the port,
- Data Direction Register DDRx—used to set a specific port pin to either output (1) or input (0), and
- Input Pin Address PINx—used to read input data from the port.

Figure 2.11b describes the settings required to configure a specific port pin to either input or output. If selected for input, the pin may be selected for either an input pin or to operate in

the high impedance (Hi-Z) mode. If selected for output, the pin may be further configured for either logic low or logic high.

Port pins are usually configured at the beginning of a program for either input or output and their initial values are then set. Usually all eight pins for a given port are configured simultaneously.

2.6.4 ATMEGA2560 INTERNAL SYSTEMS

In this section, we provide a brief overview of the internal features of the ATmega2560. It should be emphasized that these features are the internal systems contained within the confines of the microcontroller chip. These built-in features allow complex and sophisticated tasks to be accomplished by the microcontroller.

2.6.4.1 ATmega2560 Time Base

The microcontroller is a complex synchronous state machine. It responds to program steps in a sequential manner as dictated by a user-written program. The microcontroller sequences through a predictable fetch-decode-execute sequence. Each unique assembly language program instruction issues a series of signals to control the microcontroller hardware to accomplish instruction-related operations.

The speed at which a microcontroller sequences through these actions is controlled by a precise time base called the clock. The clock source is routed throughout the microcontroller to provide a time base for all peripheral subsystems. The ATmega2560 may be clocked internally using a user-selectable resistor capacitor (RC) time base or it may be clocked externally. The RC internal time base is selected using programmable fuse bits. You may choose an internal fixed clock operating frequency of 128 kHz or 8 MHz. The clock frequency may be prescaled by a number of different clock division factors (1, 2, 4, etc.).

To provide for a wider range of frequency selections an external time source may be used. The external time sources, in order of increasing accuracy and stability, are an external RC network, ceramic resonator, or crystal oscillator. The system designer chooses the time base frequency and clock source device appropriate for the application at hand. Generally speaking, if the microcontroller will be interfaced to external peripheral devices either a ceramic resonator or a crystal oscillator should be used as a time base.

2.6.4.2 ATmega2560 Timing Subsystem

The ATmega2560 is equipped with a complement of timers which allows the user to generate a precision output signal, measure the characteristics (period, duty cycle, frequency) of an incoming digital signal or count external events. Specifically, the ATmega2560 is equipped with two 8-bit timer/counters and four 16-bit timer/counters.

2.6.4.3 Pulse Width Modulation Channels

A PWM signal is characterized by a fixed frequency and a varying duty cycle. Duty cycle is the percentage of time a repetitive signal is logic high during the signal period. It may be formally expressed as:

$$dutycycle[\%] = (on\ time/period) \times (100\%).$$

The ATmega2560 is equipped with four 8-bit PWM channels and 12 PWM channels with programmable resolution. The PWM channels coupled with the flexibility of dividing the time base down to different PWM subsystem clock source frequencies allow the user to generate a wide variety of PWM signals: from relatively high-frequency low-duty cycle signals to relatively low-frequency high-duty cycle signals.

PWM signals are used in a wide variety of applications including controlling the position of a servo motor and controlling the speed of a DC motor.

2.6.4.4 ATmega2560 Serial Communications

The ATmega2560 is equipped with a host of different serial communication subsystems including the Universal Synchronous and Asynchronous Serial Receiver and Transmitter (USART), the serial peripheral interface (SPI), and the Two-wire Serial Interface. What all of these systems have in common is the serial transmission of data. In a serial communications transmission, serial data is sent a single bit at a time from transmitter to receiver.

ATmega2560 Serial USART The serial USART may be used for full duplex (two-way) communication between a receiver and transmitter. This is accomplished by equipping the ATmega2560 with independent hardware for the transmitter and receiver. The USART is typically used for asynchronous communication. That is, there is not a common clock between the transmitter and receiver to keep them synchronized with one another. To maintain synchronization between the transmitter and receiver, framing start and stop bits are used at the beginning and end of each data byte in a transmission sequence.

The ATmega2560 USART is quite flexible. It has the capability to be set to a variety of data transmission rates known as the Baud (bits per second) rate. The USART may also be set for data bit widths of 5–9 bits with one or two stop bits. Furthermore, the ATmega2560 is equipped with a hardware generated parity bit (even or odd) and parity check hardware at the receiver. A single parity bit allows for the detection of a single bit error within a byte of data. The USART may also be configured to operate in a synchronous mode.

ATmega2560 Serial Peripheral Interface-SPI The ATmega2560 Serial Peripheral Interface (SPI) can also be used for two-way serial communication between a transmitter and a receiver. In the SPI system, the transmitter and receiver share a common clock source. This requires an additional clock line between the transmitter and receiver but allows for higher data transmission rates as compared to the USART.

The SPI may be viewed as a synchronous 16-bit shift register with an 8-bit half residing in the transmitter and the other 8-bit half residing in the receiver. The transmitter is designated the master since it is providing the synchronizing clock source between the transmitter and the receiver. The receiver is designated as the slave.

ATmega2560 Two-Wire Serial Interface-TWI The TWI subsystem allows the system designer to network-related devices (microcontrollers, transducers, displays, memory storage, etc.) together into a system using a two-wire interconnecting scheme. The TWI allows a maximum of 128 devices to be interconnected together. Each device has its own unique address and may both transmit and receive over the two-wire bus at frequencies up to 400 kHz. This allows the device to freely exchange information with other devices in the network within a small area.

2.6.4.5 ATmega2560 Analog to Digital Converter-ADC

The ATmega2560 is equipped with a 16-channel analog to digital converter (ADC) subsystem. The ADC converts an analog signal from the outside world into a binary representation suitable for use by the microcontroller. The ATmega2560 ADC has 10-bit resolution. This means that an analog voltage between 0 and 5 V will be encoded into one of 1024 binary representations between $(000)_{16}$ and $(3FF)_{16}$. This provides the ATmega2560 with a voltage resolution of approximately 4.88 mV.

2.6.4.6 ATmega2560 Interrupts

The normal execution of a program follows a designated sequence of instructions. However, sometimes this normal sequence of events must be interrupted to respond to high-priority faults and status both inside and outside the microcontroller. When these higher-priority events occur, the microcontroller suspends normal operation and executes event-specific actions contained within an ISR. Once the higher-priority event has been serviced via the ISR, the microcontroller returns and continues processing the normal program.

The ATmega2560 is equipped with a complement of 57 interrupt sources. Eight interrupts are provided for external interrupt sources. Also, the ATmega2560 is equipped with three pin change interrupts. The remaining interrupts support the efficient operation of peripheral subsystems aboard the microcontroller.

2.7 ARDUINO MEGA 2560 OPEN SOURCE SCHEMATIC

The entire line of Arduino products is based on the visionary concept of open-source hardware and software. That is, hardware and software developments are openly shared among users to stimulate new ideas and advance the Arduino concept. In keeping with the Arduino concept, the Arduino team openly shares the schematic of the Arduino Mega 2560 processing board. It is available for download at `www.arduino.cc`.

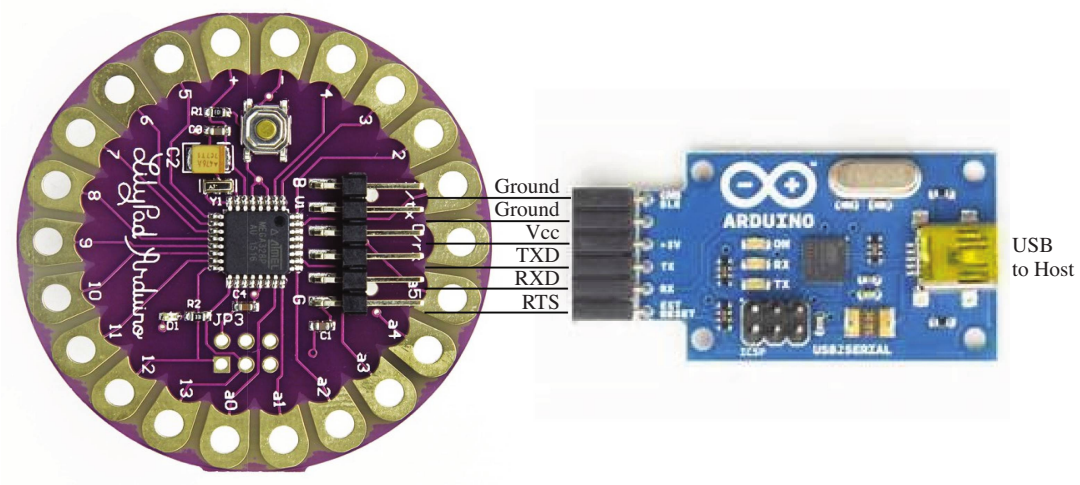


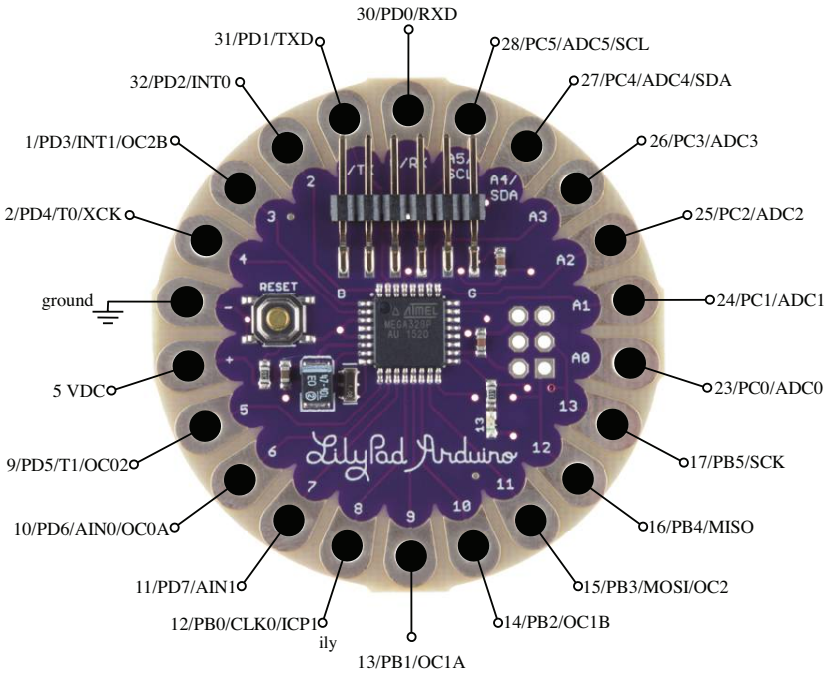
Figure 2.12: LilyPad Arduino with USB 2 serial converter. (Figure adapted and used with permission of Arduino Team (CC BY-NC-SA) www.arduino.cc.)

2.8 LILYPAD ARDUINO

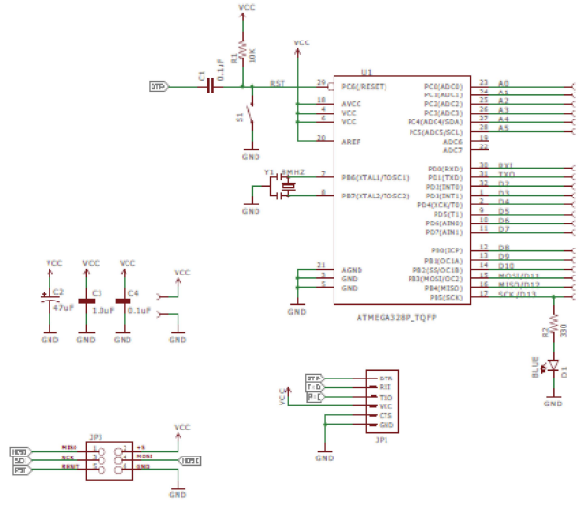
The LilyPad Arduino was developed by Dr. Leah Buechley, Ph.D. for use in wearable electronic fashions (e-textiles). The LilyPad may be sewn into clothing. Its complement of support hardware such as a power supply and input and output devices are connected to the LilyPad via conductive thread. Dr. Buechley has written several books about this fascinating synergy between electronics and the fashion world. *Sew Electric—A Collection of DIY Projects that Combine Fabric, Electronics, and Programming* co-written by Buechley, Kanjun Qiu, and Sonja de Boer provide a number of LilyPad based do it yourself (DIY) projects.

2.8.1 ADVANCED: LILYPAD PROCESSOR

A wearable LilyPad processor is available in several different configurations. The LilyPad Arduino Main Board featured here hosts an Microchip ATmega328 processor shown in Figure 2.12. Note the use of a USB 2 serial converter and an USB 2 cable (Type A male to mini Type B female) to connect to the host PC. This is the same processor onboard the UNO R3. The LilyPad mainboard operates at 8 MHz and is equipped with 14 general purpose input/output pins, six PWM pins, and six analog input/output pins. The LilyPad pinout and schematic are shown in Figure 2.13.



(a) LilyPad Arduino Pinout



(b) LilyPad Arduino Schematic

Figure 2.13: LilyPad Arduino schematic. (Figure adapted and used with permission of Arduino Team (CC BY-NC-SA) (www.arduino.cc).)

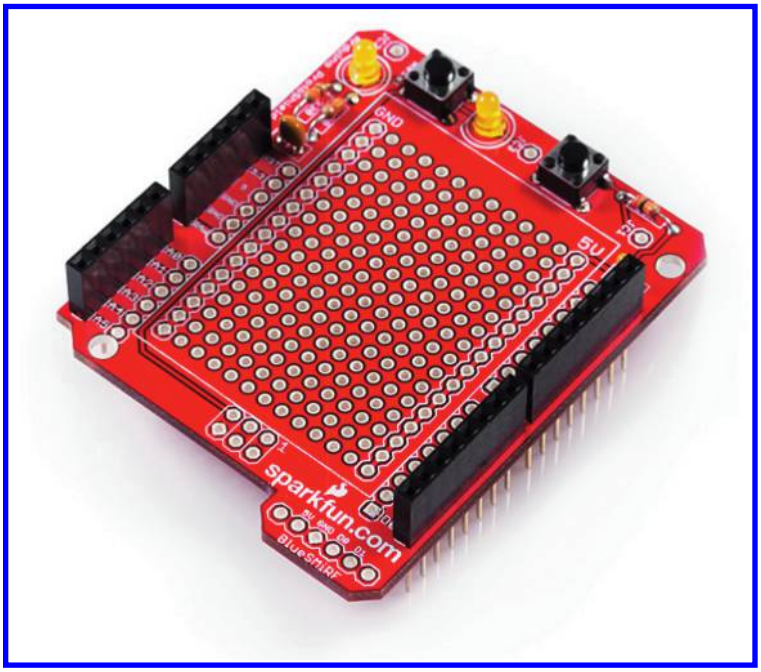


Figure 2.14: Arduino shield. (Used with permission from SparkFun Electronics (CC BY-NC-SA).)

2.9 OTHER ARDUINO-BASED PLATFORMS

There is a wide variety of Arduino-based platforms. They are categorized by function and feature into:

- entry level,
- enhanced features,
- internet of Things,
- education, and
- wearable.

Information on specific products within each category are available at `www.ardunio.cc`.

2.10 EXTENDING THE HARDWARE FEATURES OF THE ARDUINO PLATFORMS

Additional features and external hardware may be added to selected Arduino platforms by using a daughter card concept. The daughter card is called an Arduino Shield, as shown in Figure 2.14.

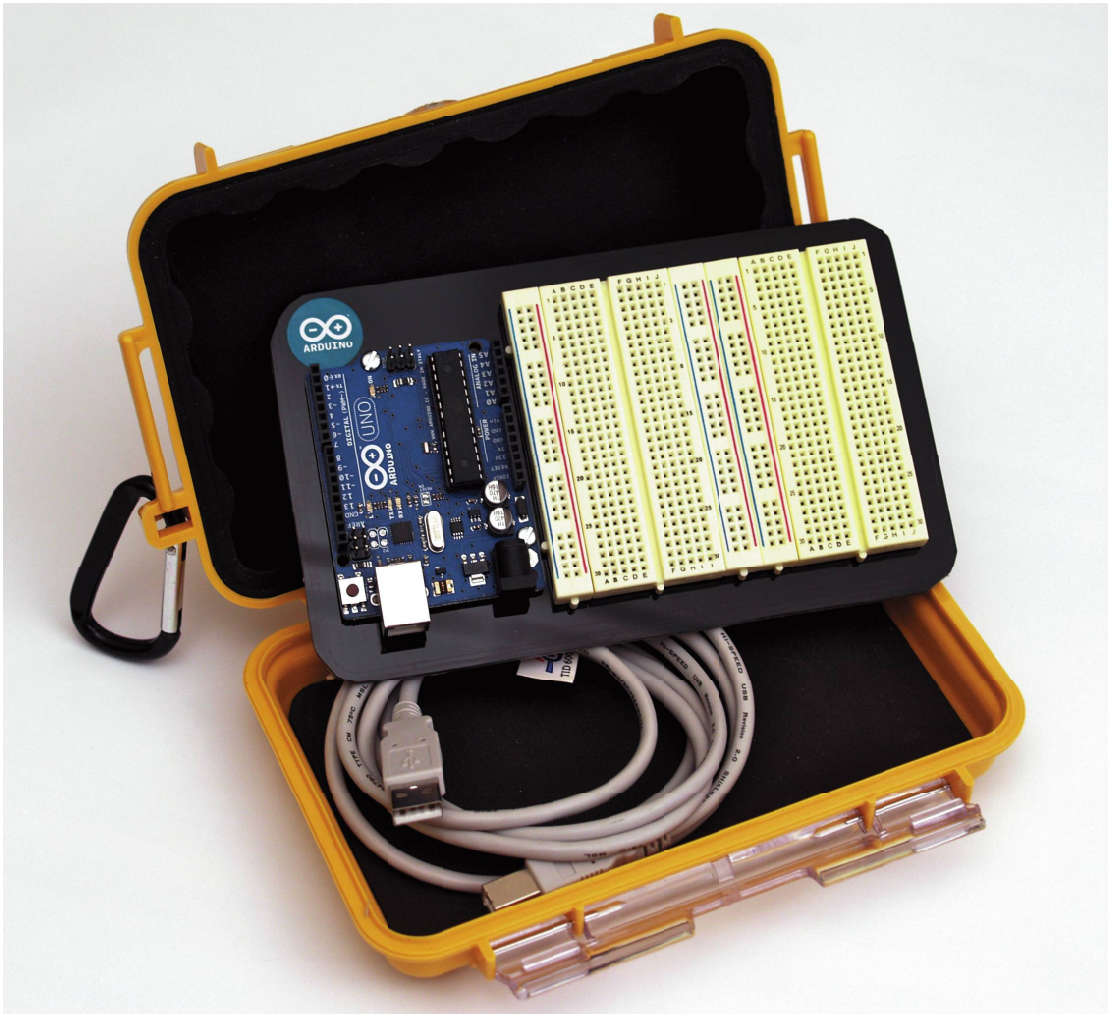


Figure 2.15: Arduino hardware studio.

The shield mates with the header pins on the Arduino board. The shield provides a small fabrication area, a processor reset button, and a general use pushbutton and two LEDs. A large number of shields have been developed to provide extended specific features (e.g., motor control, communications, etc.) for the Arduino boards. We discuss specific shields in Chapter 3.

2.11 APPLICATION: ARDUINO HARDWARE STUDIO

Much like an artist uses a sketch box, we will use an Arduino Hardware Studio throughout the book to develop projects. In keeping with the DIY spirit of Arduino, we have constructed the Studio using readily available off-the-shelf products as shown in Figure 2.15. The Studio includes the following:

- a yellow Pelican Micro Case #1040,
- an Arduino UNO R3 evaluation board,
- two Jameco JE21 3.3 x 2.1 inch solderless breadboards, and
- one piece of black plexiglass.

We purposely have not provided any construction details. Instead, we encourage you to use your own imagination to develop and construct your own Arduino Hardware Studio.

2.12 APPLICATION: AUTONOMOUS MAZE NAVIGATING ROBOT

An autonomous, maze-navigating robot is equipped with sensors to detect the presence of maze walls and navigate about the maze. The robot has no prior knowledge about the maze configuration. It uses the sensors and an onboard algorithm to determine the robot's next move. The overall goal is to navigate from the starting point of the maze to the end point as quickly as possible without bumping into maze walls, as shown in Figure 2.16. Maze walls are usually painted white to provide a good, light-reflective surface, whereas, the maze floor is painted matte black to minimize light reflections.

It would be helpful to review the fundamentals of robot steering and motor control. Figure 2.17 illustrates the fundamental concepts. Robot steering is dependent upon the number of powered wheels and whether the wheels are equipped with unidirectional or bidirectional control. Additional robot steering configurations are possible. An H-bridge is typically required for bidirectional control of a DC motor. We discuss the H-bridge in the next chapter.

We equip the Adafruit Mini Round Robot (#3216) with an Arduino UNO R3 for maze navigation; see Figure 2.18. The robot is controlled by two 6 VDC motors which independently drive a left and right wheel. A third non-powered drag ball provides tripod stability for the robot. We also equip the platform with three Sharp GP2Y0A21YKOF IR sensors, as shown in Figure 2.19. The sensors are available from SparkFun Electronics (`www.sparkfun.com`). We mount the sensors on a bracket constructed from thin aluminum. Dimensions for the bracket are provided in the figure. Alternatively, the IR sensors may be mounted to the robot platform using "L" brackets available from a local hardware store. The characteristics of the sensor are provided in Figure 2.20.

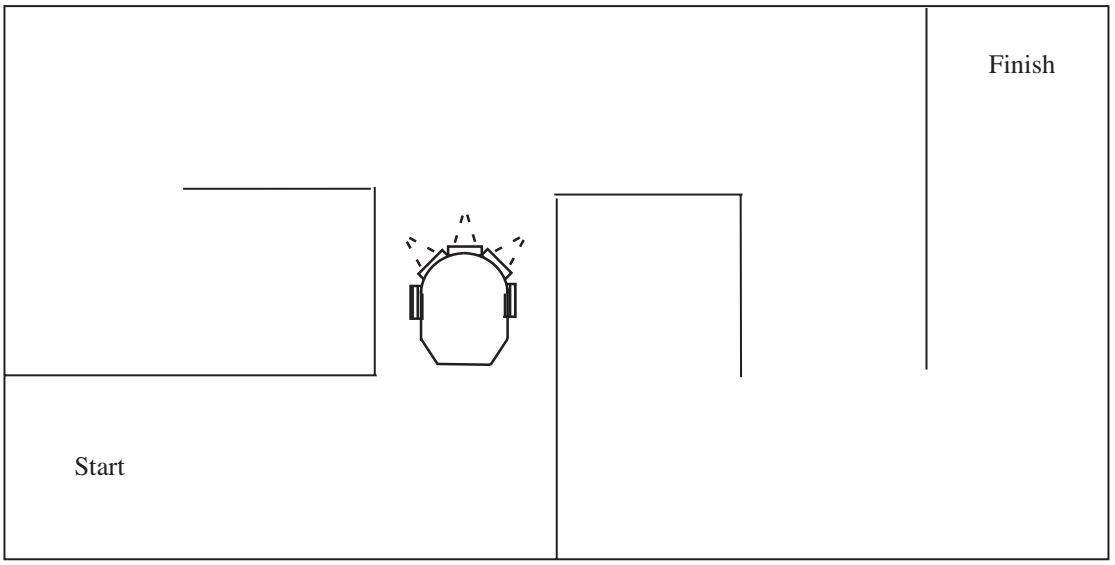


Figure 2.16: Autonomous robot within maze.

2.12.1 REQUIREMENTS

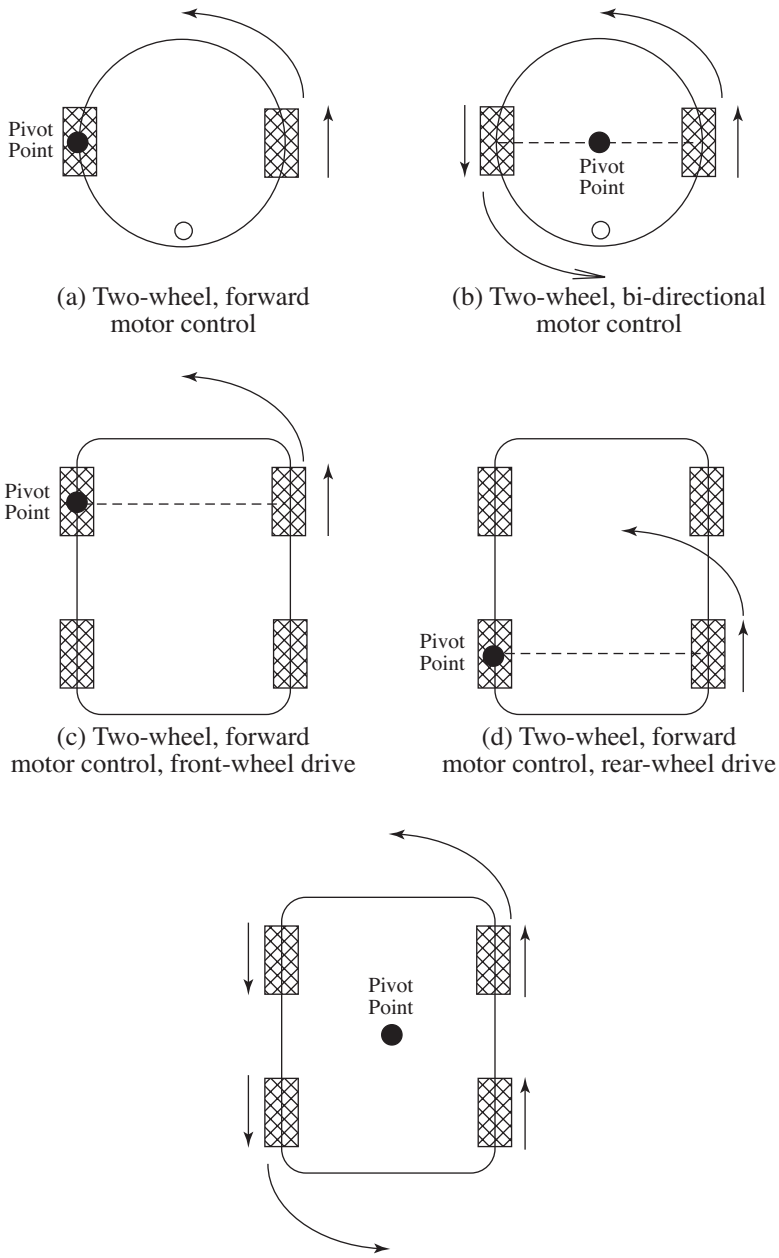
The requirements for this project are simple, the robot must autonomously navigate through the maze without touching maze walls.

2.12.2 CIRCUIT DIAGRAM

The circuit diagram for the robot is provided in Figure 2.21. The three IR sensors (left, middle, and right) are mounted on the leading edge of the robot to detect maze walls. The output from the sensors is fed to three Arduino UNO R3 ADC channels (ANALOG IN 0–2). The robot motors will be driven by PWM channels (PWM: DIGITAL 11 and PWM: DIGITAL 10).

The Arduino UNO R3 is interfaced to the motors via a Darlington NPN transistor (TIP120) with enough drive capability to handle the maximum current requirements of the motor (1.5 A hard stall current). The robot is powered by a 9 VDC power supply. Three 1N4001 diodes are placed in series with the motor to reduce the motor supply voltage to approximately 6.9 VDC. The 9 VDC supply is also fed to a 5 VDC voltage regulator to power the Arduino UNO R3. To save on battery expense, it is recommended to use a 9 VDC, 2A rated inexpensive, wall-mount power supply to provide power to the onboard 5 VDC voltage regulator. A power umbilical of braided wire may be used to provide power to the robot while navigating about the maze.

Structure Chart. The structure chart for the robot project is provided in Figure 2.22.



(a) Two-wheel, forward motor control

(b) Two-wheel, bi-directional motor control

(c) Two-wheel, forward motor control, front-wheel drive

(d) Two-wheel, forward motor control, rear-wheel drive

(e) Four-wheel, bi-directional motor control

Figure 2.17: Robot control configurations.

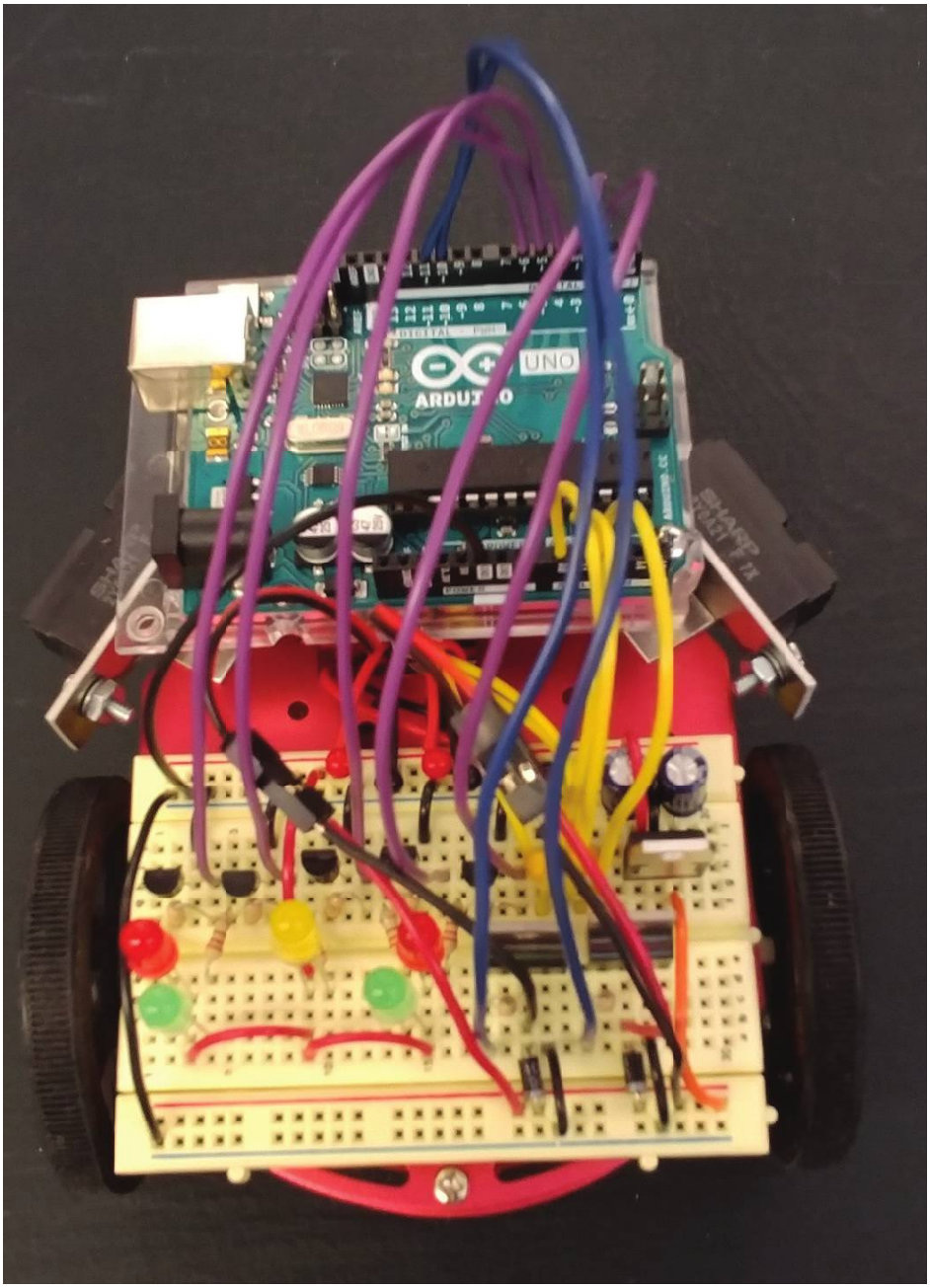
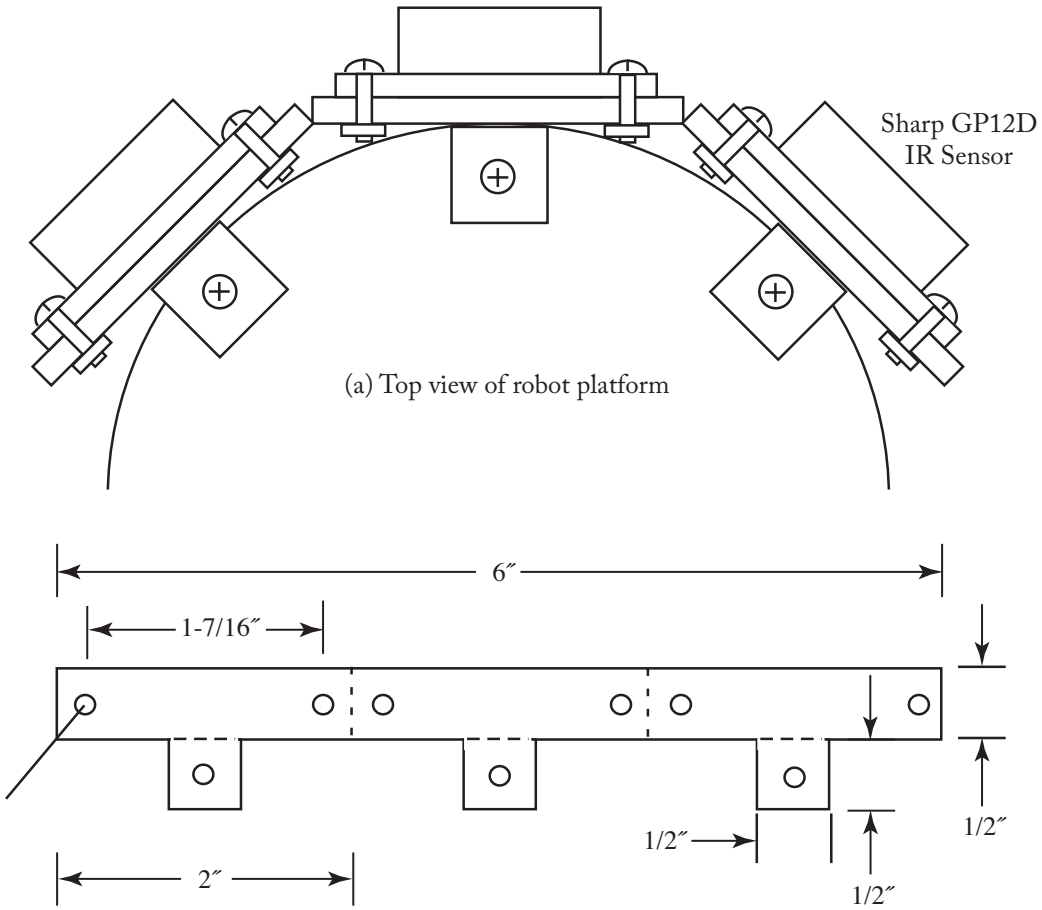


Figure 2.18: Adafruit mini round robot.



(b) Construction details for sensor bracket

Figure 2.19: Mini round robot platform modified with three IR sensors.

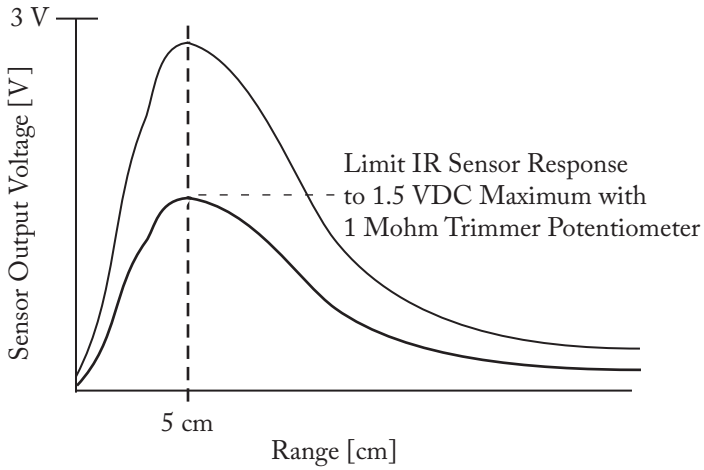


Figure 2.20: Sharp GP2Y0A21YKOF IR sensor profile.

UML Activity Diagrams. The UML activity diagram for the robot is provided in Figure 2.23.

2.12.3 MINI ROUND ROBOT CONTROL ALGORITHM

In this section, we provide the basic framework for the robot control algorithm. The control algorithm will read the IR sensors attached to the Arduino UNO R3 ANALOG IN (pins 0–2). In response to the wall placement detected, it will render signals to turn the robot to avoid the maze walls. Provided in Figure 2.24 is a truth table that shows all possibilities of maze placement that the robot might encounter. A detected wall is represented with a logic one. An asserted motor action is also represented with a logic one.

The robot motors may only be moved in the forward direction. We review techniques to provide bi-directional motor control in Chapter 3. To render a left turn, the left motor is stopped and the right motor is asserted until the robot completes the turn. To render a right turn, the opposite action is required.

The task in writing the control algorithm is to take the UML activity diagram provided in Figure 2.23 and the actions specified in the robot action truth table (Figure 2.24) and transform both into an Arduino sketch. This may seem formidable but we take it a step at a time.

The control algorithm begins with Arduino UNO R3 pin definitions. Variables are then declared for the readings from the three IR sensors. The two required Arduino functions follow: setup() and loop(). In the setup() function, Arduino UNO R3 pins are declared as output. The loop() begins by reading the current value of the three IR sensors. The 512 value corresponds to a particular IR sensor range. This value may be adjusted to change the range at which the maze wall is detected. The read of the IR sensors is followed by an eight-part if-else if statement. The

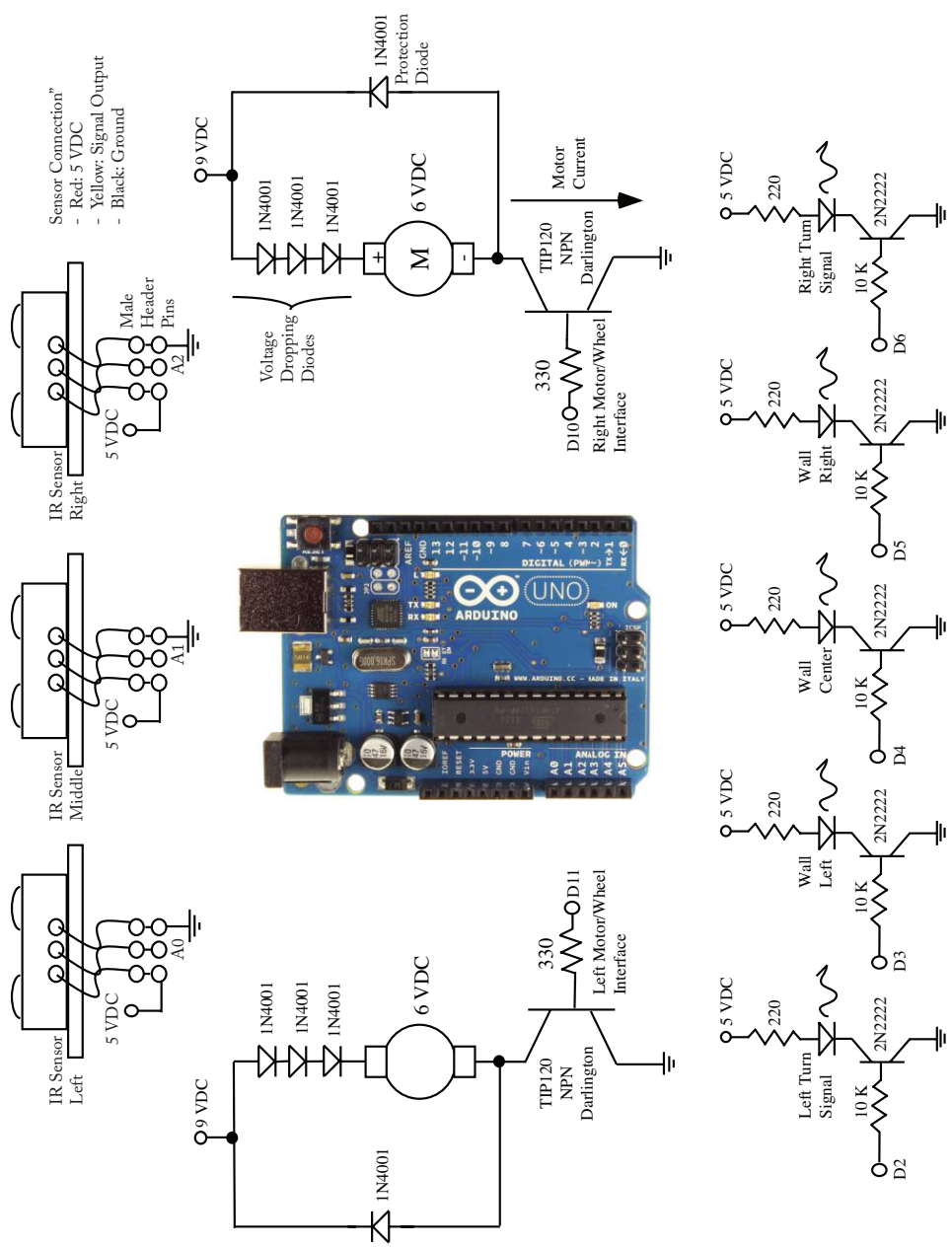


Figure 2.21: Robot circuit diagram. (UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA) (www.arduino.cc).)

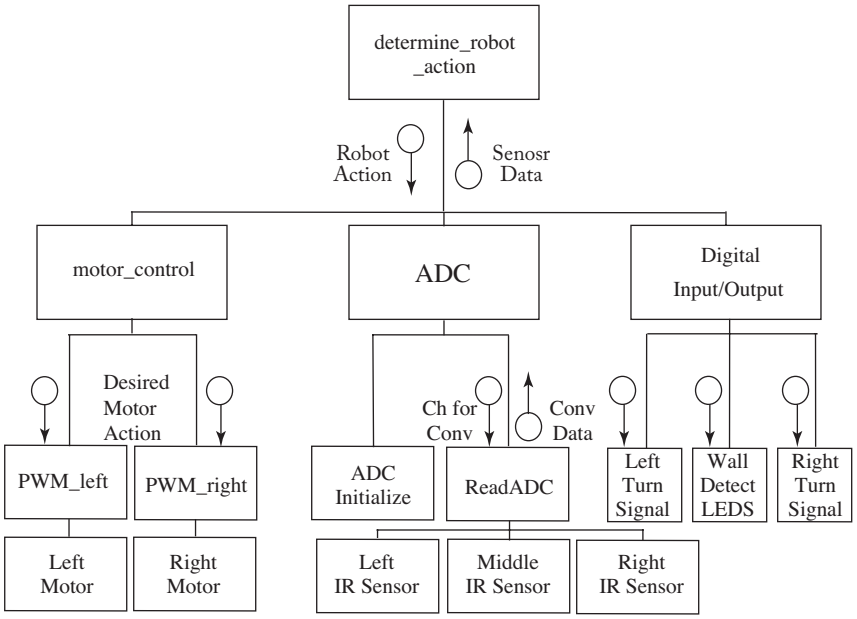
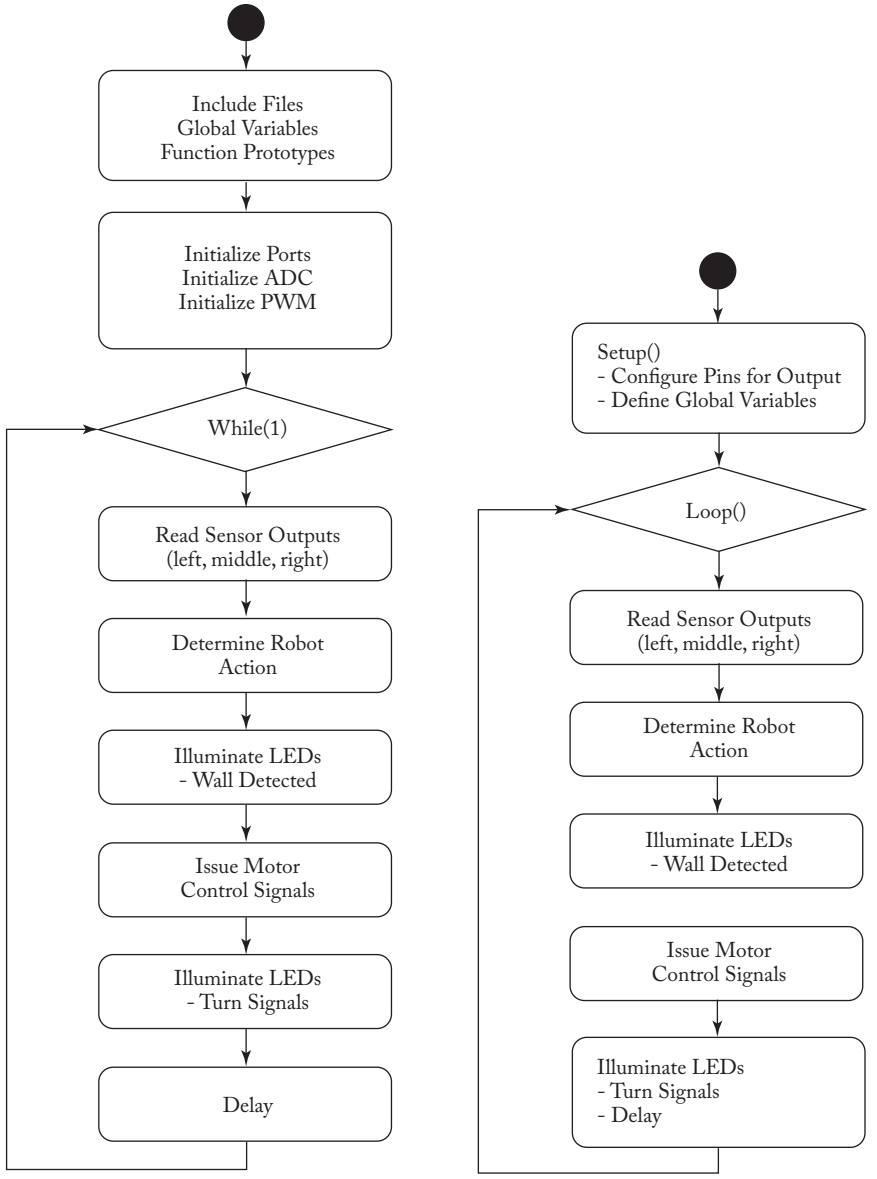


Figure 2.22: Robot structure diagram.

statement contains a part for each row of the truth table provided in Figure 2.24. For a given configuration of sensed walls, the appropriate wall detection LEDs are illuminated followed by commands to activate the motors (analogWrite) and illuminate the appropriate turn signals. The analogWrite command issues a signal from 0–5 VDC by sending a constant from 0–255 using PWM techniques. The turn signal commands provide to actions: the appropriate turns signals are flashed and a 1.5 s total delay is provided. This provides the robot 1.5 s to render a turn. This delay may need to be adjusted during the testing phase.

```
//*************************************************************************
                                    //analog input pins
#define left_IR_sensor   A0         //analog pin - left IR sensor

#define center_IR_sensor A1         //analog pin - center IR sensor
#define right_IR_sensor  A2         //analog pin - right IR sensor

                                    //digital output pins
                                    //LED indicators - wall detectors
#define wall_left         3         //digital pin - wall_left
#define wall_center       4         //digital pin - wall_center
#define wall_right        5         //digital pin - wall_right
```



(a) UML for C Programming

(b) UML for Arduino Programming

Figure 2.23: Robot UML activity diagram.

	Left Sensor	Middle Sensor	Right Sensor	Wall Left	Wall Middle	Wall Right	Left Motor	Right Motor	Left Signal	Right Signal	Comments
0	0	0	0	0	0	0	1	1	0	0	Forward
1	0	0	1	0	0	1	1	1	0	0	Forward
2	0	1	0	0	1	0	1	0	0	1	Right
3	0	1	1	0	1	1	0	1	1	0	Left
4	1	0	0	1	0	0	1	1	0	0	Forward
5	1	0	1	1	0	1	1	1	0	0	Forward
6	1	1	0	1	1	0	1	0	0	1	Right
7	1	1	1	1	1	1	1	0	0	1	Right

Figure 2.24: Truth table for robot action.

```
                                    //LED indicators - turn signals
#define left_turn_signal  2         //digital pin - left_turn_signal
#define right_turn_signal 6         //digital pin - right_turn_signal

                                    //motor outputs
#define left_motor       11         //digital pin - left_motor
#define right_motor      10         //digital pin - right_motor


int left_IR_sensor_value;           //declare
variable for left IR sensor
int center_IR_sensor_value;         //declare
variable for center IR sensor
int right_IR_sensor_value;          //declare
variable for right IR sensor

void setup()
  {
                                    //LED indicators - wall detectors
  pinMode(wall_left,   OUTPUT);     //configure pin 1 for digital output
  pinMode(wall_center, OUTPUT);     //configure pin 2 for digital output
  pinMode(wall_right,  OUTPUT);     //configure pin 3 for digital output

                                    //LED indicators - turn signals
  pinMode(left_turn_signal,OUTPUT); //configure pin 0 for digital output
  pinMode(right_turn_signal,OUTPUT); //configure pin 4 for digital output
```

```
                                    //motor outputs - PWM
  pinMode(left_motor,  OUTPUT);     //configure pin 11 for digital output
  pinMode(right_motor, OUTPUT);     //configure pin 10 for digital output
  }

void loop()
  {
                                    //read analog output from IR sensors
   left_IR_sensor_value   = analogRead(left_IR_sensor);
   center_IR_sensor_value = analogRead(center_IR_sensor);
   right_IR_sensor_value  = analogRead(right_IR_sensor);

   //robot action table row 0
   if((left_IR_sensor_value < 512)&&(center_IR_sensor_value < 512)&&
      (right_IR_sensor_value < 512))
     {
                                          //wall detection LEDs
     digitalWrite(wall_left,   LOW);      //turn LED off
     digitalWrite(wall_center, LOW);      //turn LED off
     digitalWrite(wall_right,  LOW);      //turn LED off
                                          //motor control
     analogWrite(left_motor,  128);
       //0 (off) to 255 (full speed)
     analogWrite(right_motor, 128);
       //0 (off) to 255 (full speed)
                                          //turn signals
     digitalWrite(left_turn_signal,  LOW);//turn LED off
     digitalWrite(right_turn_signal, LOW);//turn LED off
     delay(500);                          //delay 500 ms
     digitalWrite(left_turn_signal,  LOW);//turn LED off
     digitalWrite(right_turn_signal, LOW);//turn LED off
     delay(500);                          //delay 500 ms
     digitalWrite(left_turn_signal,  LOW);//turn LED off
     digitalWrite(right_turn_signal, LOW);//turn LED off
     delay(500);                          //delay 500 ms
     digitalWrite(left_turn_signal,  LOW);//turn LED off
     digitalWrite(right_turn_signal, LOW);//turn LED off
     analogWrite(left_motor, 0);          //turn motor off
```

```
  analogWrite(right_motor,0);              //turn motor off
  }

//robot action table row 1
else if((left_IR_sensor_value < 512)&&(center_IR_sensor_value < 512)&&
        (right_IR_sensor_value > 512))
  {
                                           //wall detection LEDs
  digitalWrite(wall_left,   LOW);          //turn LED off
  digitalWrite(wall_center, LOW);          //turn LED off
  digitalWrite(wall_right,  HIGH);         //turn LED on
                                           //motor control
  analogWrite(left_motor,  128);
    //0 (off) to 255 (full speed)
  analogWrite(right_motor, 128);
    //0 (off) to 255 (full speed)
                                           //turn signals
  digitalWrite(left_turn_signal,  LOW);    //turn LED off
  digitalWrite(right_turn_signal, LOW);    //turn LED off
  delay(500);                              //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);    //turn LED off
  digitalWrite(right_turn_signal, LOW);    //turn LED off
  delay(500);                              //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);    //turn LED off
  digitalWrite(right_turn_signal, LOW);    //turn LED off
  delay(500);                              //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);    //turn LED off
  digitalWrite(right_turn_signal, LOW);    //turn LED off
  analogWrite(left_motor, 0);              //turn motor off
  analogWrite(right_motor,0);              //turn motor off
  }

//robot action table row 2
else if((left_IR_sensor_value < 512)&&(center_IR_sensor_value > 512)&&
        (right_IR_sensor_value < 512))
  {
                                           //wall detection LEDs
  digitalWrite(wall_left,   LOW);          //turn LED off
  digitalWrite(wall_center, HIGH);         //turn LED on
```

```
    digitalWrite(wall_right,  LOW);          //turn LED off
                                             //motor control
    analogWrite(left_motor,  128);
      //0 (off) to 255 (full speed)
    analogWrite(right_motor, 0);
      //0 (off) to 255 (full speed)
                                             //turn signals
    digitalWrite(left_turn_signal,  LOW);    //turn LED off
    digitalWrite(right_turn_signal, HIGH);   //turn LED on
    delay(500);                              //delay 500 ms
    digitalWrite(left_turn_signal,  LOW);    //turn LED off
    digitalWrite(right_turn_signal, LOW);    //turn LED off
    delay(500);                              //delay 500 ms
    digitalWrite(left_turn_signal,  LOW);    //turn LED off
    digitalWrite(right_turn_signal, HIGH);   //turn LED on
    delay(500);                              //delay 500 ms
    digitalWrite(left_turn_signal,  LOW);    //turn LED off
    digitalWrite(right_turn_signal, LOW);    //turn LED off
    analogWrite(left_motor, 0);              //turn motor off
    analogWrite(right_motor,0);              //turn motor off
    }

  //robot action table row 3
  else if((left_IR_sensor_value < 512)&&(center_IR_sensor_value > 512)&&
          (right_IR_sensor_value > 512))
    {
                                             //wall detection LEDs
    digitalWrite(wall_left,   LOW);          //turn LED off
    digitalWrite(wall_center, HIGH);         //turn LED on
    digitalWrite(wall_right,  HIGH);         //turn LED on
                                             //motor control
    analogWrite(left_motor,  0);
      //0 (off) to 255 (full speed)
    analogWrite(right_motor, 128);
      //0 (off) to 255 (full speed)
                                             //turn signals
    digitalWrite(left_turn_signal,  HIGH);   //turn LED on
    digitalWrite(right_turn_signal, LOW);    //turn LED off
    delay(500);                              //delay 500 ms
```

```
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
  digitalWrite(right_turn_signal, LOW);   //turn LED off
  delay(500);                             //delay 500 ms
  digitalWrite(left_turn_signal,  HIGH);  //turn LED on
  digitalWrite(right_turn_signal, LOW);   //turn LED off
  delay(500);                             //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
  digitalWrite(right_turn_signal, LOW);   //turn LED off
  analogWrite(left_motor, 0);             //turn motor off
  analogWrite(right_motor,0);             //turn motor off
  }

//robot action table row 4
else if((left_IR_sensor_value > 512)&&(center_IR_sensor_value < 512)&&
        (right_IR_sensor_value < 512))
  {
                                          //wall detection LEDs
  digitalWrite(wall_left,   HIGH);        //turn LED on
  digitalWrite(wall_center, LOW);         //turn LED off
  digitalWrite(wall_right,  LOW);         //turn LED off
                                          //motor control
  analogWrite(left_motor,  128);
    //0 (off) to 255 (full speed)
  analogWrite(right_motor, 128);
    //0 (off) to 255 (full speed)
                                          //turn signals
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
  digitalWrite(right_turn_signal, LOW);   //turn LED off
  delay(500);                             //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
  digitalWrite(right_turn_signal, LOW);   //turn LED off
  delay(500);                             //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
  digitalWrite(right_turn_signal, LOW);   //turn LED off
  delay(500);                             //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
  digitalWrite(right_turn_signal, LOW);   //turn LED off
  analogWrite(left_motor, 0);             //turn motor off
  analogWrite(right_motor,0);             //turn motor off
```

```
  }

//robot action table row 5
else if((left_IR_sensor_value > 512)&&(center_IR_sensor_value < 512)&&
        (right_IR_sensor_value > 512))
  {
                                        //wall detection LEDs
  digitalWrite(wall_left,   HIGH);      //turn LED on
  digitalWrite(wall_center, LOW);       //turn LED off
  digitalWrite(wall_right,  HIGH);      //turn LED on
                                        //motor control
  analogWrite(left_motor,  128);
    //0 (off) to 255 (full speed)
  analogWrite(right_motor, 128);
    //0 (off) to 255 (full speed)
                                        //turn signals
  digitalWrite(left_turn_signal,  LOW); //turn LED off
  digitalWrite(right_turn_signal, LOW); //turn LED off
  delay(500);                           //delay 500 ms
  digitalWrite(left_turn_signal,  LOW); //turn LED off
  digitalWrite(right_turn_signal, LOW); //turn LED off
  delay(500);                           //delay 500 ms
  digitalWrite(left_turn_signal,  LOW); //turn LED off
  digitalWrite(right_turn_signal, LOW); //turn LED off
  delay(500);                           //delay 500 ms
  digitalWrite(left_turn_signal,  LOW); //turn LED off
  digitalWrite(right_turn_signal, LOW); //turn LED off
  analogWrite(left_motor, 0);           //turn motor off
  analogWrite(right_motor,0);           //turn motor off
  }

//robot action table row 6
else if((left_IR_sensor_value > 512)&&(center_IR_sensor_value > 512)&&
        (right_IR_sensor_value < 512))
  {
                                        //wall detection LEDs
  digitalWrite(wall_left,   HIGH);      //turn LED on
  digitalWrite(wall_center, HIGH);      //turn LED on
  digitalWrite(wall_right,  LOW);       //turn LED off
```

```
                                          //motor control
  analogWrite(left_motor,  128);
    //0 (off) to 255 (full speed)
  analogWrite(right_motor, 0);
    //0 (off) to 255 (full speed)
                                          //turn signals
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
  digitalWrite(right_turn_signal, HIGH);  //turn LED on
  delay(500);                             //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
  digitalWrite(right_turn_signal, LOW);   //turn LED off
  delay(500);                             //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
  digitalWrite(right_turn_signal, HIGH);  //turn LED off
  delay(500);                             //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);   //turn LED OFF
  digitalWrite(right_turn_signal, LOW);   //turn LED OFF
  analogWrite(left_motor, 0);             //turn motor off
  analogWrite(right_motor,0);             //turn motor off
  }

//robot action table row 7
else if((left_IR_sensor_value > 512)&&(center_IR_sensor_value > 512)&&
        (right_IR_sensor_value > 512))
  {
                                          //wall detection LEDs
  digitalWrite(wall_left,   HIGH);        //turn LED on
  digitalWrite(wall_center, HIGH);        //turn LED on
  digitalWrite(wall_right,  HIGH);        //turn LED on
                                          //motor control
  analogWrite(left_motor,  128);
    //0 (off) to 255 (full speed)
  analogWrite(right_motor, 0);
    //0 (off) to 255 (full speed)
                                          //turn signals
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
  digitalWrite(right_turn_signal, HIGH);  //turn LED on
  delay(500);                             //delay 500 ms
  digitalWrite(left_turn_signal,  LOW);   //turn LED off
```

```
      digitalWrite(right_turn_signal, LOW);   //turn LED off
      delay(500);                             //delay 500 ms
      digitalWrite(left_turn_signal,  LOW);   //turn LED off
      digitalWrite(right_turn_signal, HIGH);  //turn LED on
      delay(500);                             //delay 500 ms
      digitalWrite(left_turn_signal,  LOW);   //turn LED off
      digitalWrite(right_turn_signal, LOW);   //turn LED off
      analogWrite(left_motor, 0);             //turn motor off
      analogWrite(right_motor,0);             //turn motor off
      }
}
//*************************************************************************
```

Testing the Control Algorithm. It is recommended that the algorithm be first tested without the entire robot platform. This may be accomplished by connecting the three IR sensors and LEDS to the appropriate pins on the Arduino UNO R3 as specified in Figure 2.21. In place of the two motors and their interface circuits, two LEDs with the required interface circuitry may be used. The LEDs will illuminate to indicate the motors would be on during different test scenarios. Once this algorithm is fully tested in this fashion, the Arduino UNO R3 may be mounted to the robot platform and connected to the motors. Full-up testing in the maze may commence. Enjoy!

2.13 SUMMARY

In this chapter, we have provided an overview of the Arduino concept of open-source hardware. This was followed by a description of the Arduino UNO R3 processor board powered by the ATmega328. An overview of ATmega328 systems followed. This was followed by a description of the Arduino Mega 2560 R3 processor board powered by the ATmega2560 and its systems. We also reviewed the layout and features of the LilyPad Arduino. We then concluded with brief guidelines on how to download and run the Arduino software environment.

2.14 REFERENCES

[1] SparkFun Electronics, 6175 Longbow Drive, Suite 200, Boulder, CO 80301, www.sparkfun.com.

[2] Arduino homepage, www.arduino.cc.

[3] *Microchip 8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash, ATmega48PA, 88PA, 168PA, 328P* data sheet: 8171D-AVR-05/11, Microchip Corporation, 2325 Orchard Parkway, San Jose, CA 95131.

[4] *Microchip 8-bit AVR Microcontroller with 64/128/256K Bytes In-System Programmable Flash, ATmega640/V, ATmega1280/V, 2560/V* data sheet: 2549P-AVR-10/2012, Microchip Corporation, 2325 Orchard Parkway, San Jose, CA 95131.

2.15 CHAPTER PROBLEMS

2.1. Describe in your own words the Arduino open-source concept.

2.2. Sketch a block diagram of the ATmega328 or the ATmega2560 and its associated systems. Describe the function of each system.

2.3. Describe the different types of memory components within the ATmega328 or the ATmega2560. Describe applications for each memory type.

2.4. Describe the three different register types associated with each port.

2.5. How may the features of the Arduino UNO R3 be extended? The Mega 2560?

2.6. Prepare a table of features for different Arduino products.

2.7. Discuss different options for the ATmega328 or the ATmega2560 time base. What are the advantages and disadvantages of each type?

2.8. Summarize the differences between the Arduino UNO R3 and Mega 2560. How would you choose between the two in a given application?

2.9. Design and fabricate your own Arduino hardware studio.

CHAPTER 3

Arduino Power and Interfacing

Objectives: After reading this chapter, the reader should be able to:

- specify a power supply system for an Arduino-based system;
- describe the voltage and current input/output parameters for the Arduino UNO R3, the Arduino Mega 2560, and the Microchip AVR HC CMOS type microcontroller;
- apply the voltage and current input/output parameters toward properly interfacing input and output devices to an Arduino processing board;
- interface a wide variety of input and output devices to an Arduino processing board;
- discuss the requirement for an optical-based interface;
- describe how to control the speed and direction of a DC motor; and
- describe how to control several types of AC loads.

3.1 OVERVIEW

We begin this chapter with exploring the power source requirements for an Arduino-based board. We examine how to probably provide power from a variety of DC and AC power sources. The remainder of the chapter provides information on how to interface input, output, high-power DC, high-power AC, and a variety of other devices to the Arduino processor.

Some of the information for this chapter is originally from Morgan & Claypool Publishers (M&C) book: *Microcontrollers Fundamentals for Engineers and Scientists*. With M&C permission, portions of the chapter have been provided and updated here for your convenience. We have also customized the information to the Arduino UNO R3, the Arduino Mega 2560, and Arduino LilyPad.

3.2 ARDUINO POWER REQUIREMENTS

The Arduino processing boards may be powered from the USB port during project development. However, it is highly recommended that an external power supply be employed any time a peripheral component is connected. This will allow developing projects beyond the limited current capability of the USB port.

For the UNO and the MEGA platforms, Arduino (www.arduino.cc) recommends a power supply from 7–12 VDC with a 2.1-mm center positive plug. A power supply of this type is readily available from a number of electronic parts supply companies. For example, the Jameco #133891 power supply is a 9 VDC model rated at 300 mA and equipped with a 2.1-mm center positive plug. It is available for under US$10. Both the UNO and MEGA have onboard voltage regulators that maintain the incoming power supply voltage to a steady 5 VDC for the onboard processor.

An external power supply may be connected to the Arduino LilyPad via the designated "+" and "–" pads. The power supply should be a regulated 5 VDC source.

3.3 PROJECT REQUIREMENTS

An Arduino board is typically used in a wide variety of projects to control external peripheral devices. These devices may require a variety of DC and/or AC power sources. To provide a proper power source for an Ardunio-based system, the following information must be determined.

- What is the voltage and current requirements of each device in the system?
- Will the system have any current surge requirements (e.g., a motor starting current)?
- Will the system be operated where an AC source is present or will it be a remote system requiring a DC supply?
- How long must the system operate before the batteries can be replaced or recharged?
- Is an alternate power source possible (e.g., solar panel)?

Once these questions are answered, a system power supply may be assembled. In the remainder of this section, we discuss these different power alternatives.

3.3.1 AC OPERATION

If a source of AC power is readily available, an AC-to-DC converter may be used. These range from a single voltage supply (described above) to a multiple DC voltage power supply with different current specifications for each voltage. When selecting a source it is important to insure it is regulated and fused. A regulator maintains the source voltage at the same value even under different current loads. A fuse provides protection against a surge current the power supply cannot handle. When the current requirements for each voltage are determined, a power supply may be selected. Choose a power supply with at least double the current specification as required by the maximum demands of the project. Jameco Electronics provides a wide variety of power supplies (www.jameco.com).

3.3.2 DC OPERATION

For a remote or portable application a DC battery source of power may be used. To select a battery the following requirements must be known: voltage, current, polarity, capacity, and if rechargeable batteries are appropriate for the project.

- Voltage: The unit for voltage is Volts. The voltage for a battery is specified for when it is new or fully charged (for a rechargeable type) battery. Typical battery voltages for AAA, AA, C, and D cells are 1.5 VDC. The batteries may be placed in series to achieve higher voltages. Plastic battery packs are available for battery series stacking to increase the overall voltage rating of the power pack. Another common battery type is the 9 VDC rectangular battery with the plus and minus terminals on the same end of the battery.

- Current: The unit for current is amperes or amps. The current drain of the battery is determined by the load connected to it. For many Arduino based projects the current drain may be specified in mA.

- Polarity: In most projects, a positive voltage referenced to ground is required. Some circuits, for example operational amplifier based instrumentation circuits, may require both a positive and negative supply for proper operation.

- Capacity: The battery capacity specification is provide in mAH or AH (amp-hours). It provides an estimate of how long a battery will last under a particular current drain. Common battery capacities are: AAA-1,000 mAH, AA-2,250 mAH, C-7,000 mAH, D-15,000 mAH, and 9 VDC-550 mAH. These values are only estimates. The exact battery capacity is determined by battery technology and manufacturer.

- Rechargeable: Rechargeable batteries are available in a wide range of voltages and capacities. Capacity is typically provided within the manufacturer's specification for a battery.

  To properly match a battery to an embedded system, the battery voltage and capacity must be specified. Battery capacity is typically specified as a mAH rating. For example, a typical 9 VDC non-rechargeable alkaline battery has a capacity of 550 mAH. If the embedded system has a maximum operating current of 50 mA, it will operate for approximately eleven hours before battery replacement is required.

  A battery is typically used with a voltage regulator to maintain the voltage at a prescribed level. Voltage regulators are available in a variety of voltage and maximum current specification. Figure 3.1 provides sample circuits to provide a +5 VDC and a ±5 VDC portable battery source. Additional information on battery capacity and characteristics may be found in Barrett and Pack [2].

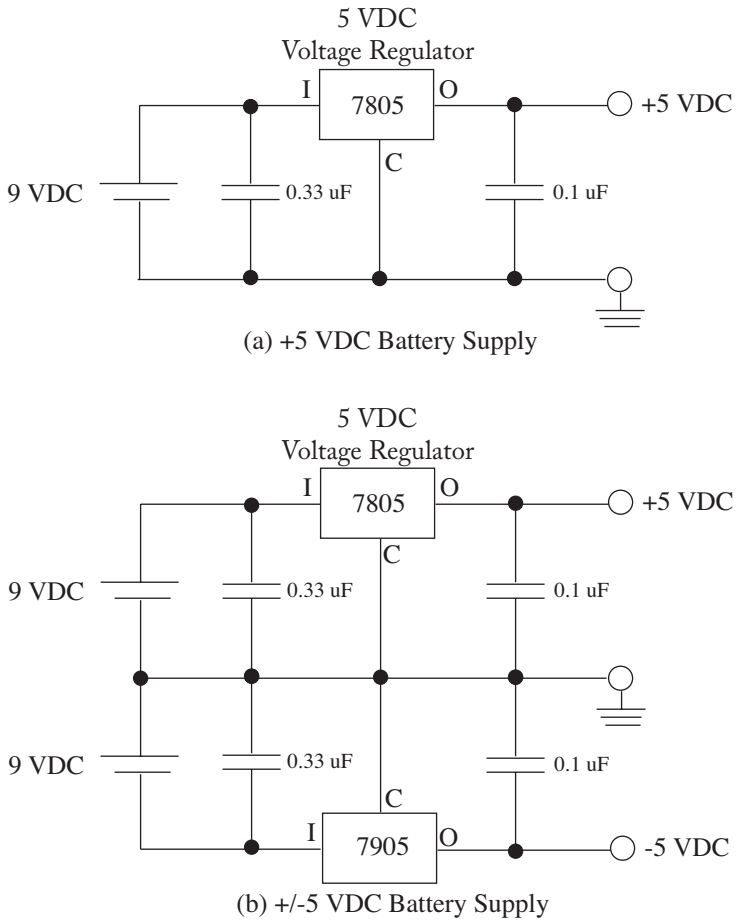


Figure 3.1: Battery supply circuits employing a 9 VDC battery with a 5 VDC regulators.

3.3.3 POWERING THE ARDUINO FROM BATTERIES

As previously mentioned, for the UNO and the MEGA platforms, Arduino recommends a power supply from 7–12 VDC with a 2.1-mm center positive plug (`www.arduino.cc`). For low-power applications a single 9 VDC battery and clip may be used, as shown in Figure 3.2. For higher-power applications, a AA battery pack may be used. It is important to note the UNO R3 and Mega R3 Arduino boards have an onboard 5 VDC regulator.

3.3.4 SOLAR POWER

For some remote applications such as a weather or data collection station solar power may be employed. A solar power system consists of a solar panel, a solar power manager, and a rechargeable

(a) 9 VDC Battery Clip (b) 9 VDC Battery Pack with Lid (c) 9 VDC Battery Pack

Figure 3.2: Arduino 9 VDC battery power.

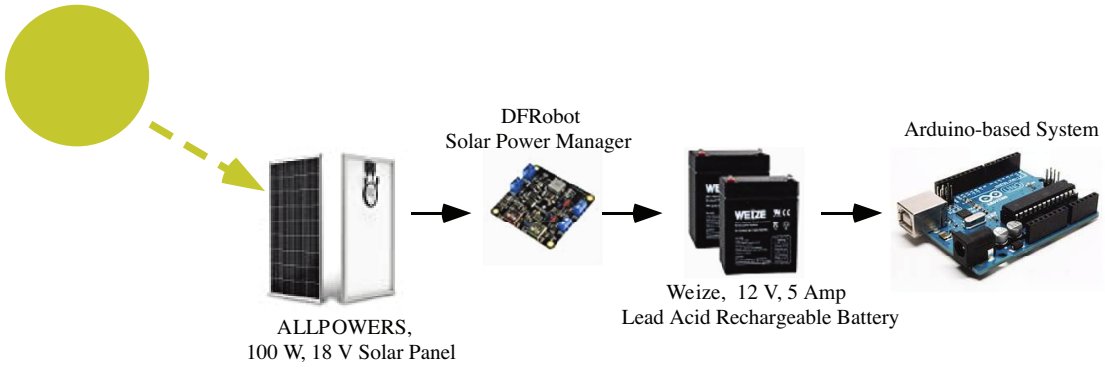


Figure 3.3: Solar power system. Images courtesy of AllPowers, DFRobot, Weize, and Arduino.

battery. DFRobot provides a series of solar power managers for a wide range of project voltages and capacities from very low power (3.3 VDC at 90 mA) to medium power. The DFR0580 Solar Power Manager for a 12 VDC lead-acid battery is highlighted here. With an 18 VDC, 100 W solar panel and a 12 VDC lead-acid battery; the DFR0580 can provide regulated output voltages of 5 VDC at 5 amps and 12 VDC at 8 amps, as shown in Figure 3.3. We use this solar power system in the next chapter to provide power to a remote weather station.

3.4 ADVANCED: OPERATING PARAMETERS

We now introduce the extremely important concepts of the operating envelope for the microcontroller. We begin by reviewing the voltage and current electrical parameters for the HC CMOS based Microchip AVR line of microcontrollers. These parameters define the safe operating envelope of the microcontroller. We then show how to apply this information to properly interface input and output devices to the Arduino UNO R3, the Arduino Mega 2560, and the Arduino

LilyPad. We then discuss the special considerations for controlling a high-power DC or AC load such as a motor and introduce the concept of an optical interface.

The importance of this interfacing information cannot be over emphasized. Any time an input or an output device is connected to a microcontroller, the interface between the device and the microcontroller must be carefully analyzed and designed. This will ensure the microcontroller will continue to operate within specified parameters. Should the microcontroller be operated outside its operational envelope, erratic, unpredictable, and an unreliable system may result.

3.4.1 ADVANCED: HC CMOS PARAMETERS

Most microcontrollers are members of the "HC," or high-speed CMOS family of integrated circuits (chips). As long as all components in a system are also of the "HC" family, as is the case for the Arduino UNO R3, the Arduino Mega 2560, and the LilyPad; electrical interface issues are minimal. If the microcontroller is connected to some component not in the "HC" family, electrical interface analysis must be completed. Manufacturers readily provide the electrical characteristic data necessary to complete this analysis in their support documentation.

To perform the interface analysis, there are eight different electrical specifications required for electrical interface analysis. The electrical parameters are:

- V_{OH}: the lowest guaranteed output voltage for a logic high,
- V_{OL}: the highest guaranteed output voltage for a logic low,
- I_{OH}: the output current for a V_{OH} logic high,
- I_{OL}: the output current for a V_{OL} logic low,
- V_{IH}: the lowest input voltage guaranteed to be recognized as a logic high,
- V_{IL}: the highest input voltage guaranteed to be recognized as a logic low,
- I_{IH}: the input current for a V_{IH} logic high, and
- I_{IL}: the input current for a V_{IL} logic low.

These electrical characteristics are required for both the microcontroller and the external components. Typical values for a microcontroller in the HC CMOS family assuming V_{DD} = 5.0 volts and V_{SS} = 0 volts are provided below. The first letter indicates the parameter (voltage (V) or current (I)). The second letter indicates an output (O) or an input (I) parameter. The final letter indicates whether the parameter is specified for a logic high (H) or low (L) levels. The minus sign on several of the currents indicates a current flow out of the device: A positive current indicates current flow into the device.

- V_{OH} = 4.2 volts,

- V_{OL} = 0.4 volts,
- I_{OH} = -0.8 milliamps,
- I_{OL} = 1.6 milliamps,
- V_{IH} = 3.5 volts,
- V_{IL} = 1.0 volt,
- I_{IH} = 10 microamps, and
- I_{IL} = -10 microamps.

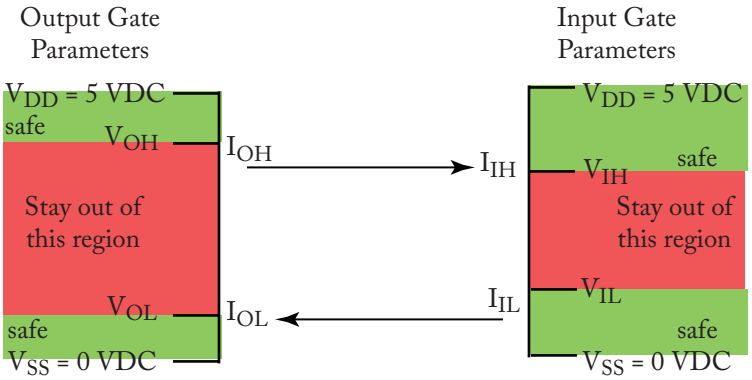
It is important to realize that these are static values taken under very specific operating conditions. If external circuitry is connected such that the microcontroller acts as a current source (current leaving the microcontroller) or current sink (current entering the microcontroller), the voltage parameters listed above will also be affected.

In the current source case, an output voltage V_{OH} is provided at the output pin of the microcontroller when the load connected to this pin draws a current of I_{OH}. If a load draws more current from the output pin than the I_{OH} specification, the value of V_{OH} is reduced. If the load current becomes too high, the value of V_{OH} falls below the value of V_{IH} for the subsequent logic circuit stage and not be recognized as an acceptable logic high signal. When this situation occurs, erratic and unpredictable circuit behavior results. This is an unacceptable condition for a logic circuit.

In the sink case, an output voltage V_{OL} is provided at the output pin of the microcontroller when the load connected to this pin delivers a current of I_{OL} to this logic pin. If a load delivers more current to the output pin of the microcontroller than the I_{OL} specification, the value of V_{OL} increases. If the load current becomes too high, the value of V_{OL} rises above the value of V_{IL} for the subsequent logic circuit stage. When this occurs the input signal will not be recognized as an acceptable logic low signal. As before, when this situation occurs, erratic and unpredictable circuit behavior results. This is an unacceptable condition for a logic circuit.

For convenience this information is illustrated in Figure 3.4. In (a), we provided an illustration of the direction of current flow from the HC device and also a comparison of voltage levels. As a reminder, current flowing out of a device is considered a negative current (source case) while current flowing into the device is considered positive current (sink case). The magnitude of the voltage and current for HC CMOS devices are shown in (b). As more current is sinked or sourced from a microcontroller pin, the voltage will be pulled up or pulled down, respectively, as shown in (c). If input and output devices are improperly interfaced to the microcontroller, these loading conditions may become excessive, and voltages will not be properly interpreted as the correct logic levels.

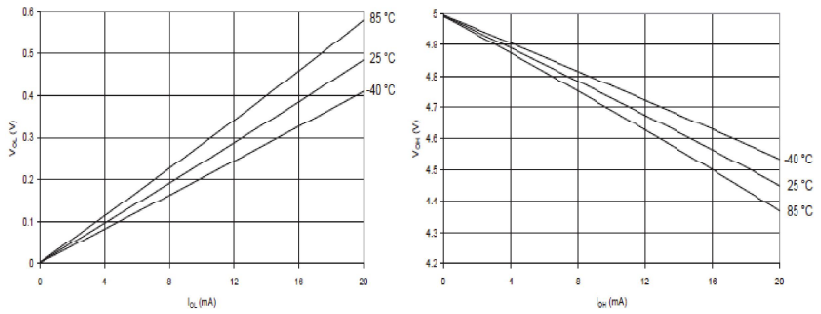
You must also ensure that total current limits for an entire microcontroller port and the overall bulk port specifications are met. For planning purposes the sum of current sourced or



(a) Voltage and Current Electrical Parameters

Output Parameters	Input Parameters
V_{OH} = 4.2 V	V_{IH} = 3.5 V
V_{OL} = 0.4 V	V_{IL} = 1.0 V
I_{OH} = -0.8 mA	I_{IH} = 10 µA
I_{OL} = 1.6 mA	I_{IL} = -10 µA

(b) HC CMOS Voltage and Current Parameters



(c) CMOS Loading Curves. (left) Sink current, (right) Source Current [ATmega328]

Figure 3.4: Electrical voltage and current parameters. Loading curves used with permission of Microchip Technology (www.microchip.com).

sinked from a port should not exceed 100 mA. Furthermore, the sum of currents for all ports should not exceed 200 mA. As before, if these guidelines are not followed, erratic microcontroller behavior may result. This is an unacceptable condition for a logic circuit.

The procedures presented in the following sections, when followed carefully, will ensure the microcontroller will operate within its designed envelope.

3.5 INPUT DEVICES

In this section we describe how to interface a wide variety of input devices to the Arduino microcontroller. In several examples we use components from the Kuman K4 Arduino Starter Kit (`www.kumantech.com`).

3.5.1 SWITCHES

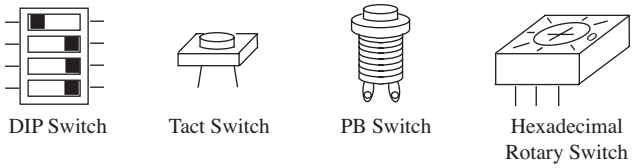
Switches come in a variety of types. As a system designer it is up to you to choose the appropriate switch for a specific application. Switch varieties commonly used in microcontroller applications are illustrated in Figure 3.5a. Here is a brief summary of the different types.

- **Slide Switch.** A slide switch has two different positions: on and off. The switch is manually moved to one position or the other. For microcontroller applications, slide switches are available that fit in the profile of a common integrated circuit size dual inline package (DIP). A bank of four or eight DIP switches in a single package is commonly available. Slide switches are used to select specific parameters at system startup.

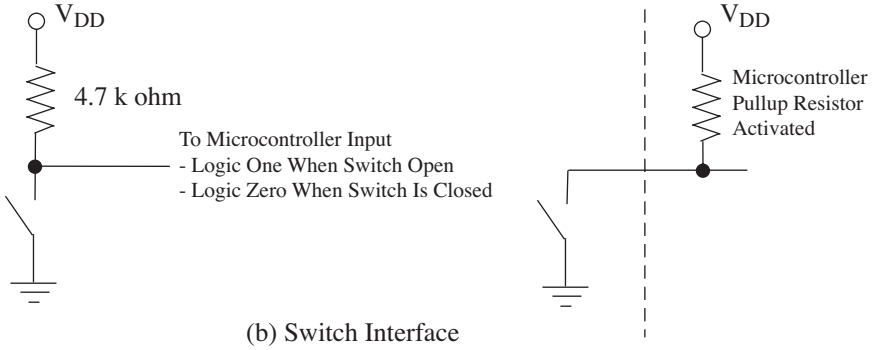
- **Momentary Contact Pushbutton Switch.** A momentary contact pushbutton switch comes in two varieties: normally closed (NC) and normally open (NO). A normally open switch, as its name implies, does not normally provide an electrical connection between its contacts. When the pushbutton portion of the switch is depressed, the connection between the two switch contacts is made. The connection is held as long as the switch is depressed. When the switch is released the connection is opened. The converse is true for a normally closed switch. For microcontroller applications, pushbutton switches are available in a small tact type switch configuration.

- **Push On/Push Off Switches.** These type of switches are also available in a normally open or normally closed configuration. For the normally open configuration, the switch is depressed to make connection between the two switch contacts. The pushbutton must be depressed again to release the connection.

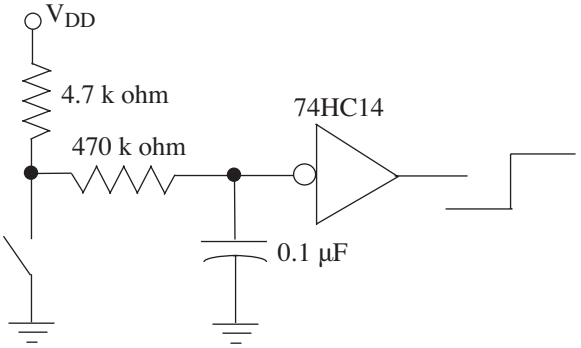
- **Hexadecimal Rotary Switches.** Small profile rotary switches are available for microcontroller applications. These switches commonly have sixteen rotary switch positions. As the switch is rotated to each position, a unique four-bit binary code is provided at the switch contacts.



(a) Switch Varieties



(b) Switch Interface



(c) Switch Interface Equipped with Debouncing Circuitry

Figure 3.5: Switch interface.

A common switch interface is shown in Figure 3.5b. This interface allows a logic one or zero to be properly introduced to a microcontroller input port pin. The basic interface consists of the switch in series with a current limiting resistor. The node between the switch and the resistor is provided to the microcontroller input pin. In the configuration shown, the resistor pulls the microcontroller input up to the supply voltage V_{DD}. When the switch is closed, the node is grounded and a logic zero is provided to the microcontroller input pin. To reverse the logic of the switch configuration, the position of the resistor and the switch is simply reversed.

3.5.1.1 Pullup Resistors in Switch Interface Circuitry

Many microcontrollers are equipped with pullup resistors at the input pins. The pullup resistors are asserted with the appropriate register setting. The pullup resistor replaces the external resistor in the switch configuration, as shown in Figure 3.5b right. The Arduino IDE provides support for pullup resistor activation using the pinmode function. To activate the pullup input resistor, the argument "INPUT_PULLUP" is used. An example is provided in the Arduino IDE under Examples – Digital – DigitalInputPullup.

3.5.1.2 Switch Debouncing

Mechanical switches do not make a clean transition from one position (on) to another (off). When a switch is moved from one position to another, it makes and breaks contact multiple times. This activity may go on for tens of milliseconds. A microcontroller is relatively fast as compared to the action of the switch. Therefore, the microcontroller is able to recognize each switch bounce as a separate and erroneous transition.

To correct the switch bounce phenomena additional external hardware components may be used or software techniques may be employed. A hardware debounce circuit is illustrated in Figure 3.5c. The node between the switch and the limiting resistor of the basic switch circuit is fed to a low-pass filter (LPF) formed by the 470-k ohm resistor and the capacitor. The LPF prevents abrupt changes (bounces) in the input signal from the microcontroller. The LPF is followed by a 74HC14 Schmitt Trigger, which is simply an inverter equipped with hysteresis. This further limits the switch bouncing.

Switches may also be debounced using software techniques. This is accomplished by inserting a 30–50-ms lockout delay in the function responding to port pin changes. The delay prevents the microcontroller from responding to the multiple switch transitions related to bouncing.

You must carefully analyze a given design to determine if hardware or software switch debouncing techniques will be used. It is important to remember that all switches exhibit bounce phenomena and, therefore, must be debounced. An example is provided in the Arduino IDE under Examples – Digital – Debounce.

3.5.2 KEYPADS

A keypad is an extension of the simple switch configuration. A typical keypad configuration and interface are shown in Figure 3.6. As you can see, the keypad contains multiple switches in a two-dimensional array configuration. The switches in the array share common row and column connections. The common column connections are pulled up to Vcc by external 10-k resistors or by pullup resistors within the Arduino. In the example, pullup internal pullup resistors are asserted.

To determine if a switch has been depressed, a single row of keypad switches is first asserted by the microcontroller, followed by a reading of the host keypad column inputs. If a switch has been depressed, the keypad pin corresponding to the column the switch is in will also be

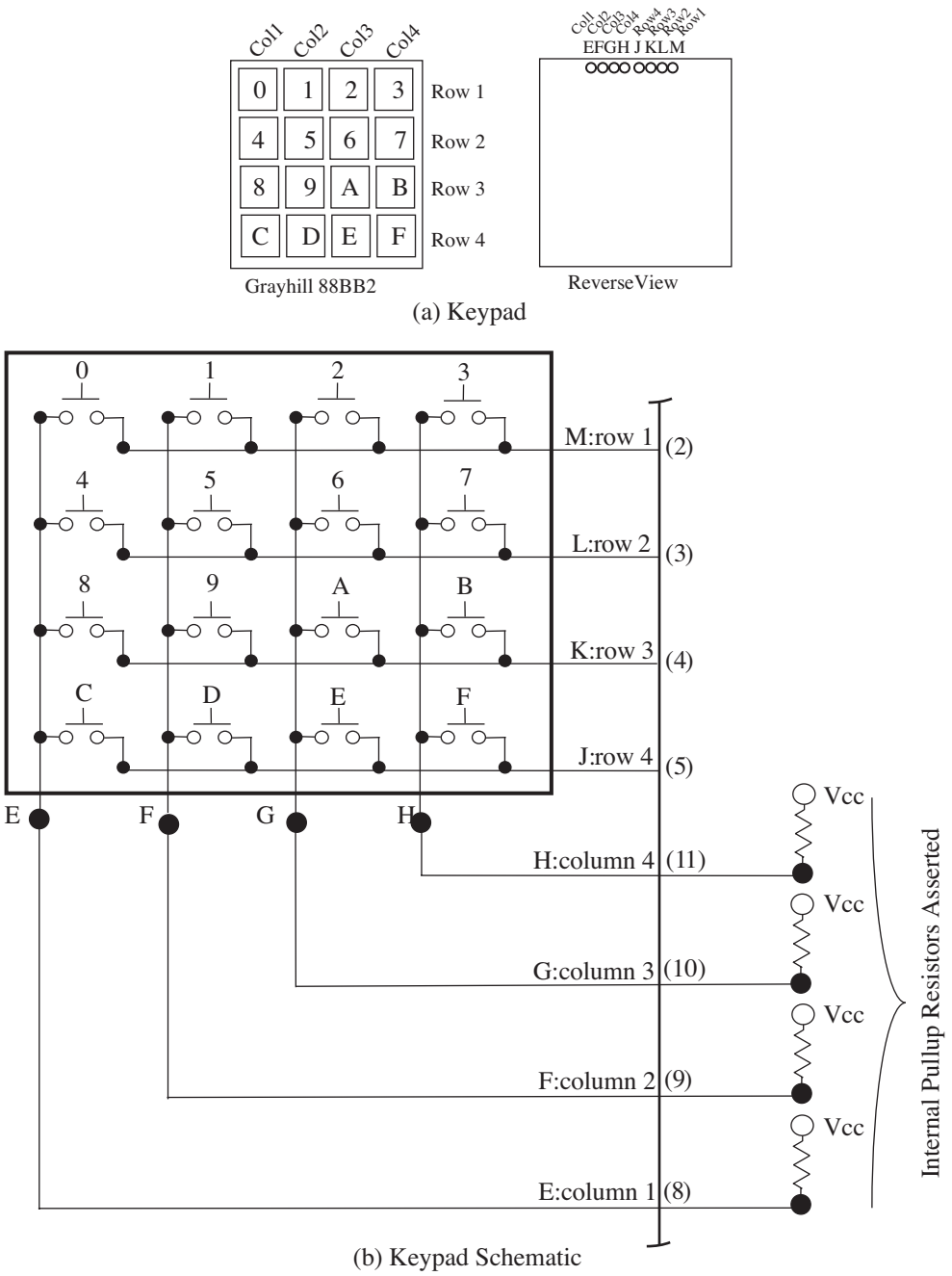


Figure 3.6: Keypad interface.

asserted. The combination of a row and a column assertion can be decoded to determine which key has been pressed. The keypad rows are sequentially asserted. Since the keypad is a collection of switches, debounce techniques must also be employed. In the example code provided, a 200-ms delay is provided to mitigate switch bounce. In the keypad shown, the rows are sequentially asserted active low (0).

The keypad is typically used to capture user requests to a microcontroller. A standard keypad with alphanumeric characters may be used to provide alphanumeric values to the microcontroller such as providing your personal identification number (PIN) for a financial transaction. However, some keypads are equipped with removable switch covers such that any activity can be associated with a key press.

Example: Keypad. In this example a Grayhill 88BB2 4-by-4 matrix keypad is interfaced to the Arduino UNO R3. The example shows how a specific switch depression can be associated with different activities by using a "switch" statement.

```
//*********************************************************************
//keypad_4X4
//Specified pins are for the Arduino UNO R3
//This code is in the public domain.
//*********************************************************************

#define row1  2
#define row2  3
#define row3  4
#define row4  5

#define col1  8
#define col2  9
#define col3  10
#define col4  11

unsigned char  key_depressed = '*';

void setup()
  {
  //start serial connection to monitor
  Serial.begin(9600);

  //configure row pins as ouput
  pinMode(row1, OUTPUT);
  pinMode(row2, OUTPUT);
```

```
  pinMode(row3, OUTPUT);
  pinMode(row4, OUTPUT);

  //configure column pins as input and assert pullup resistors
  pinMode(col1, INPUT_PULLUP);
  pinMode(col2, INPUT_PULLUP);
  pinMode(col3, INPUT_PULLUP);
  pinMode(col4, INPUT_PULLUP);
  }

void loop()
   {
   //Assert row1, deassert row 2,3,4
   digitalWrite(row1, LOW);   digitalWrite(row2, HIGH);
   digitalWrite(row3, HIGH);  digitalWrite(row4, HIGH);

   //Read columns
   if (digitalRead(col1) == LOW)
     key_depressed = '0';
   else if (digitalRead(col2) == LOW)
     key_depressed = '1';
   else if (digitalRead(col3) == LOW)
     key_depressed = '2';
   else if (digitalRead(col4) == LOW)
     key_depressed = '3';
   else
     key_depressed = '*';


   if (key_depressed == '*')
     {
     //Assert row2, deassert row 1,3,4
     digitalWrite(row1, HIGH);   digitalWrite(row2, LOW);
     digitalWrite(row3, HIGH);  digitalWrite(row4, HIGH);

     //Read columns
     if (digitalRead(col1) == LOW)
       key_depressed = '4';
     else if (digitalRead(col2) == LOW)
```

```
    key_depressed = '5';
  else if (digitalRead(col3) == LOW)
    key_depressed = '6';
  else if (digitalRead(col4) == LOW)
    key_depressed = '7';
  else
    key_depressed = '*';
  }

 if (key_depressed == '*')
  {
  //Assert row3, deassert row 1,2,4
  digitalWrite(row1, HIGH);   digitalWrite(row2, HIGH);
  digitalWrite(row3, LOW);  digitalWrite(row4, HIGH);

  //Read columns
  if (digitalRead(col1) == LOW)
    key_depressed = '8';
  else if (digitalRead(col2) == LOW)
    key_depressed = '9';
  else if (digitalRead(col3) == LOW)
    key_depressed = 'A';
  else if (digitalRead(col4) == LOW)
    key_depressed = 'B';
  else
    key_depressed = '*';
  }

if (key_depressed == '*')
  {
  //Assert row4, deassert row 1,2,3
  digitalWrite(row1, HIGH);   digitalWrite(row2, HIGH);
  digitalWrite(row3, HIGH);  digitalWrite(row4, LOW);

  //Read columns
  if (digitalRead(col1) == LOW)
    key_depressed = 'C';
  else if (digitalRead(col2) == LOW)
    key_depressed = 'D';
```

```
      else if (digitalRead(col3) == LOW)
        key_depressed = 'E';
      else if (digitalRead(col4) == LOW)
        key_depressed = 'F';
      else
        key_depressed = '*';
      }

   if(key_depressed != '*')
      {
      Serial.write(key_depressed);
      Serial.write(' ');

      switch(key_depressed)
        {
        case '0': Serial.println("Do case 0") break;
        case '1': Serial.println("Do case 1"); break;
        case '2': Serial.println("Do case 2"); break;
        case '3': Serial.println("Do case 3"); break;
        case '4': Serial.println("Do case 4"); break;
        case '5': Serial.println("Do case 5"); break;
        case '6': Serial.println("Do case 6"); break;
        case '7': Serial.println("Do case 7"); break;
        case '8': Serial.println("Do case 8"); break;
        case '9': Serial.println("Do case 9"); break;
        case 'A': Serial.println("Do case A"); break;
        case 'B': Serial.println("Do case B"); break;
        case 'C': Serial.println("Do case C"); break;
        case 'D': Serial.println("Do case D"); break;
        case 'E': Serial.println("Do case E"); break;
        case 'F': Serial.println("Do case F"); break;
        }
      }
   //limit switch bounce
   delay(200);
   }
//*********************************************************************
```

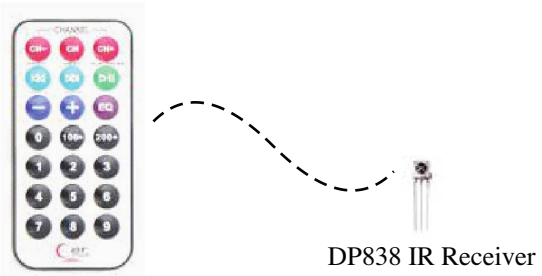


Figure 3.7: IR remote control with receiver (www.kumantech.com).

3.5.3 REMOTE CONTROL

For some Arduino-based projects a remote infrared (IR) remote control might be useful. The Kuman K4 Arduino Starter Kit (www.kumantech.com) contains an IR remote control and a DP838 IR receiver module as shown in Figure 3.7. When a remote control key is depressed a unique code is sent from the remote to the receiver. The received signal is decoded and linked to the specific key that was depressed. Example code is provided in the K4 kit documentation.

3.5.4 SENSORS

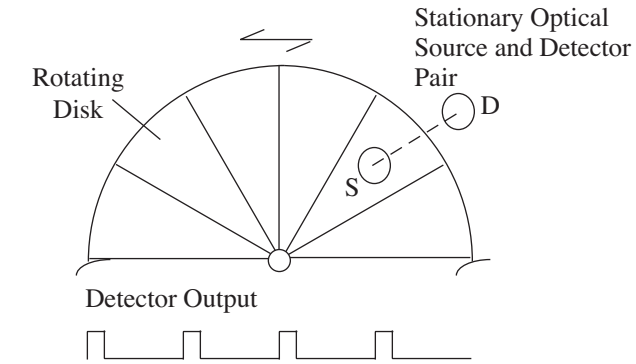
A microcontroller is typically used in applications where data is collected by input sensors, the data is assimilated and processed by the host algorithm, and a control decision and accompanying signals are provided by the microcontroller to output peripheral devices. The sensors may be digital or analog in nature.

3.5.4.1 Digital Sensors

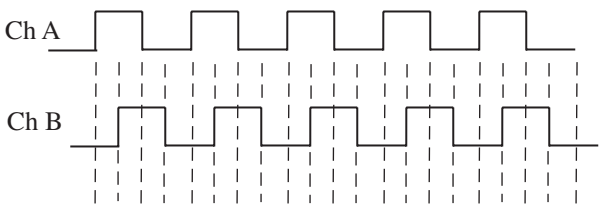
Digital sensors provide a series of digital logic pulses with sensor data encoded. The sensor data may be encoded in any of the parameters associated with the digital pulse train such as duty cycle, frequency, period, or pulse rate. The input portion of the timing system may be configured to measure these parameters.

An example of a digital sensor is the optical encoder. An optical encoder consists of a small plastic transparent disk with opaque lines etched into the disk surface. A stationary optical emitter and detector pair is placed on either side of the disk. As the disk rotates, the opaque lines break the continuity between the optical source and detector. The signal from the optical detector is monitored to determine disk rotation, as shown in Figure 3.8.

Optical encoders are available in a variety of types depending on the information desired. There are two major types of optical encoders: incremental encoders and absolute encoders. An absolute encoder is used when it is required to retain position information when power is lost. For example, if you were using an optical encoder in a security gate control system, an



(a) Incremental Tachometer Encoder



(b) Incremental Quadrature Encoder

Figure 3.8: Optical encoder.

absolute encoder would be used to monitor the gate position. An incremental encoder is used in applications where a velocity or a velocity and direction information is required.

The incremental encoder types may be further subdivided into tachometers and quadrature encoders. An incremental tachometer encoder consists of a single track of etched opaque lines, as shown in Figure 3.8a. It is used when the velocity of a rotating device is required. To calculate velocity, the number of detector pulses are counted in a fixed amount of time. Since the number of pulses per encoder revolution is known, velocity may be calculated.

The quadrature encoder contains two tracks shifted in relationship to one another by 90°. This allows the calculation of both velocity and direction. To determine direction, one would monitor the phase relationship between Channel A and Channel B, as shown in Figure 3.8b. The absolute encoder is equipped with multiple data tracks to determine the precise location of the encoder disk (Sick/Stegmann [4]).

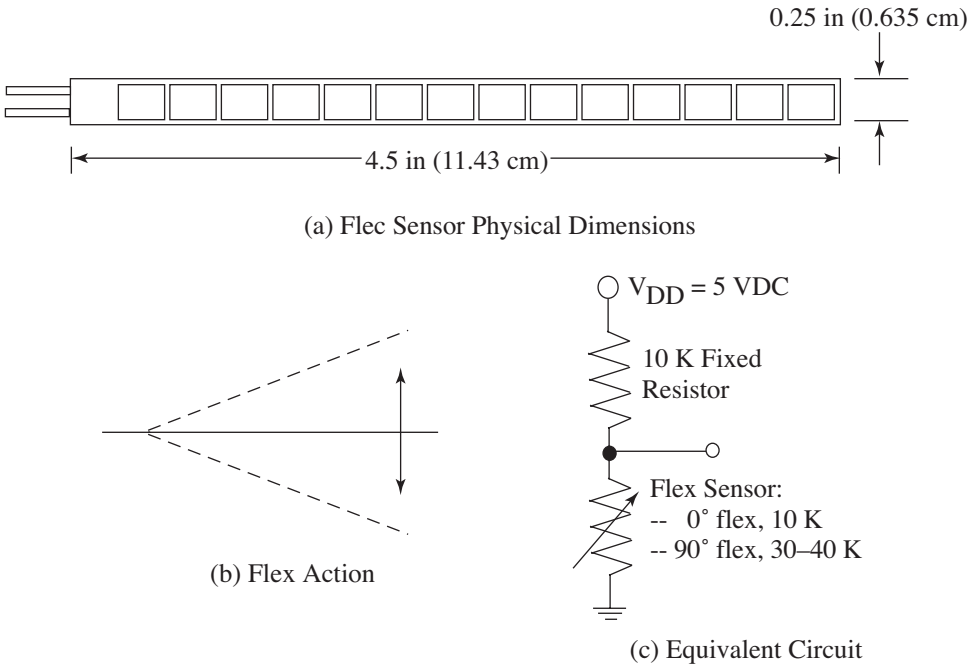


Figure 3.9: Flex sensor.

3.5.4.2 Analog Sensors

Analog sensors provide a DC voltage that is proportional to the physical parameter being measured. As discussed in the analog to digital conversion chapter, the analog signal may be first preprocessed by external analog hardware such that it falls within the voltage references of the conversion subsystem. The analog voltage is then converted to a corresponding binary representation.

Example: Flex Sensor. An example of an analog sensor is the flex sensor shown in Figure 3.9a. The flex sensor provides a change in resistance for a change in sensor flexure. At 0° flex, the sensor provides 10-k ohms of resistance. For 90° flex, the sensor provides 30–40-k ohms of resistance. Since the processor cannot measure resistance directly, the change in flex sensor resistance must be converted to a change in a DC voltage. This is accomplished using the voltage divider network shown in Figure 3.9c. For increased flex, the DC voltage will increase. The voltage can be measured using the analog-to-digital converter subsystem.

The flex sensor may be used in applications such as virtual reality data gloves, robotic sensors, biometric sensors, and in science and engineering experiments (Images Company [5]). The author used the circuit provided in Figure 3.9 to help a colleague in zoology monitor the movement of a newt salamander during a scientific experiment.

Example: Ultrasonic Sensor. The ultrasonic sensor pictured in Figure 3.10 is an example of an analog-based sensor. The sensor is based on the concept of ultrasound or sound waves that are at a frequency above the human range of hearing (20 Hz–20 kHz). The ultrasonic sensor pictured in Figure 3.10c emits a sound wave at 42 kHz. The sound wave reflects from a solid surface and returns back to the sensor. The amount of time for the sound wave to transit from the surface and back to the sensor may be used to determine the range from the sensor to the wall. Pictured in Figures 3.10c,d is an ultrasonic sensor manufactured by Maxbotix (LV-EZ3). The sensor provides an output that is linearly related to range in three different formats: (a) a serial RS-232 compatible output at 9600 bits per second, (b) a PWM output at a 147 us/inch duty cycle, and (c) an analog output at a resolution of 10 mV/inch. The sensor is powered from a 2.5–5.5 VDC source (`www.sparkfun.com`).

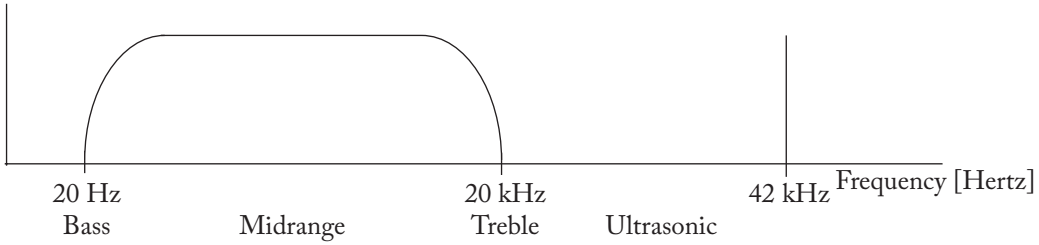
Example: LM34 and LM35 Temperature Sensor Example. Temperature may be sensed using an LM34 (Fahrenheit) or LM35 (Centigrade) temperature transducer. The LM34 provides an output voltage that is linearly related to temperature. For example, the LM34D operates from 32°–212°F providing +10 mV/°F resolution with a typical accuracy of ±0.5°F (National Semiconductor [10]). The output from the sensor is typically connected to the ADC input of the microcontroller. Example code for the LM35 sensor is provided in the Kuman K4 kit documentation (`www.kumantech.com`).

3.5.5 JOYSTICK

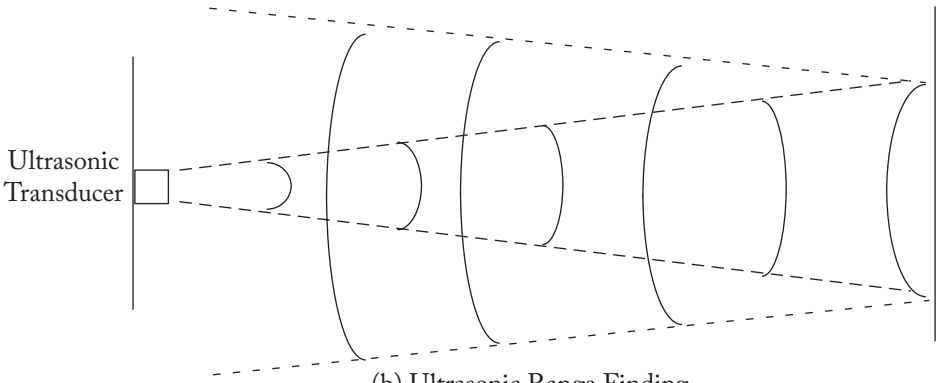
The thumb joystick is used to select a desired direction in an X–Y plane as shown in Figure 3.11. The thumb joystick contains two built-in potentiometers (horizontal and vertical). A reference voltage of 5 VDC is applied to the VCC input of the joystick. As the joystick is moved, the horizontal (HORZ) and vertical (VERT) analog output voltages will change to indicate the joystick position. The joystick is also equipped with a digital select (SEL) button. We use the joystick to control an underwater ROV in Chapter 4.

3.5.6 LEVEL SENSOR

Milone Technologies manufacture a line of continuous fluid level sensors. The sensor resembles a ruler and provides a near linear response, as shown in Figure 3.12. The sensor reports a change in resistance to indicate the distance from sensor top to the fluid surface. A wide resistance change occurs from 700 ohms at a one inch fluid level to 50 ohms at a 12.5-inch fluid level (`www.milonetech.com`). To covert the resistance change to a voltage change measurable by the Arduino, a voltage divider circuit as shown in Figure 3.12 may be used. With a supply voltage (V_{DD}) of 5 VDC, a V_{TAPE} voltage of 1.3 VDC results for a 1-inch fluid level, whereas a fluid of 12.5 inches provides a V_{TAPE} voltage level of 0.12 VDC.



(a) Sound Spectrum



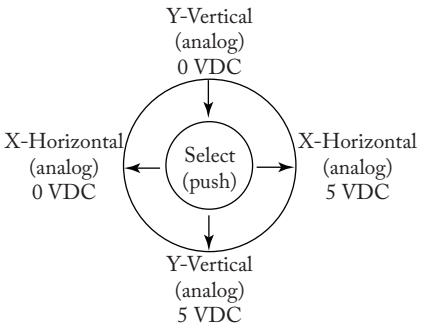
(b) Ultrasonic Range Finding

(c) Ultrasonic Range Finder Maxbotix LV-EZ3 (SparkFun SEN-08501)

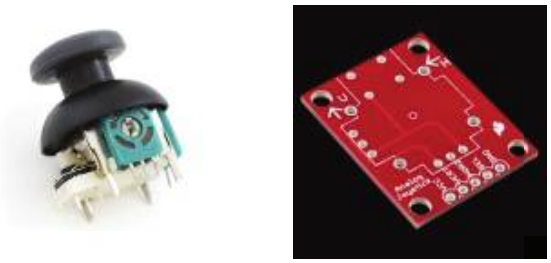
O1: leave open
O2: PW
O3: analog output
O4: RX
O5: TX
O6: V + (3.3–5.0 V)
O7: gnd

(d) Pinout

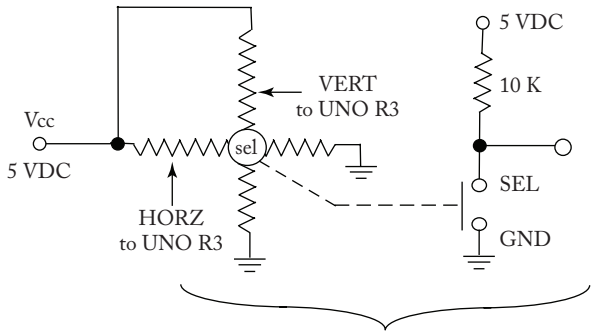
Figure 3.10: Ultrasonic sensor. Sensor image used courtesy of SparkFun, Electronics (CC BY-NC-SA) (www.sparkfun.com).



(a) Joystick Operation



(b) Sparkfun Joystick (COM-09032) and Breakout Board (BOB-09110)



(c) Thumb Joystick Circuit

Figure 3.11: Thumb joystick. Joystick image used courtesy of SparkFun, Electronics (CC BY-NC-SA; www.sparkfun.com).

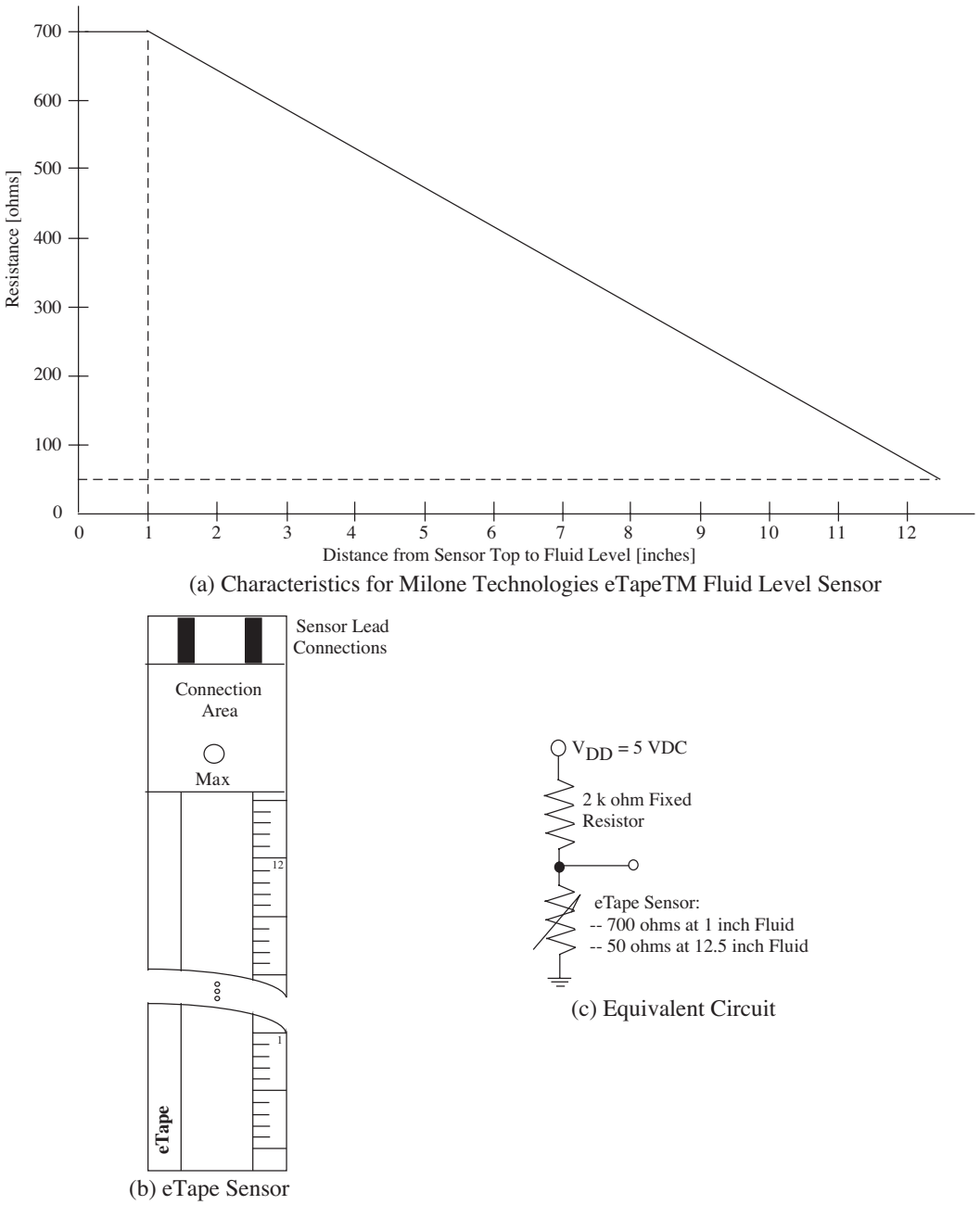


Figure 3.12: Milone Technologies fluid level sensor (`www.milonetech.com`).

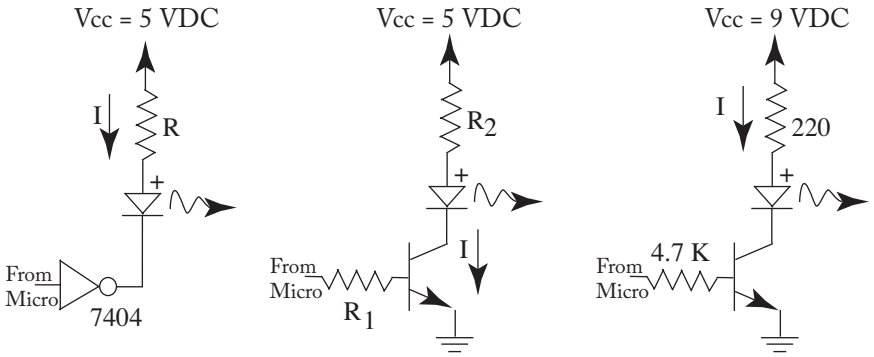


Figure 3.13: LED display devices.

3.6 OUTPUT DEVICES

As previously mentioned, an external device should not be connected to a microcontroller without first performing careful interface analysis to ensure the voltage, current, and timing requirements of the microcontroller and the external device. In this section, we describe interface considerations for a wide variety of external devices. We begin with the interface for a single LED.

3.6.1 LIGHT-EMITTING DIODES (LEDS)

An LED is typically used as a logic indicator to inform the presence of a logic one or a logic zero at a specific pin of a microcontroller. An LED has two leads: the anode or positive lead and the cathode or negative lead. To properly bias an LED, the anode lead must be biased at a level approximately 1.7–2.2 volts higher than the cathode lead. This specification is known as the forward voltage (V_f) of the LED. The LED current must also be limited to a safe level known as the forward current (I_f). The diode voltage and current specifications are usually provided by the manufacturer.

An example of an LED biasing circuit is provided in Figure 3.13. A logic one is provided by the microcontroller to the input of the inverter. The inverter provides a logic zero at its output which provides a virtual ground at the cathode of the LED. Therefore, the proper voltage biasing for the LED is provided. The resistor (R) limits the current through the LED. A proper resistor value can be calculated using $R = (V_{DD} - V_{DIODE})/I_{DIODE}$. It is important to note that a 7404 inverter must be used due to its capability to safely sink 16 mA of current. Alternately, an NPN transistor such as a 2N2222 (PN2222 or MPQ2222) may be used in place of the inverter as shown in the figure. In Chapter 1, we used large (10 mm) red LEDs in the KNH instrumentation project. These LEDs have V_f of 6–12 VDC and I_f of 20 mA at 1.85 VDC. This requires the interface circuit shown in Figure 3.13c right.

3.6.2 SEVEN-SEGMENT LED DISPLAYS – SMALL

To display numeric data, seven-segment LED displays are available, as shown in Figure 3.14a. Different numerals can be displayed by asserting the proper LED segments. For example, to display the number five, segments a, c, d, f, and g would be illuminated. Seven-segment displays are available in common cathode (CC) and common anode (CA) configurations. As the CC designation implies, all seven individual LED cathodes on the display are tied together. A limiting resistor is required for each segment to limit the current to a safe value for the LED. Conveniently, resistors are available in DIP packages of eight for this type of application.

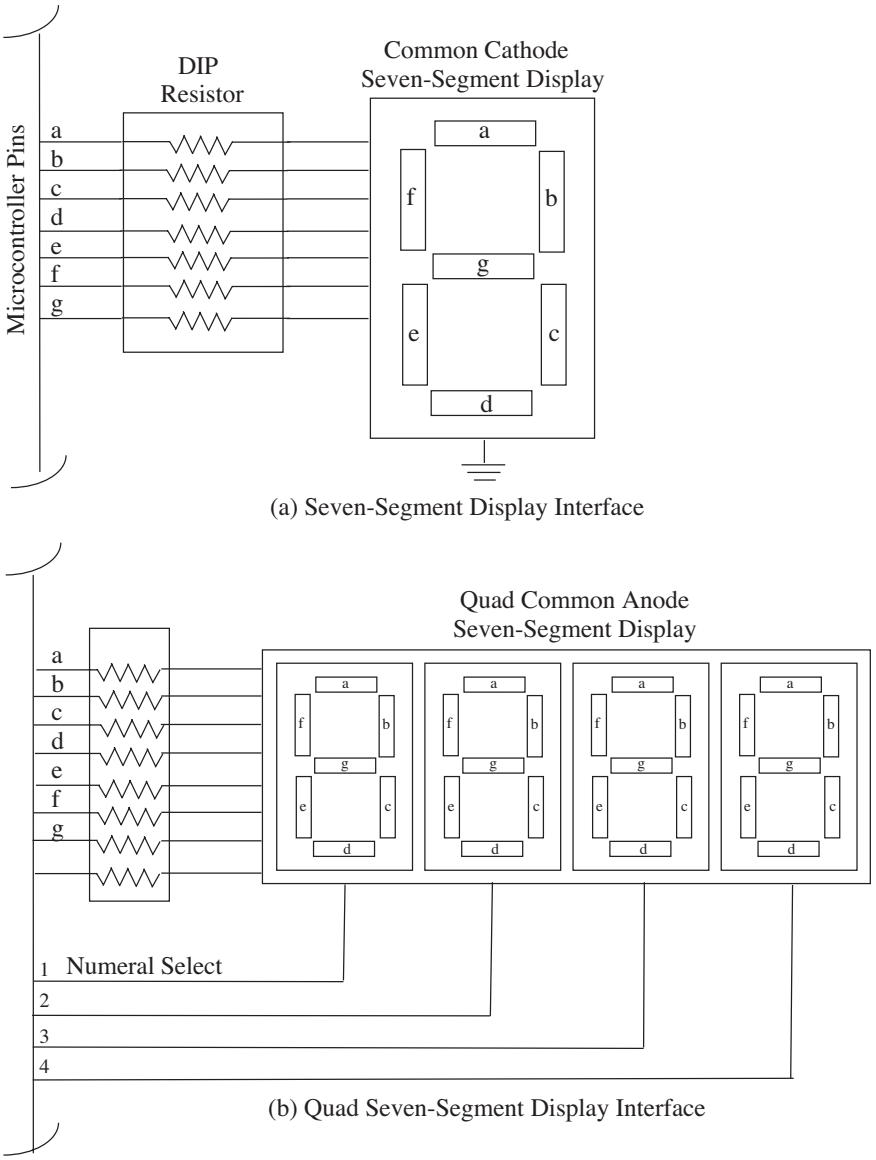
Seven-segment displays are available in multi-character panels. In this case, separate microcontroller pins are not used to provide data to each seven-segment character. Instead, a group of seven pins are used to provide character data. Another group of four pins are used to sequence through each of the characters as shown in Figure 3.14b. As the common anode of each seven-segment numeral is sequentially asserted, the specific character is illuminated. If the microcontroller sequences through the display characters at a rate greater than 30 Hz, the display will have steady illumination. The Kuman K4 kit documentation provides examples for seven-segment displays.

3.6.3 SEVEN-SEGMENT LED DISPLAYS – LARGE

Large seven-segments displays with character heights of 6.5 inches are available from SparkFun Electronics (`www.sparkfun.com`). Multiple display characters may be daisy chained together to form a display panel of desired character length. Only four lines from the Arduino are required to control the display panel (ground, latch, clock, and serial data). Each character is controlled by a Large Digit Driver Board (#WIG-13279) equipped with the Texas Instrument TPIC6C596 IC Program Logic 8-bit Shifter Register. The shift register requires a 5 VDC supply and has a V_{IH} value of 4.25 VDC.

The Arduino's Serial Peripheral Interface (SPI) system is used to send numerical data to Sparkfun's 6.5-inch seven-segment displays (COM-08530). Two of the large digit displays are serially linked together via Sparkfun's Large Digit Driver (WIG-13279). The Large Digit Drivers are soldered to the back of the 6.5-inch seven-segment displays. The hardware configuration is shown in Figure 3.15.

Numerical data is shifted out of the Arduino UNO R3 to the TPIC6C696 shift register within the Large Digit Driver (WIG-13279). In the code example, LED_big_digit2, two displays are sent an incrementing value from 00–99.



(a) Seven-Segment Display Interface

(b) Quad Seven-Segment Display Interface

Figure 3.14: Seven-segment LED display devices.

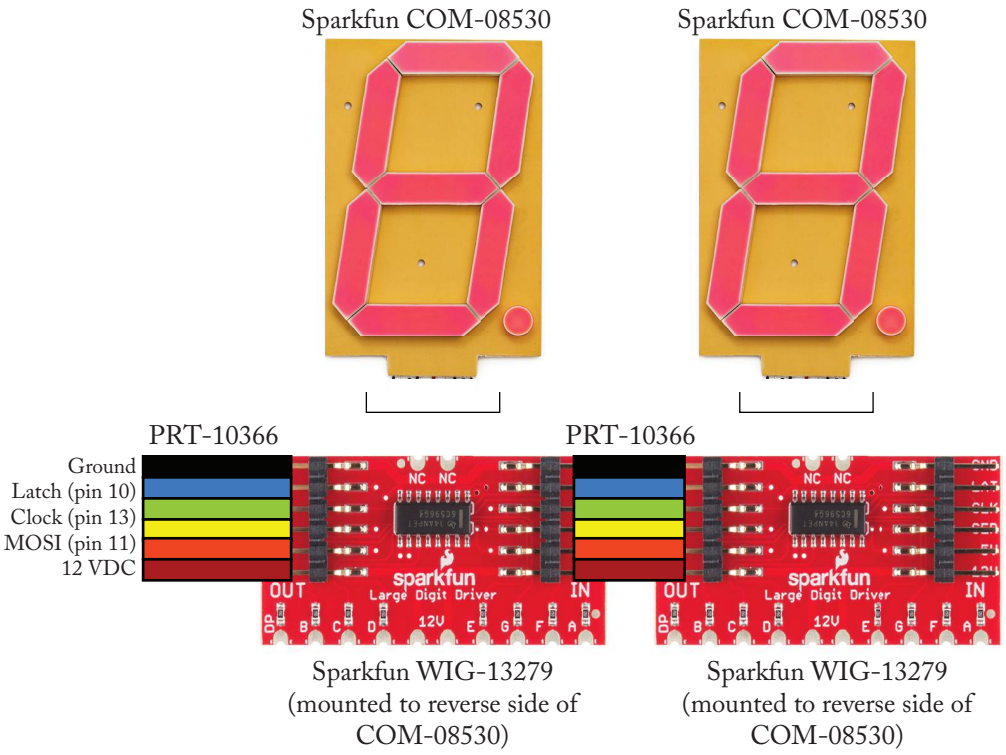


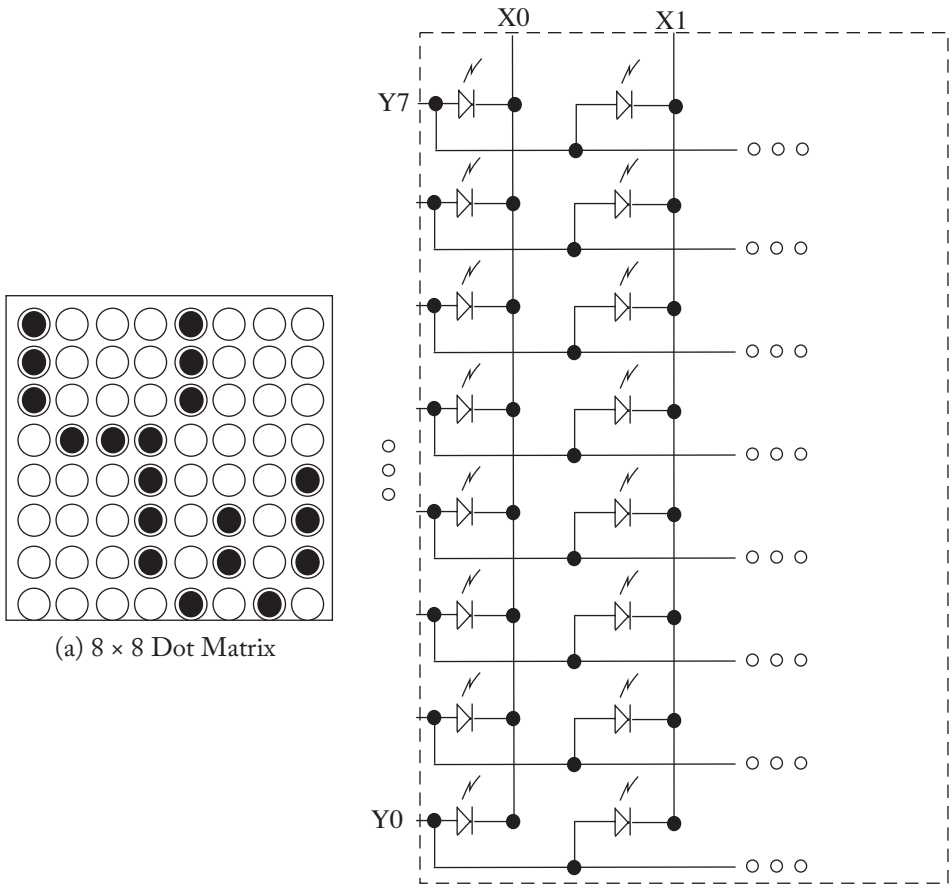
Figure 3.15: Arduino UNO R3 interface to Sparkfun's 6.5-inch seven-segment displays (COM-08530). Illustration used with permission of Sparkfun Electronics (www.sparkfun.com).

```
//*************************************************************************
//LED_big_digit2:  Demonstrates use of the Arduino SPI system to
//illuminate different numbers on Sparkfun's 6.5" 7-segment display
//(COM-08530).  Numerals are sent from the Arduino UNO R3 to Sparkfun's
//Large Digit Driver (WIG-13279).
//
//WIG-13279 pin connections:
// - External 12 VDC supply - red
// - External 5 VDC supply - orange
// - Power supply grounds should be connected to common ground
// - Serial Data Out  - MOSI pin 11 - yellow
// - CLK - SCK pin 13 - green
// - Latch - pin 10 - blue
// - Ground - black
```

```
//Notes:
// - SPI must be configured for least significant bit (LSB) first
// - The numerals 0 to 9 require the following data words as required
//   by the interface between the Spakfun Large Digit Driver (WIG-13279)
//   and the Sparkfun 6.5" 7-segment display (COM-08530).
//
//   Numeral          Data representation of numeral
//     0                          0xDE
//     1                          0x06
//     2                          0xBA
//     3                          0xAE
//     4                          0x66
//     5                          0xEC
//     6                          0xFC
//     7                          0x86
//     8                          0xFE
//     9                          0xE6
//
//This example code is in the public domain.
//*****************************************************************

#include <SPI.h>

//Seven-segment numeral code
#define  seven_seg_zero         0xDE
#define  seven_seg_one          0x06
#define  seven_seg_two          0xBA
#define  seven_seg_three        0xAE
#define  seven_seg_four         0x66
#define  seven_seg_five         0xEC
#define  seven_seg_six          0xFC
#define  seven_seg_seven        0x86
#define  seven_seg_eight        0xFE
#define  seven_seg_nine         0xE6

#define  LATCH    10                //Arduino UNO R3 pin 10

const byte    strip_length = 1;     //number of 7-segment LEDs
unsigned char troubleshooting = 0;  //allows printouts to serial
```

```
unsigned int  numeral, first_digit, second_digit;
unsigned char segment_data_return;

void setup()
  {
  pinMode(LATCH, OUTPUT);
  SPI.begin();                          //SPI support functions
  SPI.setBitOrder(LSBFIRST);            //SPI bit order - LSB first
  SPI.setDataMode(SPI_MODE3);           //SPI mode
  SPI.setClockDivider(SPI_CLOCK_DIV32);//SPI data clock rate
  Serial.begin(9600);                   //serial comm at 9600 bps
  }

void loop()
{
digitalWrite(LATCH, LOW);             //initialize LATCH signal
SPI.transfer(seven_seg_zero);         //reset to zero
assert_latch();

for(numeral = 0; numeral<=99; numeral++)
  {
  if(numeral <= 9)
    {
    segment_data_return = determine_segments(numeral);
    SPI.transfer(segment_data_return);     //transmit data via SPI
    SPI.transfer(seven_seg_zero);
    assert_latch();
    delay(1000);                           //1s delay
    } //end if
  else //numeral >=10 - two digit analysis
    {
    first_digit = numeral
    second_digit = (int)((numeral-first_digit)/10);
    segment_data_return = determine_segments(first_digit);
    SPI.transfer(segment_data_return);     //transmit data via SPI
    segment_data_return = determine_segments(second_digit);
    SPI.transfer(segment_data_return);     //transmit data via SPI
    assert_latch();
    delay(1000);                           //1s delay
```

```
    }//end else
  }  //end for
}  //end void

//*****************************************************************

void assert_latch()
{
digitalWrite(LATCH, HIGH);            //transmit latch pulse
delay(50);
digitalWrite(LATCH, LOW);             //initialize LATCH signal
}

//*****************************************************************

unsigned char determine_segments(unsigned int segment_number)
{

unsigned char segment_data;

switch(segment_number)                 //convert numeral to
      {                                //7-segment code
      case 0: segment_data = seven_seg_zero;  break;
      case 1: segment_data = seven_seg_one;   break;
      case 2: segment_data = seven_seg_two;   break;
      case 3: segment_data = seven_seg_three; break;
      case 4: segment_data = seven_seg_four;  break;
      case 5: segment_data = seven_seg_five;  break;
      case 6: segment_data = seven_seg_six;   break;
      case 7: segment_data = seven_seg_seven; break;
      case 8: segment_data = seven_seg_eight; break;
      case 9: segment_data = seven_seg_nine;  break;
      default: break;
      }

return segment_data;

}
```



(a) 8 × 8 Dot Matrix

(b) 8 × 8 Dot Matrix Details

Figure 3.16: Dot matrix display.

```
//*************************************************************
```

3.6.4 DOT MATRIX DISPLAY

The dot matrix display consists of a large number of LEDs configured in a single package. A typical 8 × 8 LED arrangement is a matrix of eight columns of LEDs with eight LEDs per row, as shown in Figure 3.16. Each LED is individually addressable by its corresponding X and Y lines. If the microcontroller sequences through each column fast enough (greater than 30 Hz), the matrix display appears to be stationary to a human viewer. The dot matrix display may be used to display alphanumeric data as well as graphics data. The Kuman K4 kit contains an 8 × 8 dot matrix display and sample code (`www.kuman.com`).

3.6.5 SERIAL LIQUID CRYSTAL DISPLAY (LCD)

An LCD is an output device to display text information. LCDs come in a wide variety of configurations including multi-character, multi-line format. A 16 × 2 LCD format is common. That is, it has the capability of displaying 2 lines of 16 characters each. Each display character and line has a specific associated address. The characters are sent to the LCD via American Standard Code for Information Interchange (ASCII) format a single character at a time.

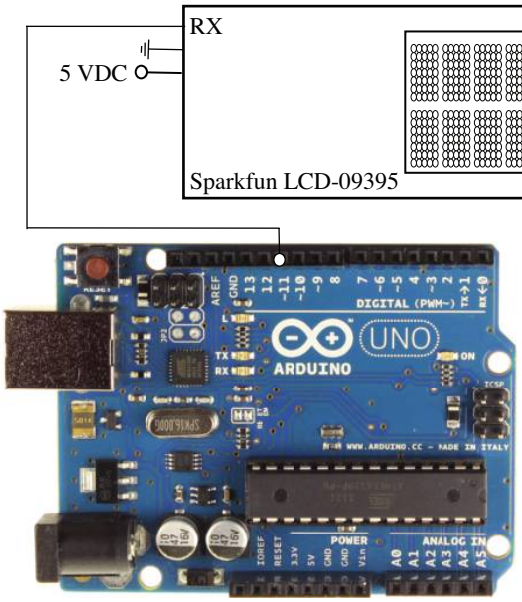
For a parallel configured LCD, an eight-bit data path and two lines are required between the microcontroller and the LCD. Many parallel configured LCDs may also be configured for a four-bit data path thus saving several precious microcontroller pins. A small microcontroller mounted to the back panel of the LCD translates the ASCII data characters and control signals to properly display the characters.

To conserve precious, limited microcontroller input/output pins, a serial configured LCD may be used. A serial LCD reduces the number of required microcontroller pins for interface, from ten down to one, as shown in Figure 3.17. Display data and control information is sent to the LCD via an asynchronous UART serial communication link (8 data bits, 1 stop bit, no parity, 9600 Baud). A serial configured LCD costs slightly more than a similarly configured parallel LCD.

Example: In this example, a Sparkfun LCD-09395, 5.0 VDC, serial, 16 by 2 character, black on white LCD display is connected to the Arduino UNO R3. Communication between the UNO R3 and the LCD is accomplished by a single 9600 bits per second (BAUD) connection.

Rather than use the onboard Universal Asynchronous Receiver Transmitter (UART), the Arduino Software Serial Library is used. The library provides functions to mimic UART activities on a digital pin. Details on the Library are provided at the Arduino website www.arduino.cc.

```
//*****************************************************************
//Example uses the Arduino Software Serial Library with the
//Sparkfun LCD-09395.
// - provides software-based serial port
//*****************************************************************

#include <SoftwareSerial.h>

//Specify Arduino pins for Serial connection:
//  SoftwareSerial LCD(RX_pin, TX_pin);
SoftwareSerial LCD(10, 11);

void setup()
{
```



Line	Character Position (n)
1	0-15
2	64-79
3	16-31
4	80-95

Note: Character position is specifed as 0X80 + n

Command Code	Command
0x01	Clear Display
0x14	Cursor One Space Right
0x10	Cursor One Space Left
0x80 + n	Cursor to Position

Note: Precede command with 0xFE (254_{10})

Figure 3.17: LCD serial display. (UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA); www.arduino.cc.)

```
LCD.begin(9600);                    //Baud rate: 9600 Baud
delay(500);                         //Delay for display
}

void loop()
{
//Cursor to line one, character one
LCD.write(254);                     //Command prefix
LCD.write(128);                     //Command

//clear display
LCD.write("                ");
LCD.write("                ");

//Cursor to line one, character one
LCD.write(254);                   //Command prefix
```

```
LCD.write(128);                    //Command

LCD.write("SerLCD Test");

//Cursor to line two, character one
LCD.write(254);                    //Command prefix
LCD.write(192);                    //Command

LCD.write("LCD-09395");

while(1);                          //pause here
}

//*****************************************************************
```

3.6.6 TEXT-TO-SPEECH MODULE

To give a project a voice, a Text-to-Speech (TTS) module may be used. Parallax manufactures the Emic 2 TTS module. The module was developed in partnership with Grand Idea Studio (www.grandideastudio.com). The module is based on early TTS techniques developed at Digital Equipment Corporation (DEC) in the early 1980s. The Emic 2 provides for a variety of voices, languages, speech rates, and volume, as shown in Figure 3.18. The different voice features are selected using commands sent from the host processor via a 9,600 bits per second asynchronous serial (8 data bits, no parity, one stop bit) bit stream (Emic 2 [13]). The Emic 2 TTS module is available from Adafruit (#924) (www.adafruit.com) or Sparkfun (www.sparkfun.com).

The Arduino UNO R3 is equipped with a serial USART channel used for full duplex (two way) communication (pin 0: RX, pin 1: TX). Alternatively, the Arduino SoftwareSerial Library may be used to emulate a USART channel. In the example, the library is used to provide USART communication on pin 2 (RX) and pin 3 TX). In the example, note the two way communication between the Emic 2 TTS module and the Arduino UNO R3.

```
//*****************************************************************
//Emic 2 Text-to-Speech Module: Basic Demonstration
//
//Author: Joe Grand [www.grandideastudio.com]
//Contact: support@parallax.com
//
//Program Description: This program provides a simple demonstration
//of the Emic 2 Text-to-Speech Module. Please refer to the product
//manual for full details of system  functionality and capabilities.
```

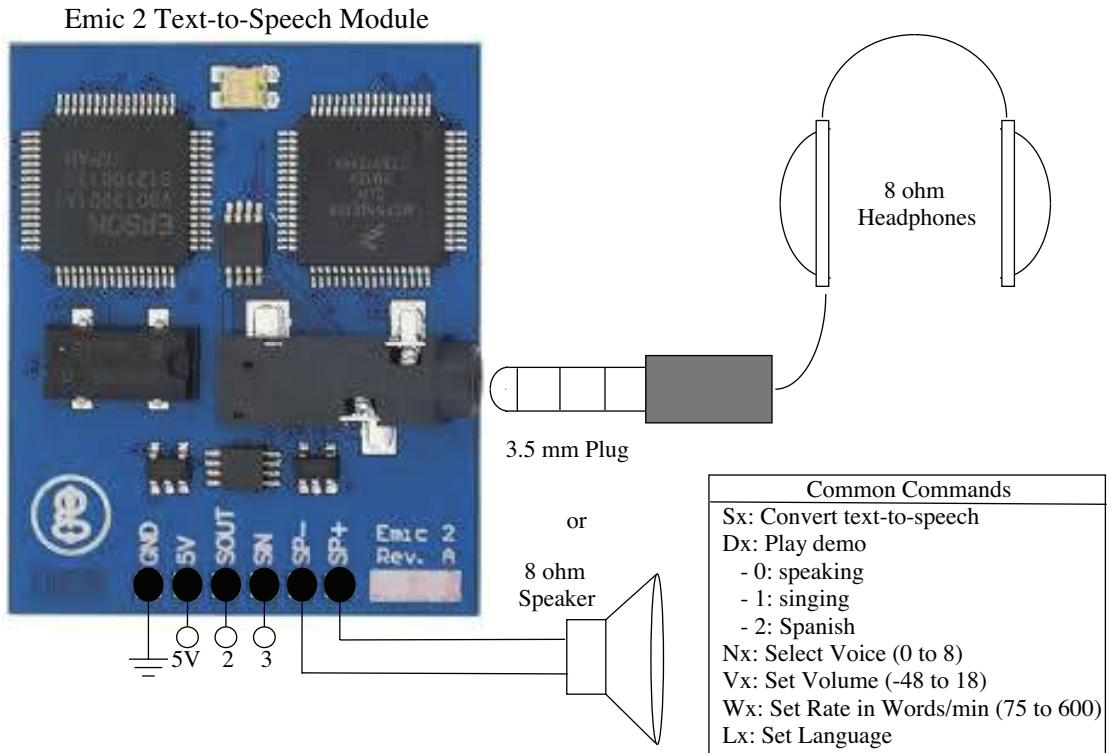


Figure 3.18: Emic 2 text-to-speech (TTS) module. Emic 2 illustration used courtesy of Adafruit (www.adafruit.com).

```
//
//Revisions: 1.0 (February 13, 2012): Initial release
//*****************************************************************

//Include SoftwareSerial library for two-way communication
//with Emic 2 module
#include <SoftwareSerial.h>

#define rxPin 2              //Serial input (connects to Emic 2 SOUT)
#define txPin 3              //Serial output (connects to Emic 2 SIN)
#define ledPin 13            //Arduino on-board LED
```

```
                              //set up a new serial port
SoftwareSerial emicSerial = SoftwareSerial(rxPin, txPin);


void setup()
{
pinMode(ledPin, OUTPUT);    //define pin modes
pinMode(rxPin, INPUT);
pinMode(txPin, OUTPUT);


emicSerial.begin(9600);     //set SoftwareSerial data rate
digitalWrite(ledPin, LOW);   //turn LED off

//Note: When the Emic 2 TTS module powers on, it takes
//about 3 seconds for it to successfully initialize.
//The Emic 2 then sends a ":" character to indicate
//it's ready to accept commands. If the Emic 2 is already
//initialized, a CR will also cause it to send a ":"

emicSerial.print('\n');       //send CR in case Emic 2 is already up
                              //When the Emic 2 has initialized
                              //and is ready, it will send a single
                              //':' character, wait until received
while (emicSerial.read() != ':');
delay(10);                    //short delay
emicSerial.flush();           //flush the receive buffer
}

void loop()
{
emicSerial.print('S');        //Command: speak some text
emicSerial.print("Hello. My name is the Emic 2 Text-to-Speech module.
   I would like to sing you a song.");
                              //Send string to convert to speech
emicSerial.print('\n');
digitalWrite(ledPin, HIGH);   //turn on LED while Emic 2 is
                              //outputting audio
                              //Wait here until the Emic 2
```

```
                              //responds with a ":" indicating it's
                              //ready to accept the next command
while (emicSerial.read() != ':');
digitalWrite(ledPin, LOW);    //turn off LED
delay(500);                   //500 ms delay

emicSerial.print("D1\n");     //sing demo song D1
digitalWrite(ledPin, HIGH);   //turn on LED while Emic 2 is
                              //outputting audio
                              //Wait here until the Emic 2
                              //responds with a ":" indicating it's
                              //ready to accept the next command
while (emicSerial.read() != ':');
digitalWrite(ledPin, LOW);    //turn off LED

while(1)                      //demonstration complete!
  {
  delay(500);                 //flash LED
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  }
}

//****************************************************************
```

3.7 EXTERNAL MEMORY-SD CARD

A MultiMediaCard/SanDisk (MMC/SD) card provides a handy method of providing a low power, non-volatile, and a small form factor (32 mm × 24 mm × 1.4 mm) bulk memory storage for a microcontroller. The microSD card has an even smaller form factor at 15 mm × 11 mm × 1 mm. The SD card is a smart peripheral device. It contains an onboard controller to manage SD operations. The SD card is useful for data logging applications in remote locations.

If our goal is to measure wind resources at remote locations as potential windfarm sites, an Arduino-based data logging system equipped with an SD card could be used. Data could be logged over a long period of time and retrieved for later analysis on a PC.

Adafruit manufacturers a data logger shield (Adafruit #1141) for the Arduino UNO R3 as shown in Figure 3.19. The shield provides multiple Gigabits of additional storage for the UNO R3. It features a Real Time Clock (RTC) with battery backup to timestamp collected data. Microcontrollers keep time based on elapsed clock ticks. They do not "understand" the

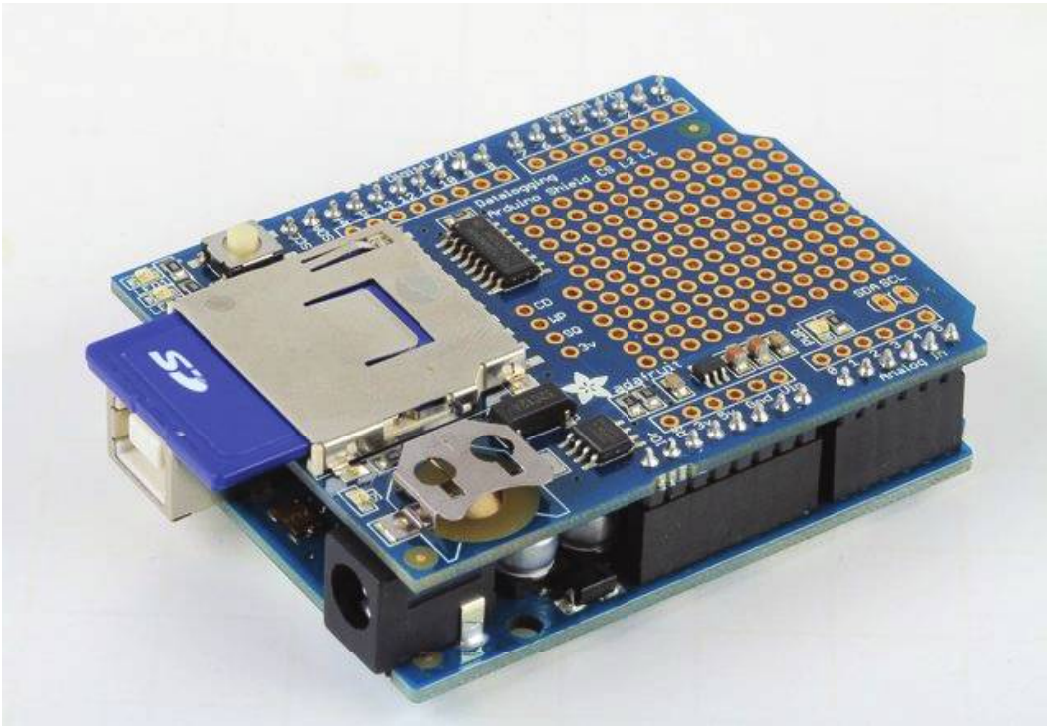
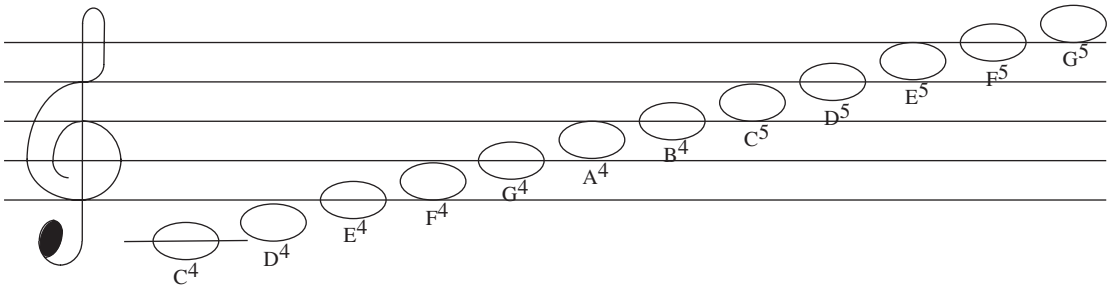


Figure 3.19: Adafruit data logger shield (www.adafruit.com). Illustration used with permission.

concepts of elapsed time in seconds, hours, etc. RTC features provide the microcontroller the ability to track calendar time based on seconds, minutes, hours, etc. We employ the Adafruit data logger in a weather station application in Chapter 4.

3.7.1 MUSICAL TONE GENERATOR

The Kuman K4 Arduino Starter Kit contains a tone generator that may be used to generate music. Some basic music theory is provided in Figure 3.20. At a fundamental level, music consists of a series of musical tones (frequencies), held for a specific length of time (beats), with occasional pauses (rests) in the music. With this basic information, many songs can be played by the tone generator.



Note	Frequency
C^4	262 Hz
D^4	294 Hz
E^4	330 Hz
F^4	349 Hz
G^4	392 Hz
A^4	440 Hz
B^4	494 Hz
C^5	523 Hz
D^5	587 Hz
E^5	659 Hz
F^5	698 Hz
G^5	784 Hz

Note Length	Beats	Length (s)*
whole	4 beats	4 s (4,000 ms)
half	2 beats	2 s (2,000 ms)
quarter	1 beat	1 s (1,000 ms)
eighth	1/2 beat	0.5 s (500 ms)

* At 60 beats per minute (BPM)

Rest Length	Beats	Length (s)*
whole	4 beats	4 s (4,000 ms)
half	2 beats	2 s (2,000 ms)
quarter	1 beat	1 s (1,000 ms)
eighth	1/2 beat	0.5 s (500 ms)

* At 60 beats per minute (BPM)

Figure 3.20: Music theory.

```
//*************************************************************
//plays a scale of whole notes and repeats
//*************************************************************

int tone_gen = 9;

void setup()
{
pinMode(tone_gen, OUTPUT);
}
void loop()
{
long freq_C4 = 262, freq_D4 = 294, freq_E4 = 300, freq_F4 = 349;
long freq_G4 = 392, freq_A4 = 440, freq_B4 = 494, freq_C5 = 784;

tone(tone_gen, freq_C4 ); delay(4000);
tone(tone_gen, freq_D4 ); delay(4000);
```

```
tone(tone_gen, freq_E4 ); delay(4000);
tone(tone_gen, freq_F4 ); delay(4000);
tone(tone_gen, freq_G4 ); delay(4000);
tone(tone_gen, freq_A4 ); delay(4000);
tone(tone_gen, freq_B4 ); delay(4000);
tone(tone_gen, freq_C5 ); delay(4000);

noTone(tone_gen);
delay(1000);
}

//************************************************************
```

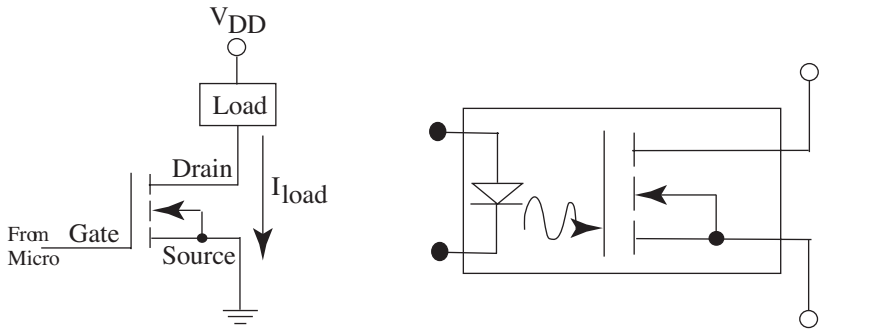
3.8 HIGH-POWER DC DEVICES

In this section we describe how to interface a control a variety of high power direct current (DC) supplied devices to the Arduino.

3.8.1 DC LOAD CONTROL

A number of direct current devices may be controlled with an electronic switching device such as a MOSFET. Specifically, an N-channel enhancement MOSFET (metal oxide semiconductor field effect transistor) may be used to switch a high current load on and off (such as a motor) using a low-voltage, low-current control signal from a microcontroller as shown in Figure 3.21a. The low-current control signal from the microcontroller is connected to the gate of the MOSFET. The MOSFET switches the high-current load on and off consistent with the control signal. The high-current load is connected between the load supply and the MOSFET drain. It is important to note that the load supply voltage and the microcontroller supply voltage do not have to be at the same value. When the control signal on the MOSFET gate is logic high, the load current flows from drain to source. When the control signal applied to the gate is logic low, no load current flows. Thus, the high-power load is turned on and off by the low power control signal from the microcontroller.

Often the MOSFET is used to control a high-power motor load. A motor is a notorious source of noise. To isolate the microcontroller from the motor noise an optical isolator may be used as an interface as shown in Figure 3.21b. The link between the control signal from the microcontroller to the high power load is via an optical link contained within a Solid State Relay (SSR). The SSR is properly biased using techniques previously discussed.



(a) N-channel Enhance MOSFET

(b) Solid State Relay with Optical Interface

Figure 3.21: MOSFET circuits.

3.8.2 DC SOLENOID CONTROL

The interface circuit for a DC solenoid is provided in Figure 3.22. A solenoid provides a mechanical insertion (or extraction) when asserted. In the interface, an optical isolator is used between the microcontroller and the MOSFET used to activate the solenoid. A reverse biased diode is placed across the solenoid. Both the solenoid power supply and the MOSFET must have the appropriate voltage and current rating to support the solenoid requirements.

Example: Water Valve Control. Solenoid controlled water valves are available from Adafruit (`www.adafruit.com`). There are plastic (#997) and brass (#996) valves available. The plastic valve activates from 6 VDC at 160 mA–12 VDC at 320 mA while the brass valve activates from 6 VDC at 1.6 A–12 VDC at 3A. An interface circuit for the plastic water solenoid valve is provided in Figure 3.23.

3.8.3 DC MOTOR SPEED AND DIRECTION CONTROL

Often, a microcontroller is used to control a high power motor load. To properly interface the motor to the microcontroller, we must be familiar with the different types of motor technologies. Motor types are illustrated in Figure 3.24.

- **DC motor:** A DC motor has a positive and negative terminal. When a DC power supply of suitable current rating is applied to the motor it will rotate. If the polarity of the supply is switched with reference to the motor terminals, the motor will rotate in the opposite direction. The speed of the motor is roughly proportional to the applied voltage up to the rated voltage of the motor.

- **Servo motor:** A servo motor provides a precision angular rotation for an applied pulse width modulation duty cycle. As the duty cycle of the applied signal is varied, the angular

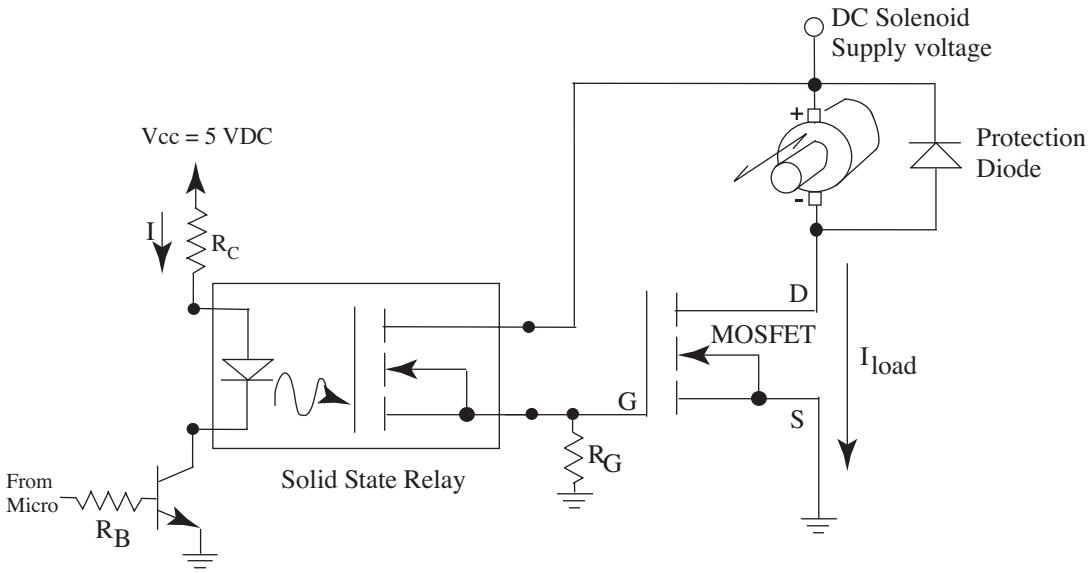


Figure 3.22: Solenoid interface circuit.

(a) Adafruit, 997, Plastic Water Solenoid Valve

(b) Adafruit, 997, Brass Water Solenoid Valve

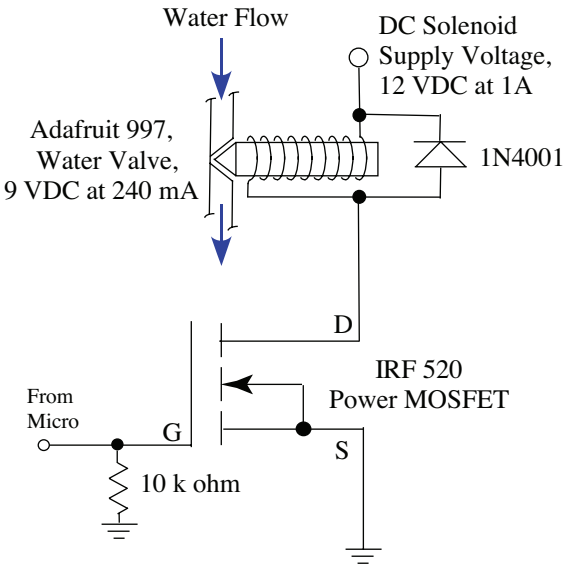


Figure 3.23: Water valve interface circuit (www.adafruit.com). Illustration used with permission.

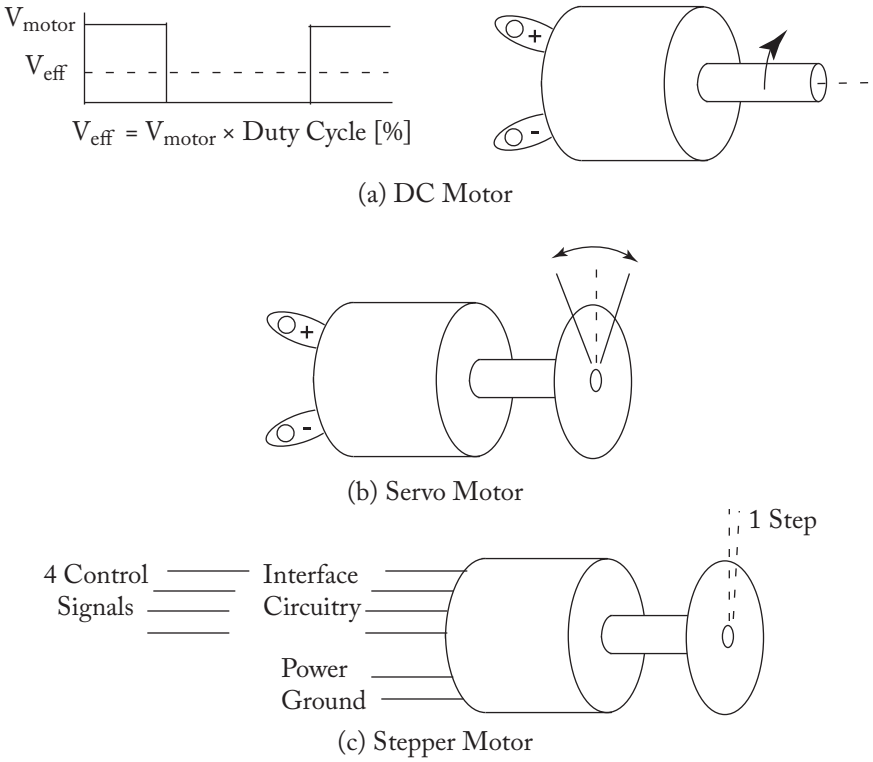


Figure 3.24: Motor types.

displacement of the motor also varies. This type of motor is used to change mechanical positions such as the steering angle of a wheel.

- **Stepper motor:** A stepper motor as its name implies provides an incremental step change in rotation (typically 2.5° per step) for a step change in control signal sequence. The motor is typically controlled by a two- or four-wire interface. For the four-wire stepper motor, the microcontroller provides a four-bit control sequence to rotate the motor clockwise. To turn the motor counterclockwise, the control sequence is reversed. The low-power control signals are interfaced to the motor via MOSFETs or power transistors to provide for the proper voltage and current requirements of the pulse sequence.

3.8.4 DC MOTOR OPERATING PARAMETERS

Space does not allow a full discussion of all motor types. We will concentrate on the DC motor. As previously mentioned, the motor speed may be varied by changing the applied voltage. This is difficult to do with a digital control signal. However, PWM control signal techniques may be

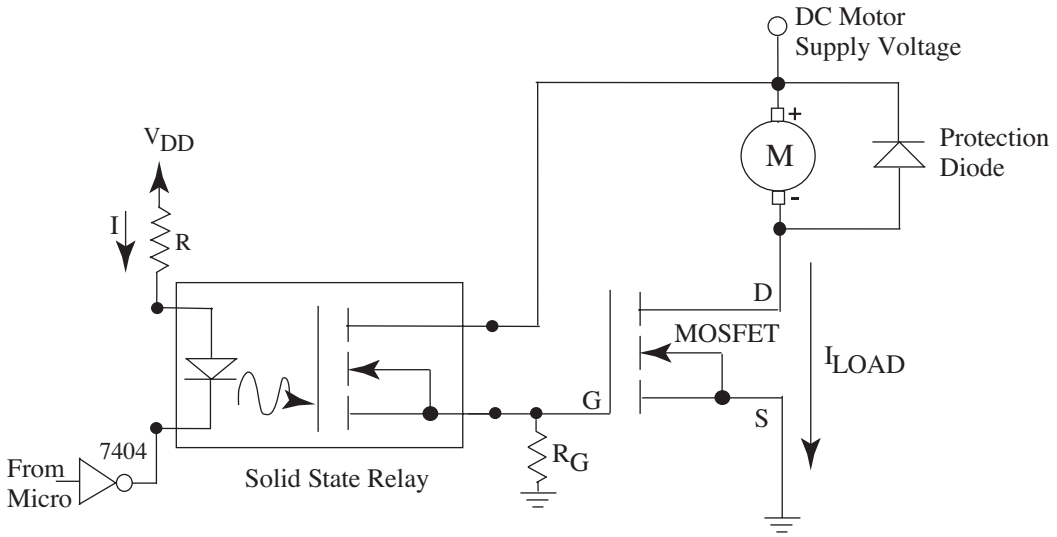


Figure 3.25: DC motor interface.

combined with a MOSFET interface to precisely control the motor speed. The duty cycle of the PWM signal will also be the percentage of the motor supply voltage applied to the motor, and hence the percentage of rated full speed at which the motor will rotate. The interface circuit to accomplish this type of control is shown in Figure 3.25. Various portions of this interface circuit have been previously discussed. The resistor R_G, typically 10 k ohm, is used to discharge the MOSFET gate when no voltage is applied to the gate. For an inductive load, a reversed biased protection diode must be across the load. The interface circuit shown allows the motor to rotate in a given direction.

3.8.5 H-BRIDGE DIRECTION CONTROL

For a DC motor to operate in both the clockwise and counter clockwise direction, the polarity of the DC motor supplied must be changed. To operate the motor in the forward direction, the positive battery terminal must be connected to the positive motor terminal while the negative battery terminal must be provided to the negative motor terminal. To reverse the motor direction the motor supply polarity must be reversed. An H-bridge is a circuit employed to perform this polarity switch. The H-bridge circuit consists of four electronic switches, as shown in Figure 3.26. For forward motor direction switches 1 and 4 are closed; whereas, for reverse direction switches 2 and 3 are closed.

Low-power H-bridges (500 mA) come in a convenient dual in line package (e.g., 754110). For higher power motors, a H-bridge may be constructed from discrete components, as shown in Figure 3.26. The ZTX451 and ZTX551 are NPN and PNP transistors with similar character-

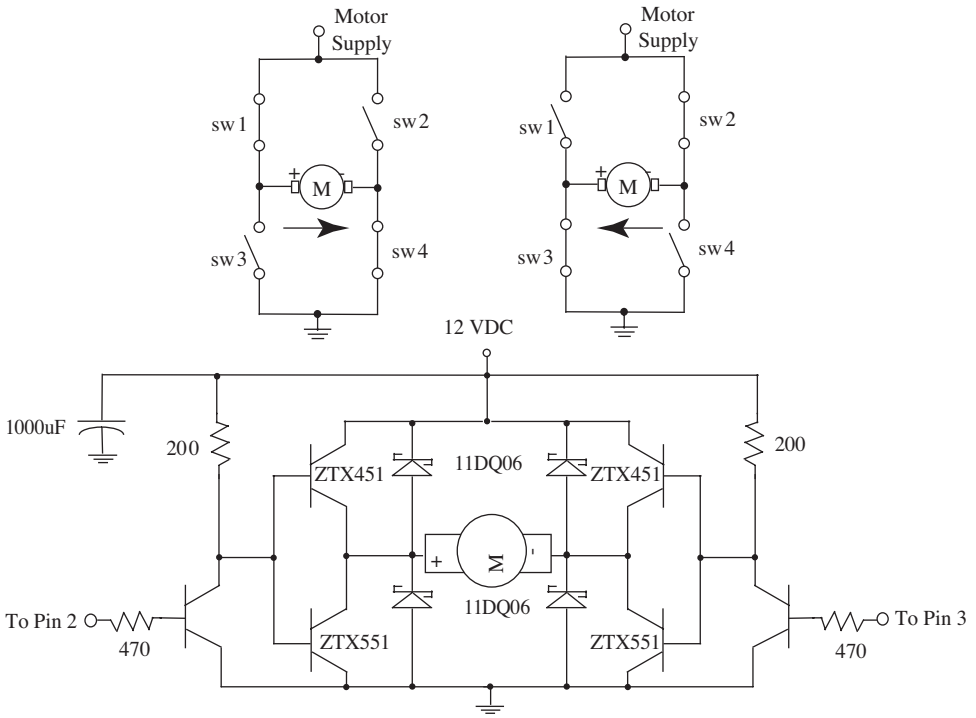


Figure 3.26: H-bridge control circuit.

istics. The 11DQ06 are Schottky diodes. For driving higher power loads, the switching devices are sized appropriately.

If PWM signals are used to drive the base of the transistors (from microcontroller pins pin 2 and pin 3), both motor speed and direction may be controlled by the circuit. The transistors used in the circuit must have a current rating sufficient to handle the current requirements of the motor during start and stall conditions.

Example: A linear actuator is a specially designed motor that converts rotary to linear motion. The linear actuator is equipped with a mechanical rod that is extended when asserted in one direction and retracted when the polarity of assertion is reversed. An H-bridge may be used to control a linear actuator as shown in Figure 3.27. In this circuit an enable signal is used to assert the H-bridge while the two PWM channels are used to extend or retract the H-bridge.

Texas Instruments provides a self-contained H-bridge motor controller integrated circuit, the DRV 8829. Within the DRV 8829 package is a single H-bridge driver. The driver may control DC loads with supply voltages from 8–45 VDC with a peak current rating of 5 amps. The single H-bridge driver may be used to control a DC motor or one winding of a bipolar stepper motor DRV 8829 [14]).

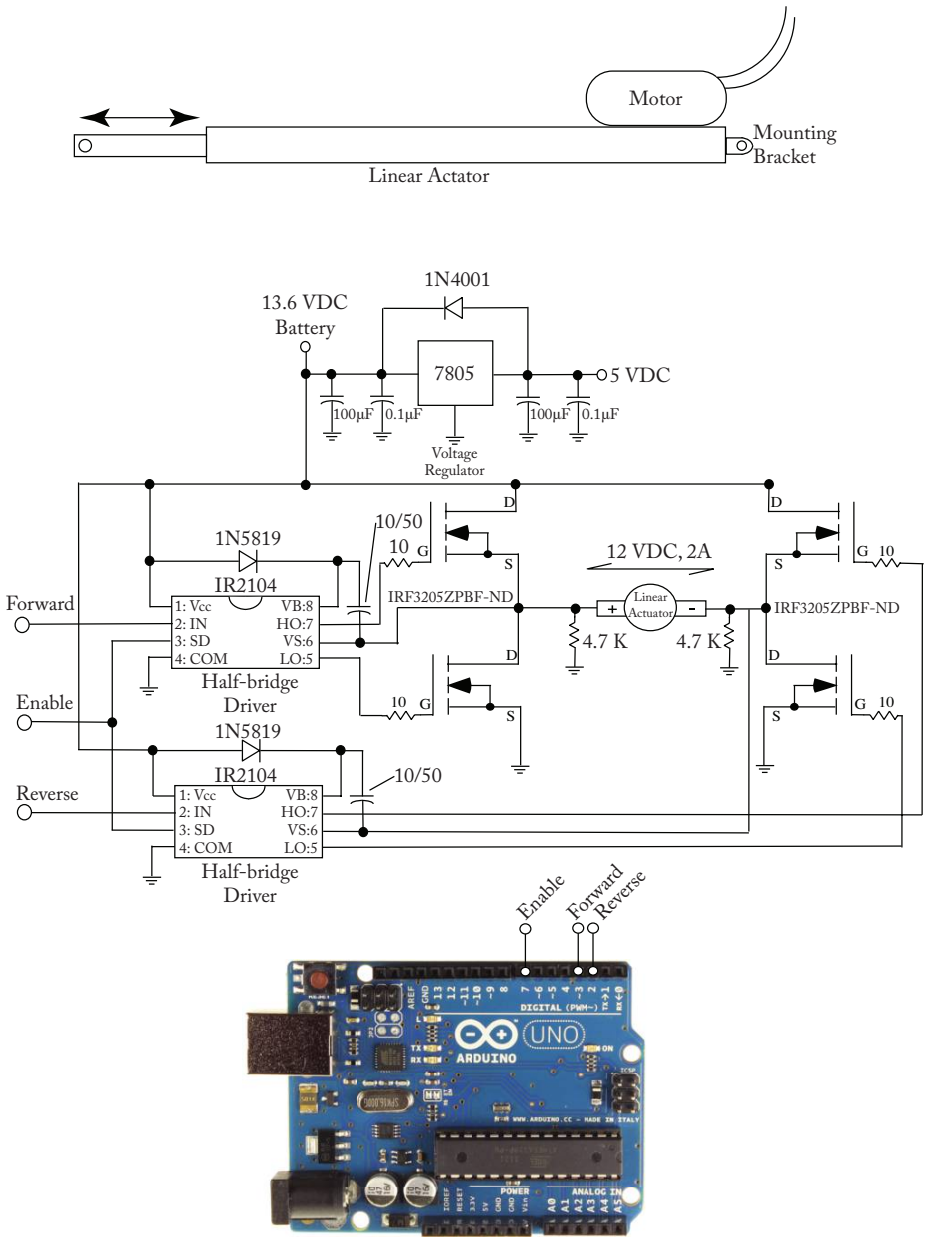


Figure 3.27: Linear actuator control circuit [O'Berto]. (UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA); www.arduino.cc.)

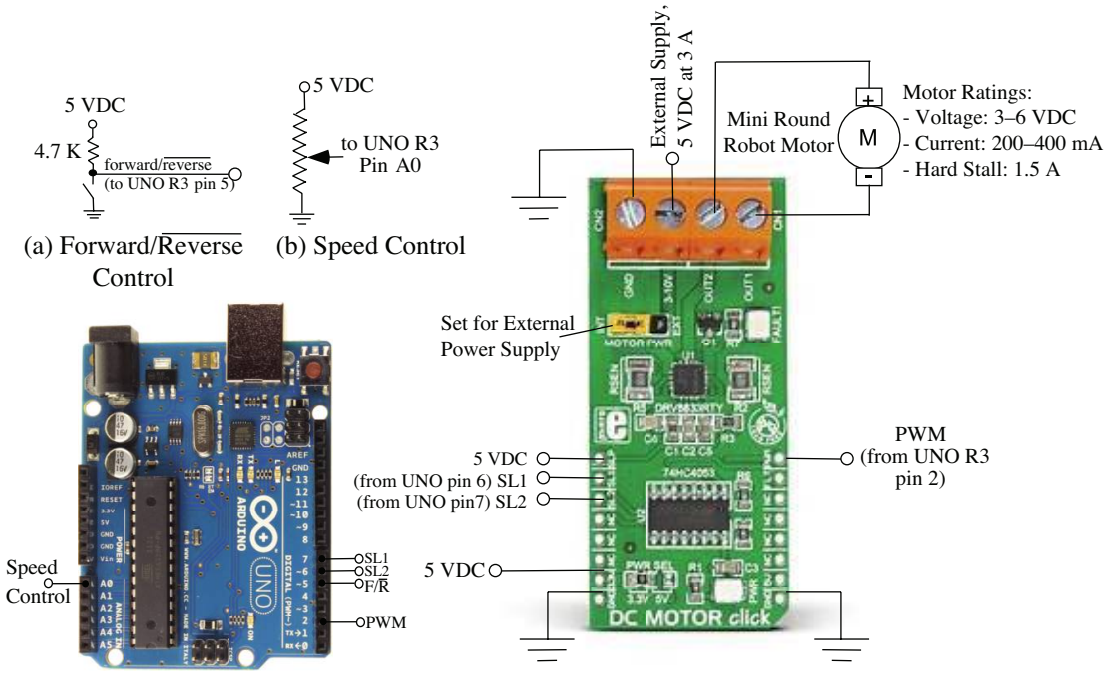


Figure 3.28: MIKROE-1526 DC MOTOR click. UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA); www.arduino.cc. MIKROE illustration used with permission (www.mikroe.com).

Example: MIKROE-1526 DC MOTOR Click. MikroElectronica (www.mikroe.com) manufactures a number of motor interface products including the MIKROE-1526 DC MOTOR click motor driver board. The board features the T.I. DRV 8833RTY H-bridge motor driver. A test circuit to control a DC motor's speed and direction is provided in Figure 3.28. We use one of the motors from the Mini Round Autonomous Maze Navigating Robot.

In the test circuit, a tact switch is used to determine motor direction and a potentiometer for motor speed control. These two inputs are read by the Arduino program and proper control signals are issued to the MIKROE-1526 (SL1, SL2, and PWM) for motor speed and direction. The nSLP pin on the MIKROE-1526 must be logic high to enable the device. For the test circuit, the nSLP pin is tied to Vcc (5 VDC) (www.mikroe.com).

```
//****************************************************************
//MIKROE-1526 DC Motor click
//Sketch demonstrates operation of the MIKROE-1526 DC Motor click
//
//The circuit:
//  - Motor speed control potentiometer.
//    Potentiometer connected to analog pin A0.  The center wiper
//    pin of the potentiometer goes to the analog pin.  The side
//    pins of the potentiometer go to 5 VDC and ground.
//  - Motor forward/reverse control.
//    Tact switch connected to UNO R3 pin 5.
//  - MIKROE-1526 connections:
//    -- Select 1 (SL1) to UNO R3 pin 6
//    -- Select 2 (SL2) to UNO R3 pin 7
//    -- PWM to UNO R3 pin 2
//    -- nSLEEP to Vcc (5 VDC) to enable device
//
//This example code is in the public domain.
//****************************************************************

int analog_in = A0;                //analog input (0 to 1023)
int analog_out = 2;                //analog output (0 to 255)
int forward_reverse = 5;           //direction control
int select1 = 6;                   //motor direction control
int select2 = 7;                   //SL1, SL2
int speed_value;                   //potentiometer input value
int switch_value;
int output_value;

void setup()
{
pinMode(forward_reverse, INPUT);
pinMode(select1, OUTPUT);
pinMode(select2, OUTPUT);
}

void loop()
{
//Deteremine motor direction
```

```
switch_value = digitalRead(forward_reverse);
if(switch_value == HIGH)   //forward direction
  {
  digitalWrite(select1, LOW);
  digitalWrite(select2, LOW);
  }
else                       //reverse direction
  {
  digitalWrite(select1, LOW);
  digitalWrite(select2, HIGH);
  }

//read analog in value
speed_value = analogRead(analog_in);

//map to analog out range
output_value = map(speed_value, 0, 1023, 0, 255);

//update analog out value
analogWrite(analog_out, output_value);

delay(50);
}

//*****************************************************************
```

3.8.6 SERVO MOTOR INTERFACE

The servo motor is used for a precise angular displacement. The displacement is related to the duty cycle of the applied control signal.

Example: Inexpensive Laser Light Show. An inexpensive laser light show may be constructed from two servos. In this example we use two Futaba 180° range servos (Parallax 900-00005, available from Jameco #283021) mounted, as shown in Figure 3.29. The X and Y control signals are provided by an Arduino processing board. The X and Y control signals are interfaced to the servos via LM324 operational amplifiers. The laser source is provided by an inexpensive laser pointer.

Sample code to drive the servos from an Arduino are provided on the Jameco website (www.jameco.com; Jameco #283021).

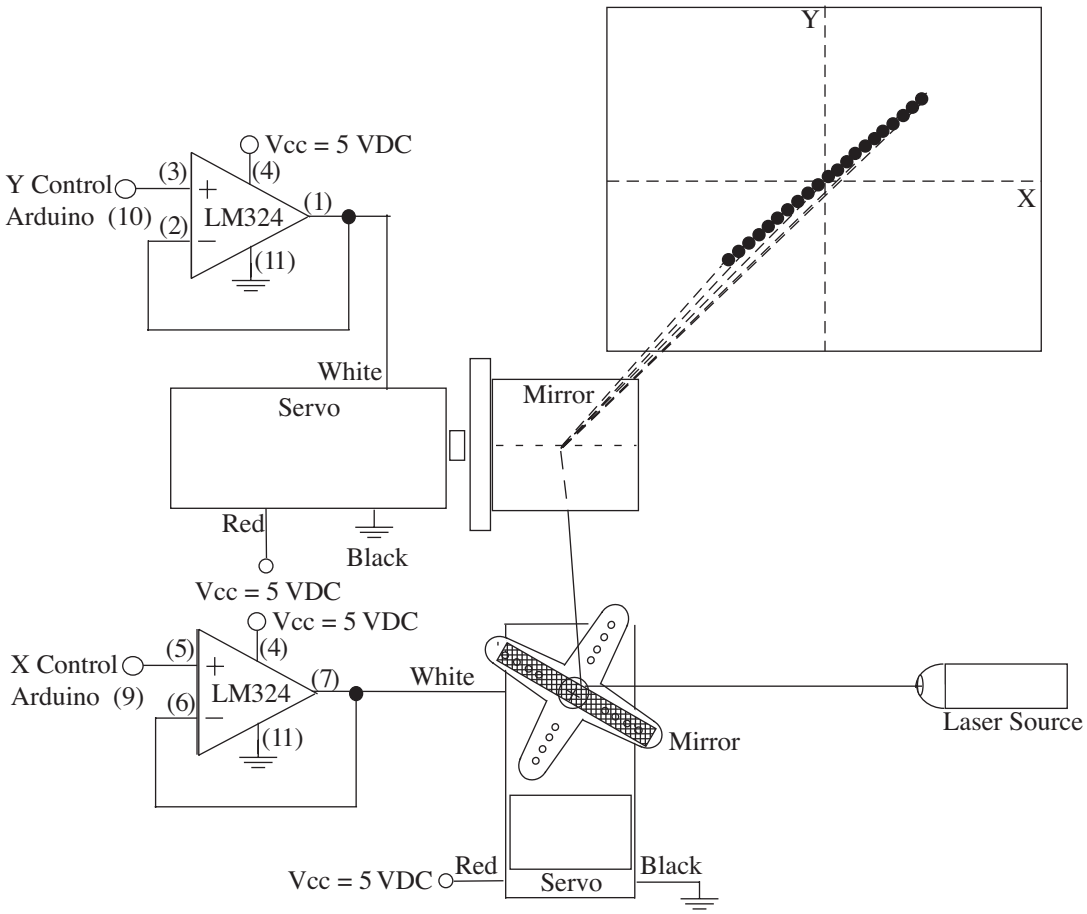


Figure 3.29: Inexpensive laser light show.

```
//*************************************************************
//X-Y ramp
//*************************************************************

#include <Servo.h>    // Use Servo library, included with IDE

Servo myServo_x;      // Create Servo object to control the servo
Servo myServo_y;

void setup() {
  myServo_x.attach(9);   // Servo is connected to digital pin 9
```

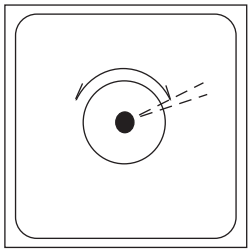
```
  myServo_y.attach(10);  // Servo is connected to digital pin 10

}

void loop() {
  int i = 0;
   for(i=0; i<=180; i++)
     {
     myServo_x.write(i);   // Rotate servo counter clockwise
     myServo_y.write(i);   // Rotate servo counter clockwise
     delay(20);          // Wait 2 seconds
     if(i==180)
       delay(5000);
     }
}
//*************************************************************
```

3.8.7 STEPPER MOTOR CONTROL

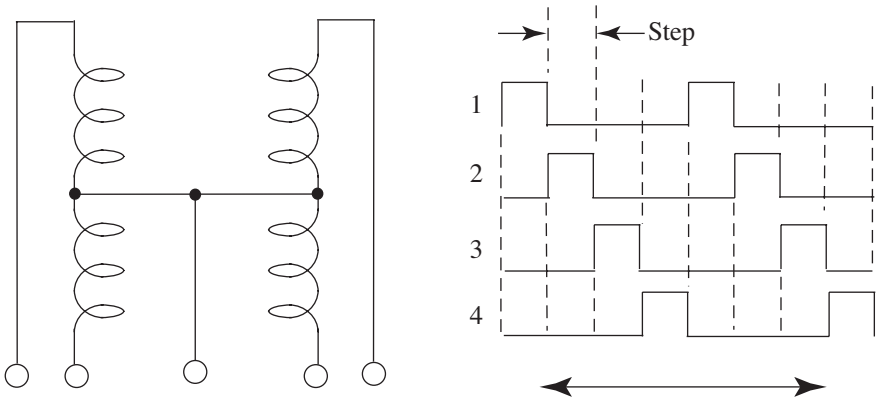
Stepper motors are used to provide a discrete angular displacement in response to a control signal step. There are a wide variety of stepper motors including bipolar and unipolar types with different configurations of motor coil wiring. Due to space limitations we only discuss the unipolar, five-wire stepper motor. The internal coil configuration for this motor is shown in Figure 3.30b.

Often, a wiring diagram is not available for the stepper motor. Based on the wiring configuration (see Figure 3.30b), one can find out the common line for both coils. It has a resistance that is one-half of all of the other coils. Once the common connection is found, one can connect the stepper motor into the interface circuit. By changing the other connections, one can determine the correct connections for the step sequence. To rotate the motor either clockwise or counterclockwise, a specific step sequence must be sent to the motor control wires, as shown in Figure 3.30b.

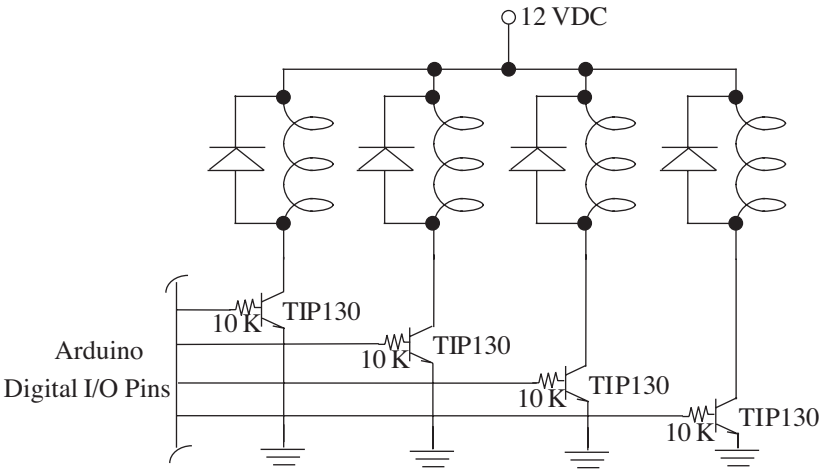
The microcontroller does not have sufficient capability to drive the motor directly. Therefore, an interface circuit is required, as shown in Figure 3.31. The speed of motor rotation is determined by how fast the control sequence is completed.



(a) A Stepper Motor Rotates a Fixed Angle Per Step



(b) Coil Configuration and Step Sequence



(c) Stepper Motor Interface Circuit

Figure 3.30: Unipolar stepper motor. (UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA); www.arduino.cc.)

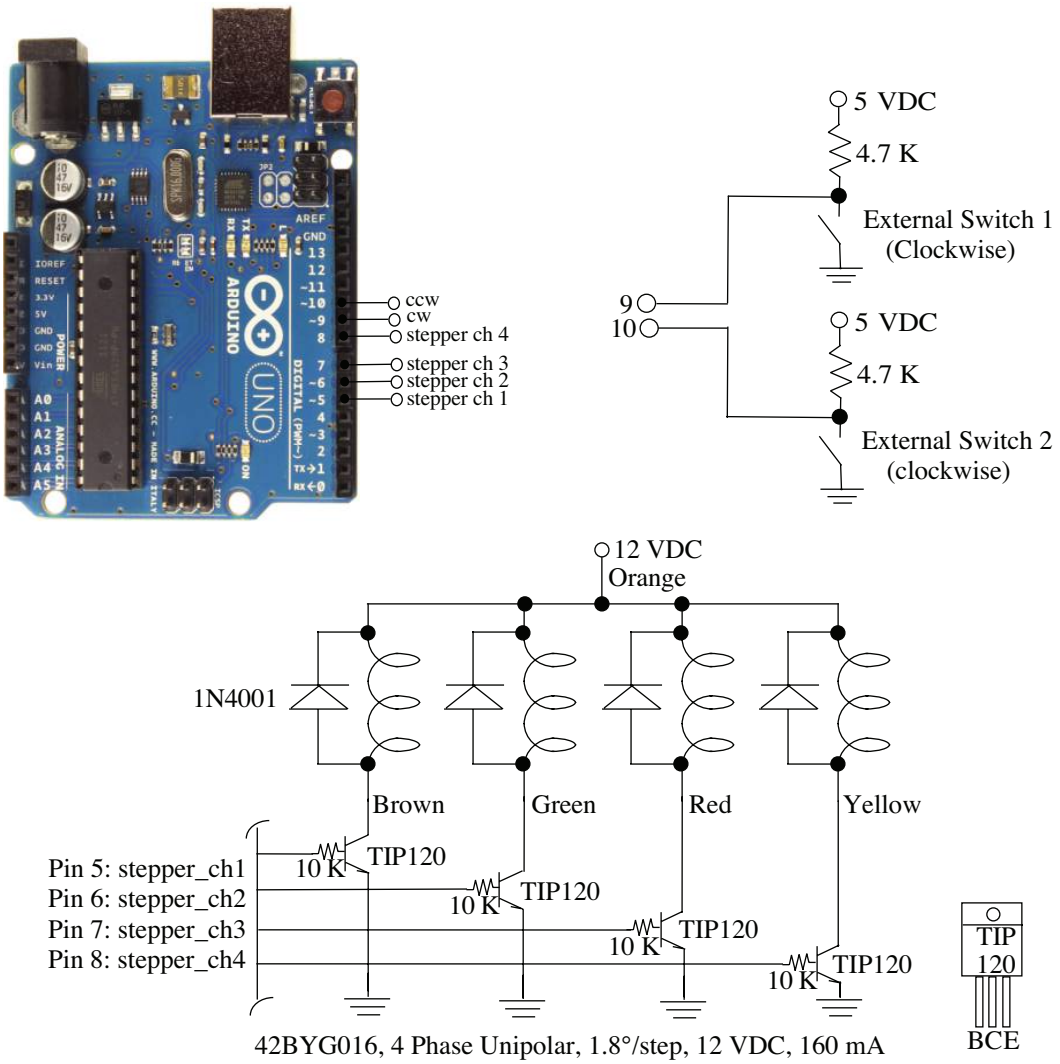


Figure 3.31: Unipolar stepper motor interface circuit. Illustration courtesy of Adafruit www.adafruit.com. UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA; www.arduino.cc).

```
//*************************************************************
//stepper
//
//This example code is in the public domain.
//*************************************************************

//external switches
#define ext_sw1  9
#define ext_sw2  10

//stepper channels
#define stepper_ch1  5
#define stepper_ch2  6
#define stepper_ch3  7
#define stepper_ch4  8

int switch_value1, switch_value2;
int motor_speed = 1000;           //motor increment time in ms
int last_step = 1;
int next_step;

void setup()
{
//Screen
Serial.begin(9600);

//external switches
pinMode(ext_sw1,  INPUT);
pinMode(ext_sw2,  INPUT);

//stepper channel
pinMode(stepper_ch1, OUTPUT);
pinMode(stepper_ch2, OUTPUT);
pinMode(stepper_ch3, OUTPUT);
pinMode(stepper_ch4, OUTPUT);

}

void loop()
```

```
{
switch_value1 = digitalRead(ext_sw1);
switch_value2 = digitalRead(ext_sw2);

if(switch_value1 == LOW)           //switch1 asserted
  {
  while(switch_value1 == LOW)      //clockwise
    {
    if(last_step == 1)
      {
      Serial.println("Switch 1: low, step 1");
      digitalWrite(stepper_ch1, HIGH);
      digitalWrite(stepper_ch2, LOW);
      digitalWrite(stepper_ch3, LOW);
      digitalWrite(stepper_ch4, LOW);
      next_step = 2;
      }
    else if(last_step == 2)
      {
      Serial.println("Switch 1: low, step 2");
      digitalWrite(stepper_ch1, LOW);
      digitalWrite(stepper_ch2, HIGH);
      digitalWrite(stepper_ch3, LOW);
      digitalWrite(stepper_ch4, LOW);
      next_step = 3;
      }
    else if(last_step == 3)
      {
      Serial.println("Switch 1: low, step 3");
      digitalWrite(stepper_ch1, LOW);
      digitalWrite(stepper_ch2, LOW);
      digitalWrite(stepper_ch3, HIGH);
      digitalWrite(stepper_ch4, LOW);
      next_step = 4;
      }
    else if(last_step == 4)
      {
      Serial.println("Switch 1: low, step 4");
      digitalWrite(stepper_ch1, LOW);
```

```
      digitalWrite(stepper_ch2, LOW);
      digitalWrite(stepper_ch3, LOW);
      digitalWrite(stepper_ch4, HIGH);
      next_step = 1;
      }
    else
      {
      ;
      }
    last_step = next_step;
    delay(motor_speed);
    switch_value1 = digitalRead(ext_sw1);
    }//end while
   }//end if

else if(switch_value2 == LOW)      //switch2 asserted
  {
  while(switch_value2 == LOW)      //counter clockwise
    {
    if(last_step == 1)
      {
      Serial.println("Switch 2: low, step 1");
      digitalWrite(stepper_ch1, HIGH);
      digitalWrite(stepper_ch2, LOW);
      digitalWrite(stepper_ch3, LOW);
      digitalWrite(stepper_ch4, LOW);
      next_step = 4;
      }
    else if(last_step == 2)
      {
      Serial.println("Switch 2: low, step 2");
      digitalWrite(stepper_ch1, LOW);
      digitalWrite(stepper_ch2, HIGH);
      digitalWrite(stepper_ch3, LOW);
      digitalWrite(stepper_ch4, LOW);
      next_step = 1;
      }
    else if(last_step == 3)
      {
```

```
      Serial.println("Switch 2: low, step 3");
      digitalWrite(stepper_ch1, LOW);
      digitalWrite(stepper_ch2, LOW);
      digitalWrite(stepper_ch3, HIGH);
      digitalWrite(stepper_ch4, LOW);
      next_step = 2;
      }
    else if(last_step == 4)
      {
      Serial.println("Switch 2: low, step 4");
      digitalWrite(stepper_ch1, LOW);
      digitalWrite(stepper_ch2, LOW);
      digitalWrite(stepper_ch3, LOW);
      digitalWrite(stepper_ch4, HIGH);
      next_step = 3;
      }
    else
      {
      ;
      }
    last_step = next_step;
    delay(motor_speed);
    switch_value2 = digitalRead(ext_sw2);
    }//end while
   }//end if

   else
     {
     digitalWrite(stepper_ch1, LOW);
     digitalWrite(stepper_ch2, LOW);
     digitalWrite(stepper_ch3, LOW);
     digitalWrite(stepper_ch4, LOW);
     }
}
//*********************************************************
```

Example. Adafruit (www.adafruit.com) manufactures a DC stepper motor breakout board (#3297) based on the Texas Instruments DRV 8833RTY H-bridge motor driver. The board can provide up to 1.2A per channel for motors from 2.7–10.8 VDC. In this example, we use the board to drive a Jameco (www.jameco.com) #238538 unipolar stepper motor rated at 12 VDC,

0.4 A. We power the motor at 10 VDC. The interface between the Arduino UNO R3, the breakout board, and the stepper motor is shown in Figure 3.32. Two external switches are used to select motor direction.

The code used in the previous stepper motor example may be modified with the step sequence required by the driver/motor combination.

3.9 AC DEVICES

A high-power AC load may be switched on and off using a low-power control signal from the microcontroller. In this case, a Solid State Relay is used as the switching device. Solid-state relays are available to switch a high power DC or AC load (Crydom Corporation [3]). For example, the Crydom 558-CX240D5R is a printed circuit board mounted, air cooled, single-pole single-throw (SPST), normally open (NO) solid-state relay. It requires a DC control voltage of 3–15 VDC at 15 mA. This microcontroller compatible DC control signal is used to switch 12–280 VAC loads rated from 0.06–5 amps [Crydom].

To vary the direction of an AC motor, you must use a bi-directional AC motor. A bi-directional motor is equipped with three terminals: common, clockwise, and counterclockwise. To turn the motor clockwise, an AC source is applied to the common and clockwise connections. In like manner, to turn the motor counterclockwise, an AC source is applied to the common and counterclockwise connections. This may be accomplished using two of the Crydom SSRs.

PowerSwitch manufacturers an easy-to-use AC interface the PowerSwitch Tail II. The device consists of a control module with attached AC connections rated at 120 VAC, 15 A for resistive loads. The device to be controlled is simply plugged in line with the PowerSwitch Tail II. A digital control signal 3 VDC at 3 mA to 12 VDC at 30 mA from the microcontroller serves as the on/off control signal for the controlled AC device. The controlled signal is connected to the PowerSwitch Tail II via a terminal block connection. The PowerSwitch II may be configured as either normally closed (NC) or normally open (NO) (`www.powerswitchtail.com`).

Example: PowerSwitch Tail II. In this example, we use an IR sensor to detect someone's presence. If the IR sensor's output reaches a predetermined threshold level, an AC desk lamp is illuminated, as shown in Figure 3.33.

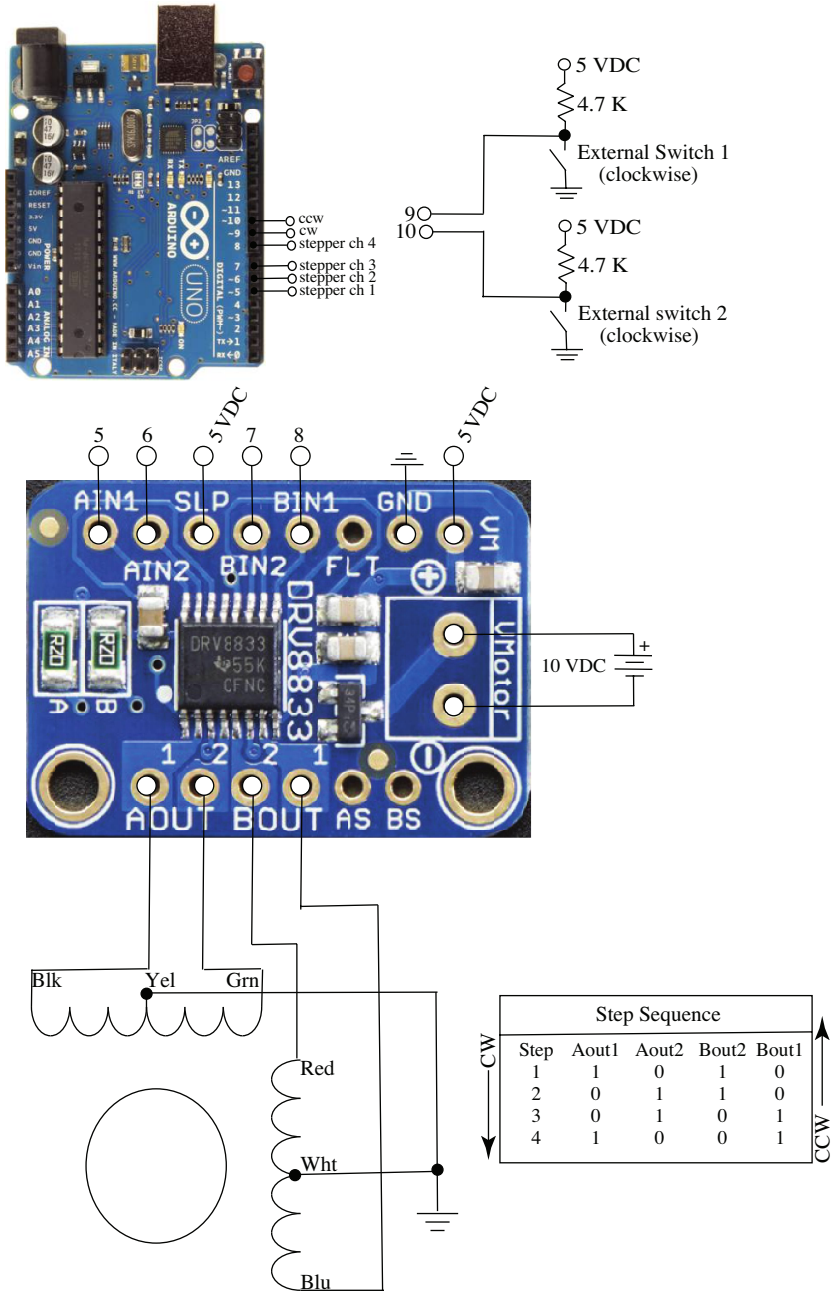


Figure 3.32: Unipolar stepper motor with DRV 8833 breakout board. (UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA); www.arduino.cc.)

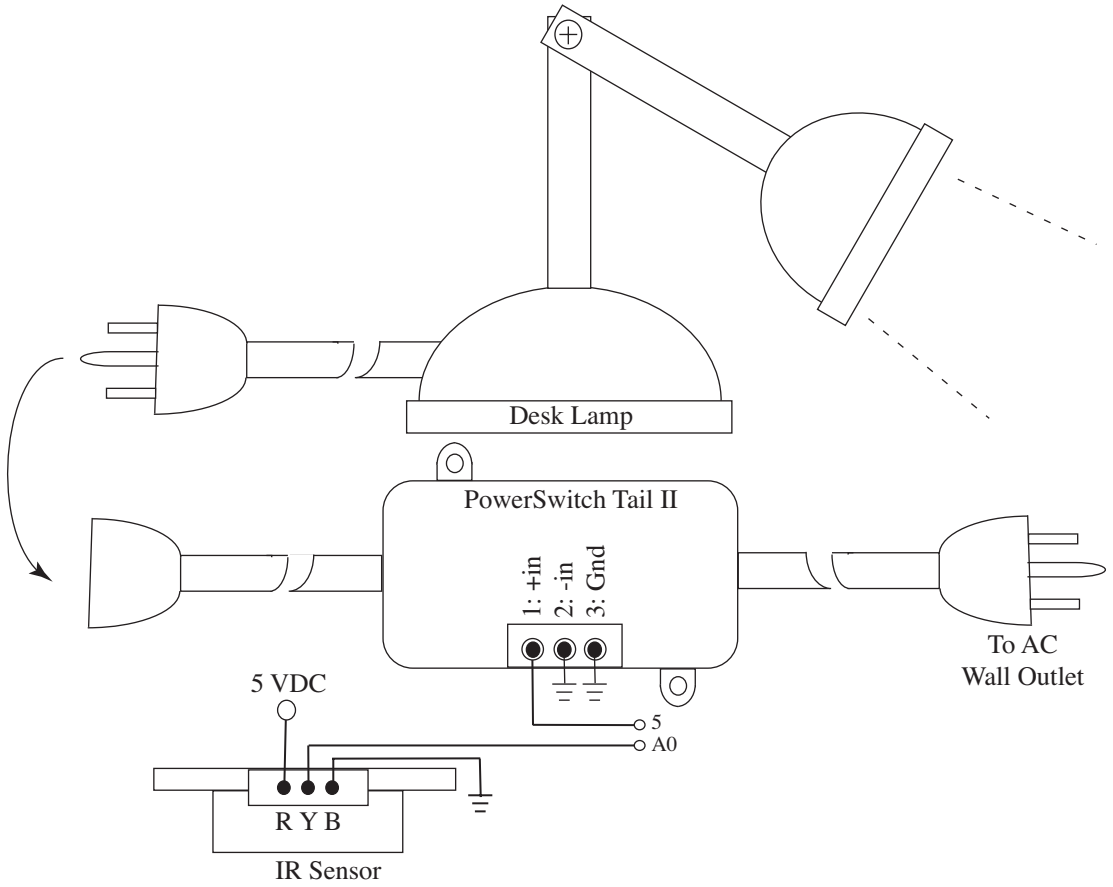


Figure 3.33: PowerSwitch Tail II.

```
//****************************************************************
//switch_tail
//
//The circuit:
//  - The IR sensor signal pin is connected to analog pin A0.
//    The sensor power and ground pins are connected to 5 VDC and
//    ground respectively.
//  - The analog output is designated as the onboard red LED.
//  - The switch tail control signal is connected to pin 5.
//
//Adapted for code originally written by Tom Igoe
```

```
//Created:  Dec 29, 2008
//Modified: Aug 15, 2019
//Author:   Tom Igoe
//
//This example code is in the public domain.
//************************************************************

const int analogInPin  = A0;      //Arduino analog input pin 5
const int analogOutPin = 13;      //Arduino onboard red LED pin
const int switch_tail_control =5; //Switch Tail control signal

int sensorValue = 0;              //value read from the OR sensor
int outputValue = 0;              //value output to the PWM (red LED)

void setup()
  {
  //initialize serial communications at 9600 bps:
  Serial.begin(9600);

  //configure Switch Tail control pin
  pinMode(switch_tail_control, OUTPUT);
  }

void loop()
  {
  //read the analog in value:
  sensorValue = analogRead(analogInPin);

  //map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);

  //change the analog out value:
  analogWrite(analogOutPin, outputValue);

  //Switch Tail control signal
  if(outputValue >= 128)
    {
    digitalWrite(switch_tail_control, HIGH);
    Serial.print("Light on");
```

```
    }
  else
    {
    digitalWrite(switch_tail_control, LOW);
    Serial.print("Light off");
    }

  // print the results to the serial monitor:
  Serial.print("sensor = " );
  Serial.print(sensorValue);
  Serial.print("\t output = ");
  Serial.println(outputValue);

  // wait 10 milliseconds before the next loop
  // for the analog-to-digital converter to settle
  // after the last reading:
  delay(10);
}

//*************************************************************
```

3.10 INTERFACING TO MISCELLANEOUS DEVICES

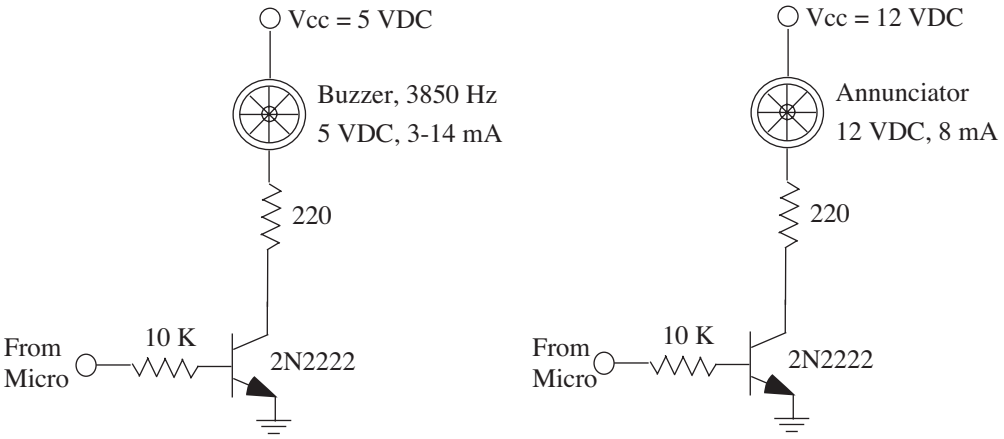
In this section, we provide a potpourri of interface circuits to connect a microcontroller to a wide variety of peripheral devices.

3.10.1 SONALERTS, BEEPERS, BUZZERS

In Figure 3.34, we provide several circuits used to interface a microcontroller to a buzzer, beeper, or other types of annunciator devices such as a sonalert. It is important that the interface transistor and the supply voltage are matched to the requirements of the sound-producing device.

3.10.2 VIBRATING MOTOR

A vibrating motor is often used to gain one's attention as in a cell phone. These motors are typically rated at 3 VDC and a high current. The interface circuit shown in Figure 3.35 is used to drive the low-voltage motor. Note that the control signal provided to the transistor base is 5 VDC. To step the motor supply voltage down to the motor voltage of 3 VDC, two 1N4001 silicon rectifier diodes are used in series. These diodes provide approximately 1.4–1.6 VDC voltage drop. Another 1N4001 diode is reversed biased across the motor and the series diode string. The motor may be turned on and off with a 5 VDC control signal from the microcontroller or



(a) 5 VDC Buzzer Interface

(b) 12 VDC Annunciator

Figure 3.34: Sonalert, beepers, buzzers.

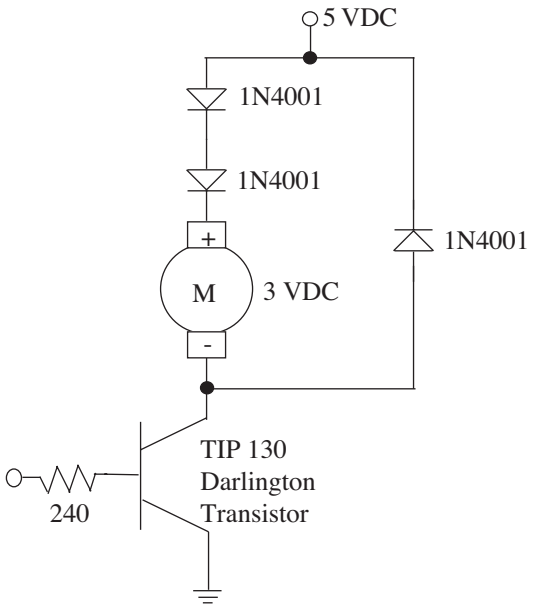


Figure 3.35: Controlling a low-voltage motor.

a PWM signal may be used to control motor speed. It is recommended that a Darlington NPN transistor (TIP 120) be employed in this application.

3.11 APPLICATION: SPECIAL EFFECTS LED CUBE

To illustrate some of the fundamentals of microcontroller interfacing, we construct a three-dimensional LED cube. This design was inspired by an LED cube kit available from Jameco (www.jameco.com). The LED cube consists of 4-layers of LEDs with 16 layers per LED. Only a single LED is illuminated at a specific time.

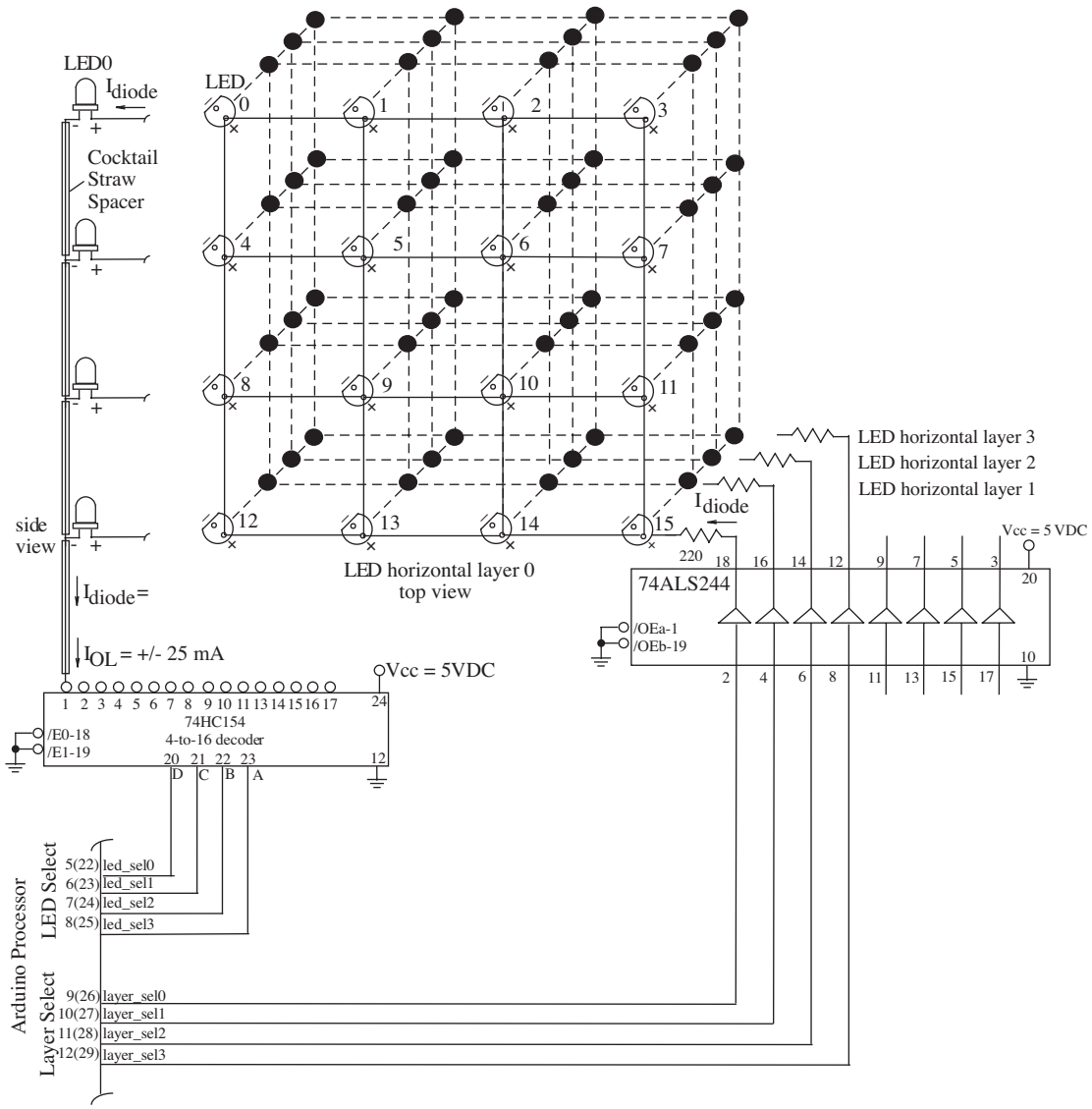
Only a single LED is illuminated at a given time. However, different effects may be achieved by how long a specific LED is left illuminated. A specific LED layer is asserted using the layer select pins on the microcontroller using a one-hot-code (a single line asserted while the others are de-asserted). The asserted line is fed through a 74HC244 (three state, octal buffer, line driver) which provides an I_{OH}/I_{OL} current of $\pm$ 35 mA, as shown in Figure 3.36. A given output from the 74HC244 is fed to a common anode connections for all 16 LEDs in a layer. All four LEDs in a specific LED position share a common cathode connection. That is, an LED in a specific location within a layer shares a common cathode connection with three other LEDs that share the same position in the other three layers. The common cathode connection from each LED location is fed to a specific output of the 74HC154 4-to-16 decoder. The decoder has a one-cold-code output. To illuminate a specific LED, the appropriate layer select and LED select lines are asserted using the layer_sel[3:0] and led_sel[3:0] lines, respectively. This basic design may be easily expanded to a larger LED cube.

3.11.1 CONSTRUCTION HINTS

To limit project costs, low-cost red LEDs (Jameco #333973) were used. The project requires a total of 64 LEDs (4 layers of 16 LEDs each). A LED template pattern was constructed from a 5-inch by 5-inch piece of pine wood. A 4-by-4 pattern of holes were drilled into the wood. Holes were spaced 3/4-inch apart. The hole diameter was slightly smaller than the diameter of the LEDs to allow for a snug LED fit.

The LED array was constructed a layer at a time using the wood template. Each LED was tested before inclusion in the array. A 5 VDC power supply with a series 220 ohm resistor was used to insure each LED was fully operational. The LED anodes in a given LED row were then soldered together. A fine-tip soldering iron and a small bit of solder were used for each interconnect, as shown in Figure 3.37. Cross wires were then used to connect the cathodes of adjacent rows. A 22-gauge bare wire was used. Again, a small bit of solder was used for the interconnect points. Four separate LED layers (4 by 4 array of LEDs) were completed.

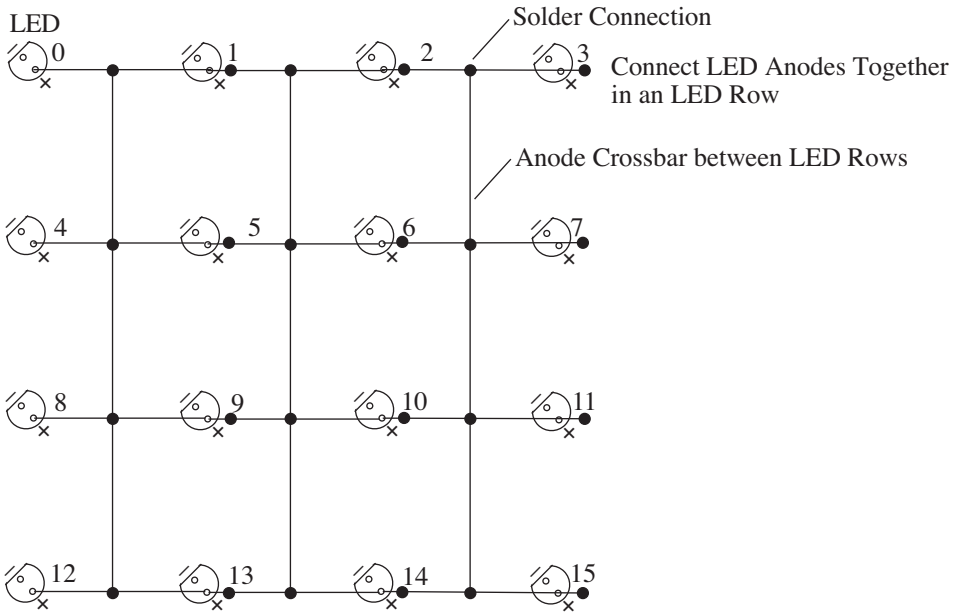
To assemble the individual layers into a cube, cocktail straw segments were used as spacers between the layers. The straw segments provided spacing between the layers and also offered improved structural stability. The anodes for a given LED position were soldered together. For example, all LEDs in position 0 for all four layers shared a common anode connection.



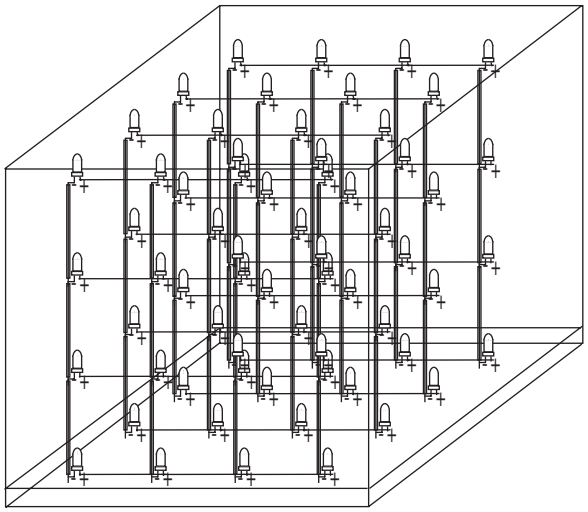
Notes:

1. LED cube consists of 4 layers of 16 LEDs each.
2. Each LED is individually addressed by asserting the appropriate cathode signal (0–15) and asserting a specific LED layer.
3. All LEDs in a given layer share a common anode connection.
4. All LEDs in a given position (0–15) share a common cathode connection.

Figure 3.36: LED special effects cube.



(a) LED Soldering Diagram



(b) 3D LED Array Mounted within Plexiglass Cube

Figure 3.37: LED cube construction.

The completed LED cube was mounted on a perforated printed circuit board (perfboard) to provide a stable base. LED sockets for the 74LS244 and the 74HC154 were also mounted to the perfboard. Connections were routed to a 16-pin ribbon cable connector. The other end of the ribbon cable was interfaced to the appropriate pins of the Arduino processor. The entire LED cube was mounted within a 4-inch plexiglass cube. The cube is available from the Container Store (www.containerstore.com). A construction diagram is provided in Figure 3.37.

3.11.2 LED CUBE ARDUINO SKETCH CODE

Provided below is the basic code template to illuminate a single LED (LED 0, layer 0). This basic template may be used to generate a number of special effects (e.g., tornado, black hole, etc.). Pin numbers are provided for the Arduino UNO R3. Pin numbers for the Arduino Mega 2560 are provided in the comments.

```
//*******************************************
                            //led select pins
#define led_sel0  5         //Mega2560: pin 22
#define led_sel1  6         //Mega2560: pin 23
#define led_sel2  7         //Mega2560: pin 24
#define led_sel3  8         //Mega2560: pin 25

                            //layer select pins
#define layer_sel0   9      //Mega2560: pin 26
#define layer_sel1  10      //Mega2560: pin 27
#define layer_sel2  11      //Mega2560: pin 28
#define layer_sel3  12      //Mega2560: pin 29

void setup()
{
pinMode(led_sel0, OUTPUT);
pinMode(led_sel1, OUTPUT);
pinMode(led_sel2, OUTPUT);
pinMode(led_sel3, OUTPUT);

pinMode(layer_sel0, OUTPUT);
pinMode(layer_sel1, OUTPUT);
pinMode(layer_sel2, OUTPUT);
pinMode(layer_sel3, OUTPUT);
}

void loop()
```

```
{
                          //illuminate LED 0, layer 0
                          //led select
digitalWrite(led_sel0, LOW);
digitalWrite(led_sel1, LOW);
digitalWrite(led_sel2, LOW);
digitalWrite(led_sel3, LOW);
                          //layer select
digitalWrite(layer_sel0, HIGH);
digitalWrite(layer_sel1, LOW);
digitalWrite(layer_sel2, LOW);
digitalWrite(layer_sel3, LOW);

delay(500);               //delay specified in ms
}

//*******************************************
```

In the next example, a function "illuminate_LED" has been added. To illuminate a specific LED, the LED position (0–15), the LED layer (0–3), and the length of time to illuminate the LED in milliseconds is specified. In this short example, LED 0 is sequentially illuminated in each layer. An LED grid map is provided in Figure 3.38. It is useful for planning special effects.

```
//*******************************************
                           //led select pins
#define led_sel0  5        //Mega2560: pin 22
#define led_sel1  6        //Mega2560: pin 23
#define led_sel2  7        //Mega2560: pin 24
#define led_sel3  8        //Mega2560: pin 25

                            //layer select pins
#define layer_sel0   9      //Mega2560: pin 26
#define layer_sel1  10      //Mega2560: pin 27
#define layer_sel2  11      //Mega2560: pin 28
#define layer_sel3  12      //Mega2560: pin 29

void setup()
{
pinMode(led_sel0, OUTPUT);
pinMode(led_sel1, OUTPUT);
pinMode(led_sel2, OUTPUT);
```

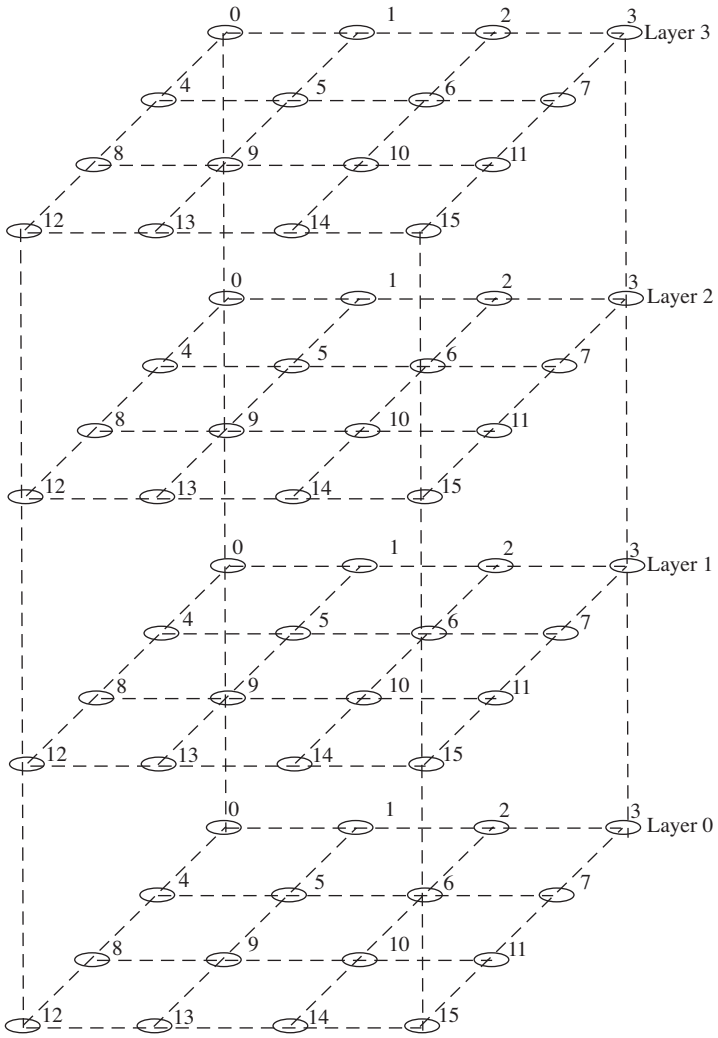


Figure 3.38: LED grid map.

```
pinMode(led_sel3, OUTPUT);

pinMode(layer_sel0, OUTPUT);
pinMode(layer_sel1, OUTPUT);
pinMode(layer_sel2, OUTPUT);
pinMode(layer_sel3, OUTPUT);
}

void loop()
{
illuminate_LED(0, 0, 500);
illuminate_LED(0, 1, 500);
illuminate_LED(0, 2, 500);
illuminate_LED(0, 3, 500);
}

//*******************************************

void illuminate_LED(int led, int layer, int delay_time)
{
if(led==0)
  {
  digitalWrite(led_sel0, LOW);
  digitalWrite(led_sel1, LOW);
  digitalWrite(led_sel2, LOW);
  digitalWrite(led_sel3, LOW);
  }
else if(led==1)
  {
  digitalWrite(led_sel0, HIGH);
  digitalWrite(led_sel1, LOW);
  digitalWrite(led_sel2, LOW);
  digitalWrite(led_sel3, LOW);
  }
else if(led==2)
  {
  digitalWrite(led_sel0, LOW);
  digitalWrite(led_sel1, HIGH);
  digitalWrite(led_sel2, LOW);
```

```
  digitalWrite(led_sel3, LOW);
  }
else if(led==3)
  {
  digitalWrite(led_sel0, HIGH);
  digitalWrite(led_sel1, HIGH);
  digitalWrite(led_sel2, LOW);
  digitalWrite(led_sel3, LOW);
  }
else if(led==4)
  {
  digitalWrite(led_sel0, LOW);
  digitalWrite(led_sel1, LOW);
  digitalWrite(led_sel2, HIGH);
  digitalWrite(led_sel3, LOW);
  }
else if(led==5)
  {
  digitalWrite(led_sel0, HIGH);
  digitalWrite(led_sel1, LOW);
  digitalWrite(led_sel2, HIGH);
  digitalWrite(led_sel3, LOW);
  }
else if(led==6)
  {
  digitalWrite(led_sel0, LOW);
  digitalWrite(led_sel1, HIGH);
  digitalWrite(led_sel2, HIGH);
  digitalWrite(led_sel3, LOW);
  }
else if(led==7)
  {
  digitalWrite(led_sel0, HIGH);
  digitalWrite(led_sel1, HIGH);
  digitalWrite(led_sel2, HIGH);
  digitalWrite(led_sel3, LOW);
  }
if(led==8)
  {
```

```
  digitalWrite(led_sel0, LOW);
  digitalWrite(led_sel1, LOW);
  digitalWrite(led_sel2, LOW);
  digitalWrite(led_sel3, HIGH);
  }
else if(led==9)
  {
  digitalWrite(led_sel0, HIGH);
  digitalWrite(led_sel1, LOW);
  digitalWrite(led_sel2, LOW);
  digitalWrite(led_sel3, HIGH);
  }
else if(led==10)
  {
  digitalWrite(led_sel0, LOW);
  digitalWrite(led_sel1, HIGH);
  digitalWrite(led_sel2, LOW);
  digitalWrite(led_sel3, HIGH);
  }
else if(led==11)
  {
  digitalWrite(led_sel0, HIGH);
  digitalWrite(led_sel1, HIGH);
  digitalWrite(led_sel2, LOW);
  digitalWrite(led_sel3, HIGH);
  }
else if(led==12)
  {
  digitalWrite(led_sel0, LOW);
  digitalWrite(led_sel1, LOW);
  digitalWrite(led_sel2, HIGH);
  digitalWrite(led_sel3, HIGH);
  }
else if(led==13)
  {
  digitalWrite(led_sel0, HIGH);
  digitalWrite(led_sel1, LOW);
  digitalWrite(led_sel2, HIGH);
  digitalWrite(led_sel3, HIGH);
```

```
  }
else if(led==14)
  {
  digitalWrite(led_sel0, LOW);
  digitalWrite(led_sel1, HIGH);
  digitalWrite(led_sel2, HIGH);
  digitalWrite(led_sel3, HIGH);
  }
else if(led==15)
  {
  digitalWrite(led_sel0, HIGH);
  digitalWrite(led_sel1, HIGH);
  digitalWrite(led_sel2, HIGH);
  digitalWrite(led_sel3, HIGH);
  }

if(layer==0)
  {
  digitalWrite(layer_sel0, HIGH);
  digitalWrite(layer_sel1, LOW);
  digitalWrite(layer_sel2, LOW);
  digitalWrite(layer_sel3, LOW);
  }
else if(layer==1)
  {
  digitalWrite(layer_sel0, LOW);
  digitalWrite(layer_sel1, HIGH);
  digitalWrite(layer_sel2, LOW);
  digitalWrite(layer_sel3, LOW);
  }
else if(layer==2)
  {
  digitalWrite(layer_sel0, LOW);
  digitalWrite(layer_sel1, LOW);
  digitalWrite(layer_sel2, HIGH);
  digitalWrite(layer_sel3, LOW);
  }
else if(layer==3)
  {
```

```
  digitalWrite(layer_sel0, LOW);
  digitalWrite(layer_sel1, LOW);
  digitalWrite(layer_sel2, LOW);
  digitalWrite(layer_sel3, HIGH);
  }

delay(delay_time);
}

//*******************************************
```

In the next example, a "fireworks" special effect is produced. The firework goes up, splits into four pieces, and then falls back down, as shown in Figure 3.39. It is useful for planning special effects.

```
//*******************************************
                          //led select pins
#define led_sel0  5       //Mega2560: pin 22
#define led_sel1  6       //Mega2560: pin 23
#define led_sel2  7       //Mega2560: pin 24
#define led_sel3  8       //Mega2560: pin 25

                           //layer select pins
#define layer_sel0   9     //Mega2560: pin 26
#define layer_sel1  10     //Mega2560: pin 27
#define layer_sel2  11     //Mega2560: pin 28
#define layer_sel3  12     //Mega2560: pin 29

void setup()
{
pinMode(led_sel0, OUTPUT);
pinMode(led_sel1, OUTPUT);
pinMode(led_sel2, OUTPUT);
pinMode(led_sel3, OUTPUT);

pinMode(layer_sel0, OUTPUT);
pinMode(layer_sel1, OUTPUT);
pinMode(layer_sel2, OUTPUT);
pinMode(layer_sel3, OUTPUT);
}
```

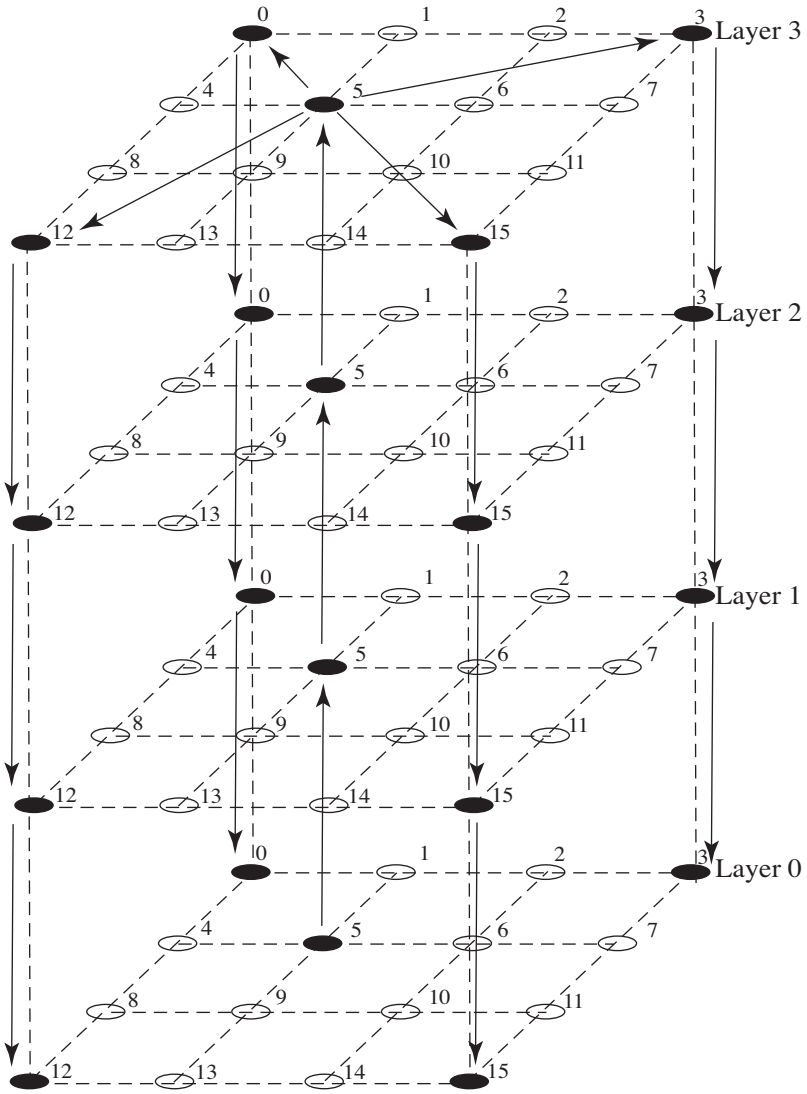


Figure 3.39: LED grid map for a fire work.

```
void loop()
{
int i;

//firework going up
illuminate_LED(5, 0, 100);
illuminate_LED(5, 1, 100);
illuminate_LED(5, 2, 100);
illuminate_LED(5, 3, 100);

//firework exploding into four pieces
//at each cube corner
for(i=0;i<=10;i++)
  {
  illuminate_LED(0,  3, 10);
  illuminate_LED(3,  3, 10);
  illuminate_LED(12, 3, 10);
  illuminate_LED(15, 3, 10);
  delay(10);
  }

delay(200);

//firework pieces falling to layer 2
for(i=0;i<=10;i++)
  {
  illuminate_LED(0,  2, 10);
  illuminate_LED(3,  2, 10);
  illuminate_LED(12, 2, 10);
  illuminate_LED(15, 2, 10);
  delay(10);
  }

delay(200);

//firework pieces falling to layer 1
for(i=0;i<=10;i++)
  {
  illuminate_LED(0,  1, 10);
```

```
  illuminate_LED(3,  1, 10);
  illuminate_LED(12, 1, 10);
  illuminate_LED(15, 1, 10);
  delay(10);
  }

delay(200);

//firework pieces falling to layer 0
for(i=0;i<=10;i++)
  {
  illuminate_LED(0,  0, 10);
  illuminate_LED(3,  0, 10);
  illuminate_LED(12, 0, 10);
  illuminate_LED(15, 0, 10);
  delay(10);
  }

delay(10);
}

//*******************************************

void illuminate_LED(int led, int layer, int delay_time)
{
if(led==0)
  {
   :
   :  //code provided for this function in the previous example.

  }
}

//*******************************************
```

3.12 SUMMARY

In this chapter, we discussed the voltage and current operating parameters for the Arduino UNO R3 and Mega 2560 processing board and the Microchip ATmega328 microcontroller. We discussed how this information may be applied to properly design an interface for common input

and output circuits. It must be emphasized a properly designed interface allows the microcontroller to operate properly within its parameter envelope. If due to a poor interface design, a microcontroller is used outside its prescribed operating parameter values, spurious and incorrect logic values will result. We provided interface information for a wide range of input and output devices. We also discussed the concept of interfacing a motor to a microcontroller using PWM techniques coupled with high power MOSFET or SSR switching devices.

3.13 REFERENCES

[1] Pack, D. and Barrett, S. (2002). *68HC12 Microcontroller: Theory and Applications*. Prentice Hall Incorporated, Upper Saddle River, NJ.

[2] Barrett, S. and Pack, D. (2004). *Embedded Systems Design with the 68HC12 and HCS12*. Prentice Hall Incorporated, Upper Saddle River, NJ. 77

[3] Crydom Corporation, 2320 Paseo de las Americas, Suite 201, San Diego, CA, `www.crydom.com`. 132

[4] Sick/Stegmann Incorporated, Dayton, OH, `www.stegmann.com`. 92

[5] Images Company, 39 Seneca Loop, Staten Island, NY 10314. 93

[6] *Microchip 8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash, ATmega48PA, ATmega88PA, ATmega168PA, ATmega328P* data sheet: Rev. 8161D-AVR-10/09, Microchip Corporation, 2325 Orchard Parkway, San Jose, CA 95131.

[7] Barrett, S. and Pack, D. (2006). *Microcontrollers Fundamentals for Engineers and Scientists*. Morgan & Claypool Publishers. DOI: 10.2200/S00025ED1V01Y200605DCS001

[8] Barrett, S. and Pack, D. (2008). *Atmel AVR Microcontroller Primer Programming and Interfacing*. Morgan & Claypool Publishers. DOI: 10.2200/S00100ED1V01Y200712DCS015

[9] Barrett, S. (2010). *Embedded Systems Design with the Atmel AVR Microcontroller*. Morgan & Claypool Publishers. DOI: 10.2200/S00225ED1V01Y200910DCS025

[10] National Semiconductor, *LM34/LM34A/LM34C/LM34CA/LM34D Precision Fahrenheit Temperature Sensor*, 1995. 94

[11] SparkFun, `www.sparkfun.com`.

[12] Pack, D. and Barrett, S. (2011). *Microcontroller Programming and Interfacing Texas Instruments MSP430*. Morgan & Claypool Publishers.

[13] *Emic 2 Text-to-Speech Module (#30016)*, Parallax Corporation, `www.parallax.com`, 2015. 108

[14] DRV 8829, 5-A 45-V Single H-Bridge Motor Driver, SLVSA74E, Texas Instruments, Sep. 2015. 119

3.14 CHAPTER PROBLEMS

3.1. What will happen if a microcontroller is used outside of its prescribed operating envelope?

3.2. Discuss the difference between the terms "sink" and "source" as related to current loading of a microcontroller.

3.3. Can an LED with a series limiting resistor be directly driven by the microchip microcontroller? Explain.

3.4. In your own words, provide a brief description of each of the microcontroller electrical parameters.

3.5. What is switch bounce? Describe two techniques to minimize switch bounce.

3.6. Describe a method of debouncing a keypad.

3.7. What is the difference between an incremental encoder and an absolute encoder? Describe applications for each type.

3.8. What must be the current rating of the 2N2222 and 2N2907 transistors used in the tri-state LED circuit? Support your answer.

3.9. Draw the circuit for a six character seven-segment display. Fully specify all components. Write a program to display "ATmega328."

3.10. Repeat the question above for a dot matrix display.

3.11. Repeat the question above for a LCD display.

3.12. What is the difference between a unipolar and bipolar stepper motor?

3.13. What controls the speed of rotation of a stepper motor?

3.14. A stepper motor provides and angular displacement of 1.8° per step. How can this resolution be improved?

3.15. Write a function to convert an ASCII numeral representation (0–9) to a seven-segment display.

3.16. Why is an interface required between a microcontroller and a stepper motor?

3.17. How must the LED cube design be modified to incorporate 8 layers of LEDs with 16 LEDs per layer?

3.18. In the LED cube design, what is the maximum amount of forward current that can be safely delivered to a given LED?

CHAPTER 4

Arduino System Examples

Objectives: After reading this chapter, the reader should be able to:

- provide a detailed design for a remote weather station and
- provide a detailed design for a submersible robot.

4.1 OVERVIEW

In Chapters 1 and 2, we provided an overview of the Arduino Development Environment and hardware to get you quickly up and operating with this user friendly processor. Chapter 3 provided information on how to properly connect a wide variety of devices to the Arduino. In this chapter we provide several examples of Arduino-based systems.

4.2 WEATHER STATION

In this project we design a weather station with the following requirements:

- design a weather station to sense wind direction and speed, ambient temperature, and accumulated rainfall;
- wind direction should be displayed on LEDs arranged in a circular pattern; and
- collected weather data should be displayed on a two line LCD and transmitted serially and time stamped to a microSD card for storage.

4.2.1 STRUCTURE CHART

To begin, the design process, a structure chart is used to provide an overall system picture and partition the system into definable pieces. The structure chart for the weather station is provided in Figure 4.1. We employ a top-down design/bottom-up implementation approach. The system is partitioned until the lowest level of the structure chart contains "doable" pieces of hardware components or software functions. Data flow is shown on the structure chart as directed arrows.

The main microcontroller subsystems needed for this project are the UART (serial port) for the LCD display, the ADC system for wind direction and temperature sensing, the I2C system to communicate with the Real Time Clock (RTC) module, the Serial Peripheral Interface (SPI) for the SD card data logger, and the interrupt system for the wind speed and the rain gage.

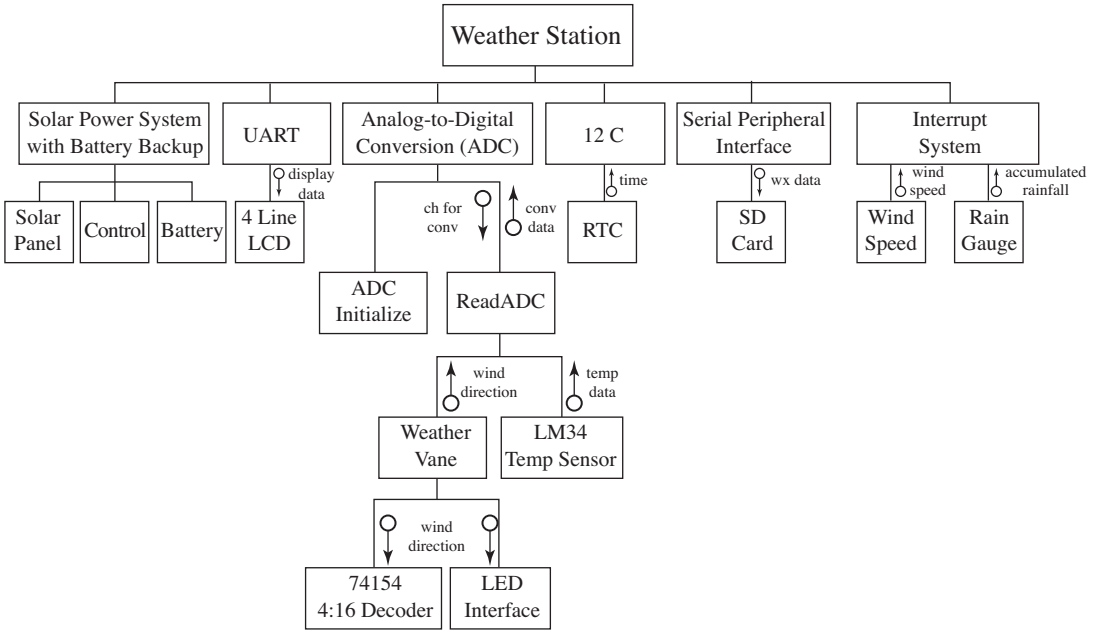


Figure 4.1: Weather station structure chart.

4.2.2 CIRCUIT DIAGRAM

The circuit diagram for the weather station and subsystems is provided in Figure 4.2. We discuss each subsystem in turn.

Liquid Crystal Display. We use the Sparkfun LCD-09535 serial enabled, 16 character, two line LCD to display weather data. To display four lines of weather data we toggle between two line displays. This LCD was discussed in Chapter 3.

Temperature Sensor. The weather station is equipped with an LM34 Precision Fahrenheit Temperature Sensor. The LM34 provides 10 mV output per degree Fahrenheit. The output from the sensor is provided to Arduino analog input pin A1 (SNIS161D [4]).

Weather Vane. The Sparkfun Weather Meters kit (# SEN-08942) is equipped with a weather vane, an anemometer (wind speed), and a rain gauge. The weather vane provides a voltage output from 0–5 VDC for different wind directions, as shown in Figure 4.2. The weather vane must be oriented to a known direction with the output voltage at this direction noted. The output voltage is provided to an RJ11 connector. Pins 1 and 4 of the RJ11 connector provide access to the vane's resistance. A 10-k ohm resistor is placed in series with the vane's resistance to provide a voltage divider circuit (`www.sparkfun.com`).

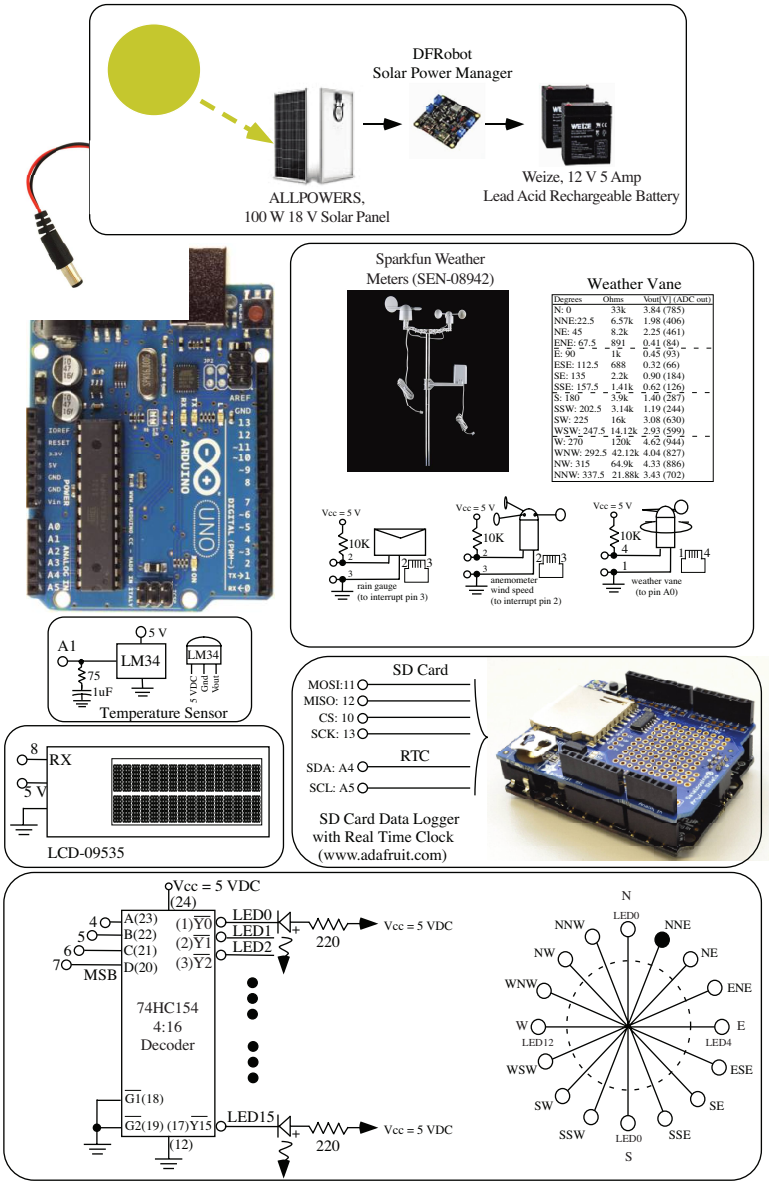


Figure 4.2: Circuit diagram for weather station. Illustrations used with permission of Sparkfun Electronics (www.sparkfun.com) and Adafruit (www.adafruit.com). UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA) (www.arduino.cc).

Rain Gauge. The rain gauge contains an enclosed rain tipping bucket. When the bucket is full, it tips, tripping a magnetic read switch. The switch closure is routed via an RJ11 connector to Arduino interrupt pin 3. Each tip represents 0.011 inch (0.2794 mm) of rainfall.

Anemometer-Wind Speed. The anemometer is equipped with three cups to catch the wind. With each rotation of the anemometer, a reed switch is tripped. Switch closures one second apart equates to 1.492 miles per hour (2.4 km/h) of wind speed. The switch closure is routed via an RJ11 connector to Arduino interrupt pin 2.

Wind Direction Display. The internal circuitry of the weather vane consists of 2 magnetic switches and 16 different resistor values to indicate wind direction. As the weather vane is turned by the wind, 1 of the 16 resistors is selected. The selected resistor is in series with a 10-k ohm resistor to provide a voltage divider circuit. The voltage across the selected resistor is provided to Arduino UNO R3 analog input pin A1. The sensed voltage is converted to a corresponding ADC output, as shown in Figure 4.2. The ADC output is converted to wind direction and displayed on the LCD.

The wind direction display also has 16 different LEDs to indicate wind direction. To conserve Arduino pins, a 74HC154 4:16 decoder is used. When a binary value is provided to inputs A-D of the 74HC154, the corresponding decoder output goes to logic low while the remaining outputs remain at logic high. The LED at the logic low pin illuminates. An interface circuit is required for each LED, as shown in Figure 4.2.

SD Card with Real Time Clock. We use the Adafruit Data Logger Shield (Adafruit #1141). The shield provides multiple gigabits of additional storage for the UNO R3. It features a Real Time Clock (RTC) with battery backup to timestamp collected data. RTC features provide the microcontroller the ability to track calendar time based on seconds, minutes, hours, etc. (`www.adafruit.com`).

Solar Power System. The solar power system consists of a solar panel, a solar power manager, a rechargeable battery, and fuses for circuit protection. In this project we use the DFRobot DFR0580 Solar Power Manager for a 12 VDC lead-acid battery. With an 18 VDC, 100 W solar panel and a 12 VDC lead-acid battery; the DFR0580 can provide regulated output voltages of 5 VDC at 5 amps and 12 VDC at 8 amps. This is more than required for the weather station; however, we expand upon this project in future volumes of this text (`www.DFRobot.com`).

4.2.3 BOTTOM-UP IMPLEMENTATION

A sound implementation approach is to build up a complex system a subsystem at a time. We use this approach to assemble the weather station. We start with the LCD and then develop small test programs to assemble and test each weather station subsystem.

Liquid Crystal Display. We use Sparkfun LCD-09395 to display temperature, wind speed, wind direction, and accumulated rainfall. The following sketch may be used to display these weather parameters.

```
//*****************************************************************
//Example uses the Arduino Software Serial Library
// - provides software-based serial port
//Sparkfun LCD-09395: 2 line, 16 character LCD,
// - toggles between weather readings every 2 seconds
//*****************************************************************

#include <SoftwareSerial.h>

//Arduino pins for Serial connection:
//  format: SoftwareSerial LCD(RX_pin, TX_pin);

SoftwareSerial LCD(8, 9);

void setup()
{
LCD.begin(9600);                    //Baud rate: 9600 Baud
delay(500);                         //Delay for display
}

void loop()
{
//Cursor to line one, character one
LCD.write(254);
LCD.write(128);

//clear display
LCD.write("                ");
LCD.write("                ");

//Cursor to line one, character one
LCD.write(254);
LCD.write(128);
LCD.write("Temp:");

//Cursor to line two, character one
LCD.write(254);
LCD.write(192);
LCD.write("Speed:");
```

```
delay(2000);                        //delay 2s

//Cursor to line one, character one
LCD.write(254);
LCD.write(128);

//clear display
LCD.write("                ");
LCD.write("                ");

//Cursor to line one, character one
LCD.write(254);
LCD.write(128);
LCD.write("Dir:");

//Cursor to line two, character one
LCD.write(254);
LCD.write(192);
LCD.write("Rain:");

delay(2000);                        //delay 2s

}

//*********************************************************************
```

Temperature Sensor. The weather station uses an LM34 Precision Fahrenheit Temperature Sensor manufactured by Texas Instruments. The LM34 provides 10 mV of output per degree Fahrenheit. The output (center) pin of the LM34 is provided to analog input pin A0 on the Arduino UNO R3. Provided below is test code to measure the output from the LM34, convert the LM34 output to temperature, and display the result on the LCD.

```
//*********************************************************************
//Example uses the Arduino Software Serial Library
// - provides software-based serial port
//*********************************************************************

#include <SoftwareSerial.h>

//Specify Arduino pins for Serial connection:
```

```
//        SoftwareSerial LCD(RX_pin, TX_pin);
SoftwareSerial LCD(8, 9);

//analog input pins
#define LM34_sensor     A0          //LM34 temp sensor at A0


int analog_temp;
int int_temp;
int troubleshoot = 1;               //1: serial monitor prints
char tempstring[10];                //create string array for LCD

void setup()
{
if (troubleshoot == 1) Serial.begin(9600);
LCD.begin(9600);                    //Baud rate: 9600 Baud
delay(500);                         //Delay for display
analogReference(DEFAULT);           //Reference voltage for ADC
}

void loop()
{                                   //Read temp from LM34 at A0
analog_temp = analogRead(LM34_sensor);
if (troubleshoot == 1) Serial.println (analog_temp);
                                    //LM34 10mV/degree
int_temp = (int)(((analog_temp/1023.0) * 5.0)/.010);
if (troubleshoot == 1) Serial.println (int_temp);
sprintf(tempstring, "

//Cursor to line one, character one
LCD.write(254);
LCD.write(128);

//clear display
LCD.write("                ");
LCD.write("                ");

//Cursor to line one, character one
LCD.write(254);
```

```
LCD.write(128);

LCD.write("Temp [F]:");

//Cursor to line one, character ten
LCD.write(254);
LCD.write(137);
LCD.write(tempstring);   //write temp string

delay(2000);
}

//*********************************************************************
```

Wind Direction Display. Provided below is an Arduino sketch to measure the output voltage from the weather vane, convert the voltage to a corresponding wind direction, and display the result to the LCD and the appropriate LED on the wind direction display described earlier.

```
//*********************************************************************
//Example uses the Arduino Software Serial Library
// - provides software-based serial port
//*********************************************************************

#include <SoftwareSerial.h>

//Specify Arduino pins for Serial connection:
//  SoftwareSerial LCD(RX_pin, TX_pin);

SoftwareSerial LCD(8, 9);

//analog input pins
#define wind_vane_sensor     A1     //wind vane sensor
#define p74HC154A             4     //LSB input to 74HC154 (1)
#define p74HC154B             5     //input to 74HC154     (2)
#define p74HC154C             6     //input to 74HC154     (4)
#define p74HC154D             7     //MSB input to 74HC154 (8)
#define LOW                   0
#define HIGH                  1

int analog_dir;
int int_dir;
```

```
int troubleshoot = 1;                //1: serial monitor prints
char dirstring[4];                   //create string array for LCD

void setup()
{
if (troubleshoot == 1) Serial.begin(9600);

LCD.begin(9600);                      //Baud rate: 9600 Baud
delay(500);                           //Delay for display
analogReference(DEFAULT);             //Reference voltage for ADC
pinMode(p74HC154A, OUTPUT);           //LSB input to 74HC154 (1)
pinMode(p74HC154B, OUTPUT);           //input to 74HC154     (2)
pinMode(p74HC154C, OUTPUT);           //input to 74HC154     (4)
pinMode(p74HC154D, OUTPUT);           //MSB input to 74HC154 (8)
}

void loop()
{                                     //Read dir from vane at A1
analog_dir = analogRead(wind_vane_sensor);
if (troubleshoot == 1) Serial.println (analog_dir);

//Cursor to line two, character one
LCD.write(254);
LCD.write(192);

//clear display
LCD.write("                ");
LCD.write("                ");

//Cursor to line two, character one
LCD.write(254);
LCD.write(192);

LCD.write("Dir:");

//Cursor to line two, character ten
LCD.write(254);
LCD.write(202);
```

```
//Determine wind direction - vane 0 degree aligned with North
//North - ADC output 785
if((analog_dir >= 744) && (analog_dir <= 806))
  {
  LCD.write("N  ");
  digitalWrite(p74HC154D, 0); digitalWrite(p74HC154C, 0);
  digitalWrite(p74HC154B, 0); digitalWrite(p74HC154A, 0);
  }
//NNE - ADC output 406
else if((analog_dir >= 347) && (analog_dir <= 433))
  {
  LCD.write("NNE");
  digitalWrite(p74HC154D, 0); digitalWrite(p74HC154C, 0);
  digitalWrite(p74HC154B, 0); digitalWrite(p74HC154A, 1);
  }
//NE - ADC output 461
else if((analog_dir >= 434) && (analog_dir <= 530))
  {
  LCD.write("NE ");
  digitalWrite(p74HC154D, 0); digitalWrite(p74HC154C, 0);
  digitalWrite(p74HC154B, 1); digitalWrite(p74HC154A, 0);
  }
//ENE - ADC output 84
else if((analog_dir >= 76) && (analog_dir <= 88))
  {
  LCD.write("ENE");
  digitalWrite(p74HC154D, 0); digitalWrite(p74HC154C, 0);
  digitalWrite(p74HC154B, 1); digitalWrite(p74HC154A, 1);
  }
//E - ADC output 93
else if((analog_dir >= 89) && (analog_dir <= 109))
  {
  LCD.write("E  ");
  digitalWrite(p74HC154D, 0); digitalWrite(p74HC154C, 1);
  digitalWrite(p74HC154B, 0); digitalWrite(p74HC154A, 0);
  }
//ESE - ADC output 66
else if((analog_dir >= 62) && (analog_dir <= 68))
  {
```

```
  LCD.write("ESE");
  digitalWrite(p74HC154D, 0); digitalWrite(p74HC154C, 1);
  digitalWrite(p74HC154B, 0); digitalWrite(p74HC154A, 1);
  }
//SE - ADC output 184
else if((analog_dir >= 181) && (analog_dir <= 187))
  {
  LCD.write("SE ");
  digitalWrite(p74HC154D, 0); digitalWrite(p74HC154C, 1);
  digitalWrite(p74HC154B, 1); digitalWrite(p74HC154A, 0);
  }
//SSE - ADC output 126
else if((analog_dir >= 110) && (analog_dir <= 155))
  {
  LCD.write("SSE");
  digitalWrite(p74HC154D, 0); digitalWrite(p74HC154C, 1);
  digitalWrite(p74HC154B, 1); digitalWrite(p74HC154A, 1);
  }
//S - ADC output 287
else if((analog_dir >= 266) && (analog_dir <= 346))
  {
  LCD.write("S  ");
  digitalWrite(p74HC154D, 1); digitalWrite(p74HC154C, 0);
  digitalWrite(p74HC154B, 0); digitalWrite(p74HC154A, 0);
  }
//SSW - ADC output 244
else if((analog_dir >= 214) && (analog_dir <= 265))
  {
  LCD.write("SSW");
  digitalWrite(p74HC154D, 1); digitalWrite(p74HC154C, 0);
  digitalWrite(p74HC154B, 0); digitalWrite(p74HC154A, 1);
  }
//SW - ADC output 630
else if((analog_dir >= 615) && (analog_dir <= 665))
  {
  LCD.write("SW ");
  digitalWrite(p74HC154D, 1); digitalWrite(p74HC154C, 0);
  digitalWrite(p74HC154B, 1); digitalWrite(p74HC154A, 0);
  }
```

```
//WSW - ADC output 599
else if((analog_dir >= 531) && (analog_dir <= 614))
  {
  LCD.write("WSW");
  digitalWrite(p74HC154D, 1); digitalWrite(p74HC154C, 0);
  digitalWrite(p74HC154B, 1); digitalWrite(p74HC154A, 1);
  }
//W - ADC output 944
else if((analog_dir >= 916) && (analog_dir <= 1023))
  {
  LCD.write("W  ");
  digitalWrite(p74HC154D, 1); digitalWrite(p74HC154C, 1);
  digitalWrite(p74HC154B, 0); digitalWrite(p74HC154A, 0);
  }
//WNW - ADC output 827
else if((analog_dir >= 807) && (analog_dir <= 856))
  {
  LCD.write("WNW");
  digitalWrite(p74HC154D, 1); digitalWrite(p74HC154C, 1);
  digitalWrite(p74HC154B, 0); digitalWrite(p74HC154A, 1);
  }
//NW - ADC output 886
else if((analog_dir >= 857) && (analog_dir <= 915))
  {
  LCD.write("NW ");
  digitalWrite(p74HC154D, 1); digitalWrite(p74HC154C, 1);
  digitalWrite(p74HC154B, 1); digitalWrite(p74HC154A, 0);
  }
//NNW - ADC output 702
else if((analog_dir >= 667) && (analog_dir <= 743))
  {
  LCD.write("NNW");
  digitalWrite(p74HC154D, 1); digitalWrite(p74HC154C, 1);
  digitalWrite(p74HC154B, 1); digitalWrite(p74HC154A, 1);
  }
else
  {
  LCD.write("<-->");
  }
```

```
delay(100);
}

//*****************************************************************
```

Rain Gauge. The rain gauge portion of the Sparkfun Weather Meters (SEN-08942) provides a switch closure for each tip of the rain bucket. When the bucket is full, it tips, tripping a magnetic read switch. The switch closure is routed via an RJ11 connector to Arduino interrupt pin 3. Each tip represents 0.011 inch (0.2794 mm) of rainfall. In the Arduino sketch provided below, the interrupt increments a switch closure counter and converts the result to accumulated rainfall.

```
//*****************************************************************
//Program measures the accumulated rainfall since the last reset.
//The rain gauge switch is in series with a 10K resistor pulled up
//to 5 VDC.  The switch is provided to INT1 (pin 3) of the UNO R3.
//*****************************************************************

unsigned long rain_sw_closures = 0;
float rainfall;

void setup()
{
Serial.begin(9600);
pinMode(3, INPUT);
attachInterrupt(1, int1_ISR, FALLING);
}

void loop()
{

//wait for interrupts

}

//*****************************************************************
//int_ISR: interrupt service routine for INT1
//*****************************************************************

void int1_ISR(void)
{
```

```
rain_sw_closures++;                          //increment rainfall count
                                             //switch closures to inches
rainfall = ((float)(rain_sw_closures) * 0.011);
Serial.print(rainfall);
Serial.println("   inches");
Serial.println();
}

//****************************************************************
```

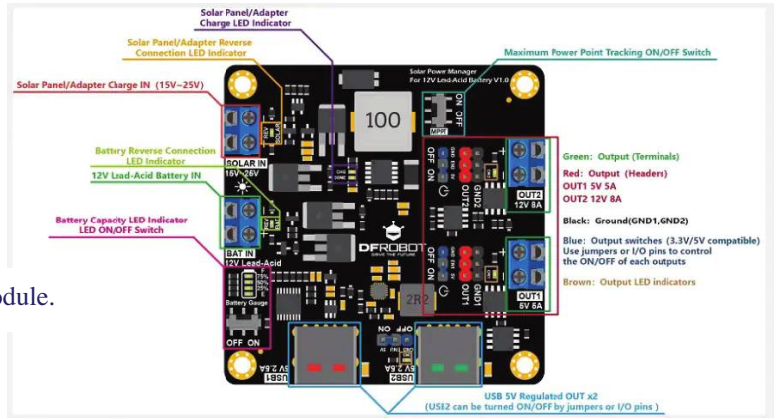
Anemometer-Wind Speed. The anemometer portion of the Sparkfun Weather Meters (SEN-08942) is equipped with a three cups to catch the wind. With each rotation of the anemometer, a reed switch is tripped. Switch closures one second apart equates to 1.492 miles per hour (2.4 km/h) of wind speed. The switch closure is routed via an RJ11 connector to Arduino interrupt pin 2. In the Arduino sketch provided below, the time between two switch closures is measured and converted to wind speed in MPH.

```
//****************************************************************
//Program measures the elapsed time in ms between two switch
//closures of the anemometer.  The anemometer switch is in series
//with a 10K resistor pulled up to 5 VDC.  The switch is provided
//to INT0 (pin 2) of the UNO R3.
//****************************************************************

unsigned long first, second, elapsed_time;        //milliseconds
unsigned int first_time_hack = 1, wind_speed;

void setup()
{
Serial.begin(9600);
pinMode(2, INPUT);
attachInterrupt(0, int0_ISR, FALLING);
}

void loop()
{

//wait for interrupts

}
```

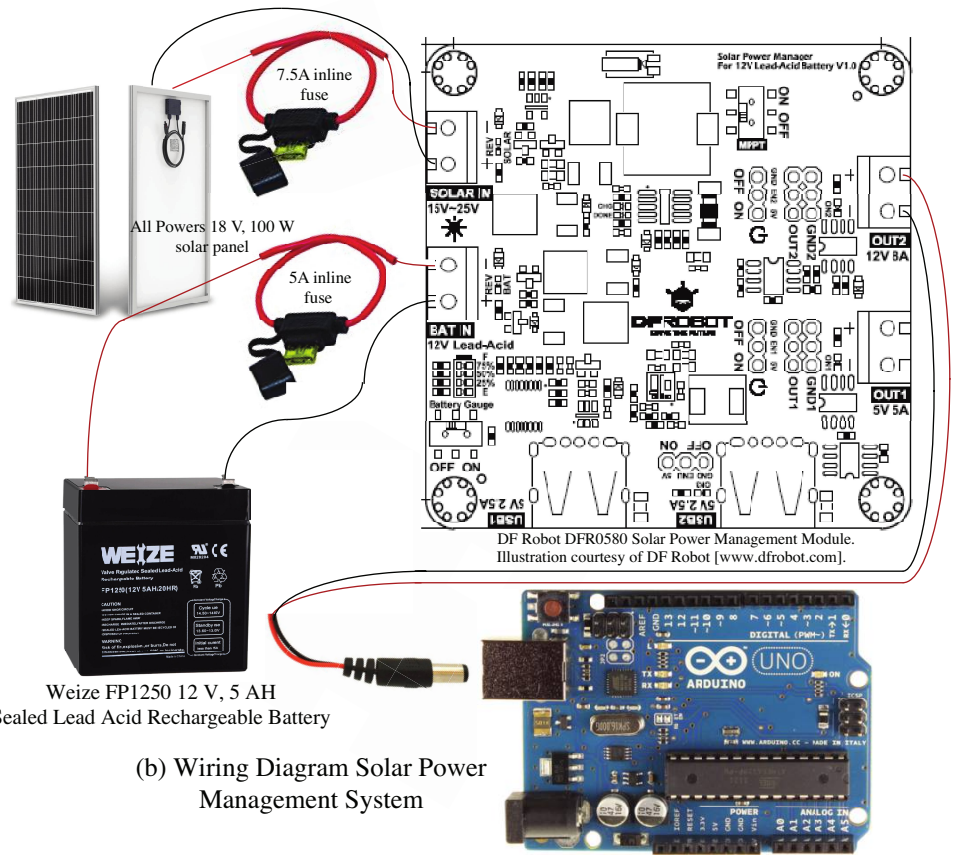
```
//*************************************************************
//int0_ISR: interrupt service routine for INT0
//*************************************************************

void int0_ISR(void)
{
if(first_time_hack ==1)
  {
  first = millis();                                  //milliseconds
  first_time_hack = 0;
  }
else
  {
  second = millis();                                 //milliseconds
  first_time_hack = 1;
  elapsed_time = second-first;                       //milliseconds
                                                     //ms to MPH
  wind_speed = (unsigned int)((1000.0/(float)(elapsed_time)) * 1.492);
  Serial.print(wind_speed);
  Serial.println(" MPH");
  Serial.println();
  }
}

//*************************************************************
```

SD Card with Real Time Clock. We use the Adafruit data logger shield (Adafruit #1141). It is equipped with an RTC and battery backup to timestamp collected data. Adafruit provides step-by-step instructions to connect header pins and program the SD Card shield in "Adafruit Data Logger Shield (www.adafruit.com)."

Solar Power System. The solar power system for the remote weather station is shown in Figure 4.3. The system is managed by the DFRobot DFR0590 Solar Power Management Module. This module was designed for use with an 18 V, 100 W solar panel and provides trickle charging for a 12 V lead-acid battery. The solar panel chosen for the system is an All Powers 18 V, 100 W solar panel (#AP-SP-016-SIL). It has a working voltage of 18 V and working current of 5.8 A. The solar panel is connected to the DFR0590 Solar Power Management Module via 10 AWG insulated power cables equipped with MC4 connectors. The battery chosen for the system is the Weize FP1250 12 V, 5AH sealed lead-acid rechargeable battery. The power cable from the solar panel to the power management module is fused at 7.5 A. The power cable between the power management module and the battery is fused at 5 A. The solar power system



(a) DF Robot DFR0580 Solar Power Management Module. Illustration courtesy of DF Robot (www.dfrobot.com).



(b) Wiring Diagram Solar Power Management System

Figure 4.3: Solar power system for the remote weather station. Illustrations courtesy of DFRobot (www.DFRobot.com), All Powers (www.allpowers.com), and Weize (www.weize.com).

is more than adequate to meet the needs of the remote weather station. We add to the project in future volumes of the textbook series. Due to the wide range of applicable codes and standards and permitting requirements for photovoltaic systems, the solar panel and associated equipment should be installed by a licensed electrician.

4.2.4 UML ACTIVITY DIAGRAM

The UML activity diagram for the main program is provided in Figure 4.4. After initializing the subsystems, the program enters a continuous loop where temperature and wind direction is sensed and displayed on the LCD and the LED display. Interrupts are used to capture data on wind speed and rainfall. The sensed values are then transmitted via the SPI to the MMC/SD card. The system then enters a delay to set how often the temperature and wind direction parameters are updated. We have you construct the individual UML activity diagrams for each function as an end of chapter exercise.

4.2.5 MICROCONTROLLER CODE

We leave the final weather station code as a homework assignment at the end of the chapter. The final code may be assembled from all of the individual code sketches provided in this section.

4.2.6 FINAL ASSEMBLY

The weather station may be assembled into a small package by using an Adafruit Wing Shield (#196). The Wing Shield couples to the Arduino UNO R3 with onboard stacking headers. The Adafruit data logger shield (#1141) is then stacked upon the Wing Shield. The Wing Shield is equipped with screw terminals to connect the weather station peripherals to the stacked modules, as shown in Figure 4.5.

4.3 SUBMERSIBLE ROBOT

The area of submersible robots is fascinating and challenging. To design a robot is quite complex (yet fun). To add the additional requirement of waterproofing key components provides an additional level of challenge. (Water and electricity do not mix!) In this section we provide the construction details and a control system for a remotely operated vehicle, an ROV. Specifically, we develop the structure and control system for the SeaPerch style ROV. By definition, an ROV is equipped with a tether umbilical cable that provides power and control signals from a surface support platform. An Autonomous Underwater Vehicle (AUV) carries its own power and control equipment and does not require surface support (SeaPerch [3]).

Details on the construction and waterproofing of an ROV are provided in the excellent and fascinating *Build Your Own Underwater Robot and Other Wet Projects* by Harry Bohm and Vickie Jensen. For an advanced treatment, please see *The ROV Manual—A User Guide for Remotely Operated Vehicles* by Robert Crist and Robert Wernli, Sr. There is a national—level competition

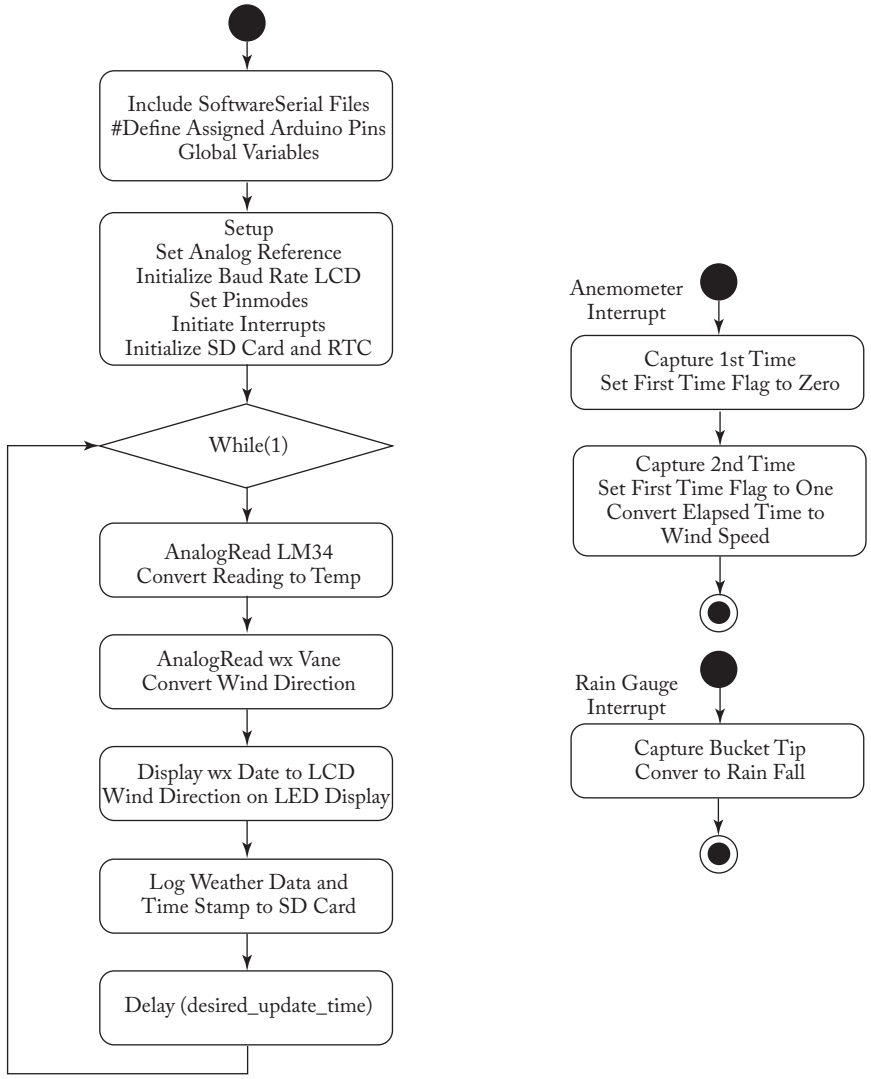


Figure 4.4: Weather station UML activity diagram.

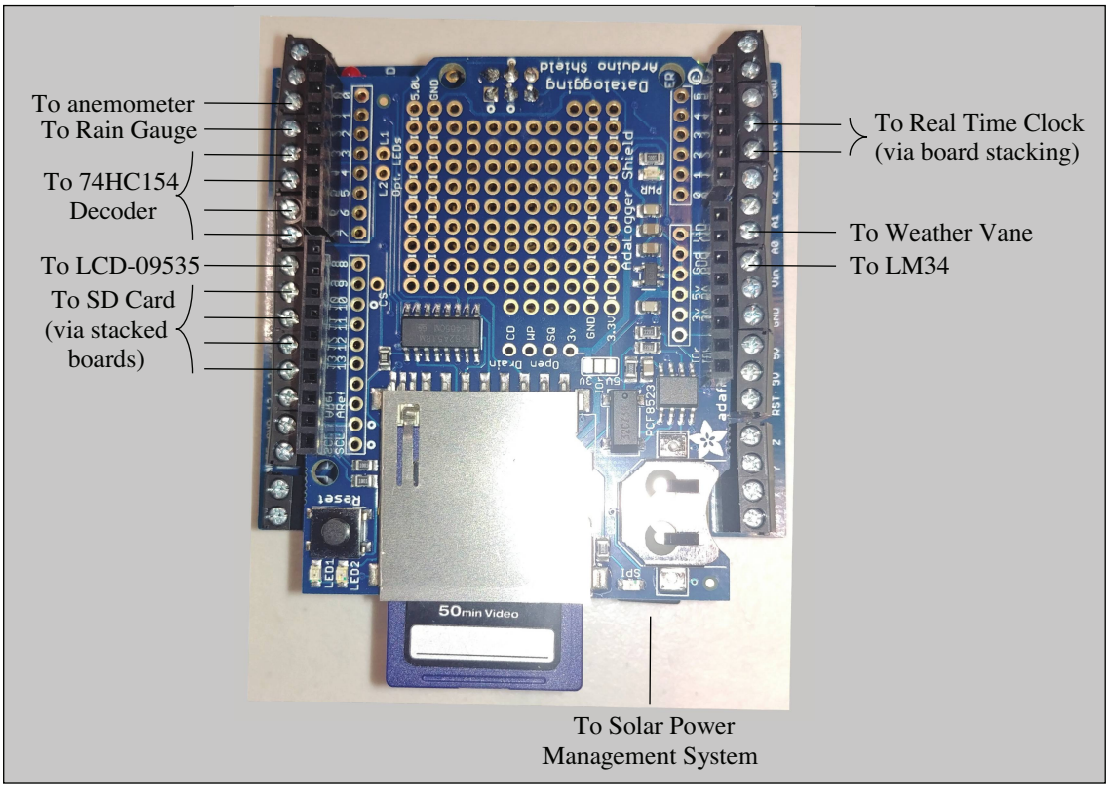


Figure 4.5: Assembled weather station.

for students based on the SeaPerch ROV. The goal of the program is to stimulate interest in the next generation of marine related engineering specialties (SeaPerch [3]).

4.3.1 APPROACH

This is a challenging project; however, we take a methodical, step-by-step approach to successfully design and construct the ROV. We complete the design tasks in the following order:

- determine requirements;
- design and construct ROV structure;
- design and fabricate control electronics;
- design and implement control software using the Arduino Development Environment;
- construct and assemble a prototype; and
- test the prototype.

4.3.2 REQUIREMENTS

The requirements for the ROV system include:

- develop a control system to allow a three-thruster (motor or bilge pump) ROV to move forward, left (port), and right (starboard);
- the ROV will be pushed down to a shallow depth via a vertical thruster and return to surface based on its own, slightly positive buoyancy;
- ROV movement will be under joystick control;
- LEDs are used to indicate thruster assertion;
- all power and control circuitry will be maintained in a surface support platform, as shown in Figure 4.6; and
- an umbilical cable connects the support platform to the ROV.

4.3.3 ROV STRUCTURE

The ROV structure is shown in Figure 4.7. The structure is constructed with 0.75 inch PVC piping. The structure is assembled quickly using "T" and corner connectors. The pieces are connected using PVC glue or machine screws. The PVC pipe and connectors are readily available in hardware and home improvement stores.

The fore or bow portion of the structure is equipped with plexiglass panels to serve as mounting bulkheads for the thrusters. The panels are mounted to the PVC structure using ring clamps. Either waterproofed electric motors or submersible bilge pumps are used as thrusters. A bilge pump is a pump specifically designed to remove water from the inside of a boat. The pumps are powered from a 12 VDC source and have typical flow rates from 360 to over 3,500 gallons per minute. They range in price from U.S. $20–$80 (`www.shorelinemarinedevelopment.com`). Details on waterproofing electric motors are provided in *Build Your Own Underwater Robot and Other Wet Projects*. We use three Shoreline Bilge Pumps rated at 600 gallons per minute (GPM). They are available online from `www.walmart.com`.

The aft or stern portion of the structure is designed to hold the flexible umbilical cable. The cable provides a link between the Arduino UNO R3 based control system and the thrusters. Each thruster may require up to 1–2 amps of current. Therefore, a 4-conductor, 16 AWG, braided (for flexibility) conductor cable is recommended. The cable is interfaced to the bilge pump leads using soldered connections or Butt connectors. The interface should be thoroughly waterproofed using caulk. For this design the interface was placed within a section of PVC pipe equipped with end caps. The resulting container is filled with waterproof caulk.

Once the ROV structure is complete its buoyancy is tested. This is accomplished by placing the ROV structure in water. The goal is to achieve a slightly positive buoyancy. With positive

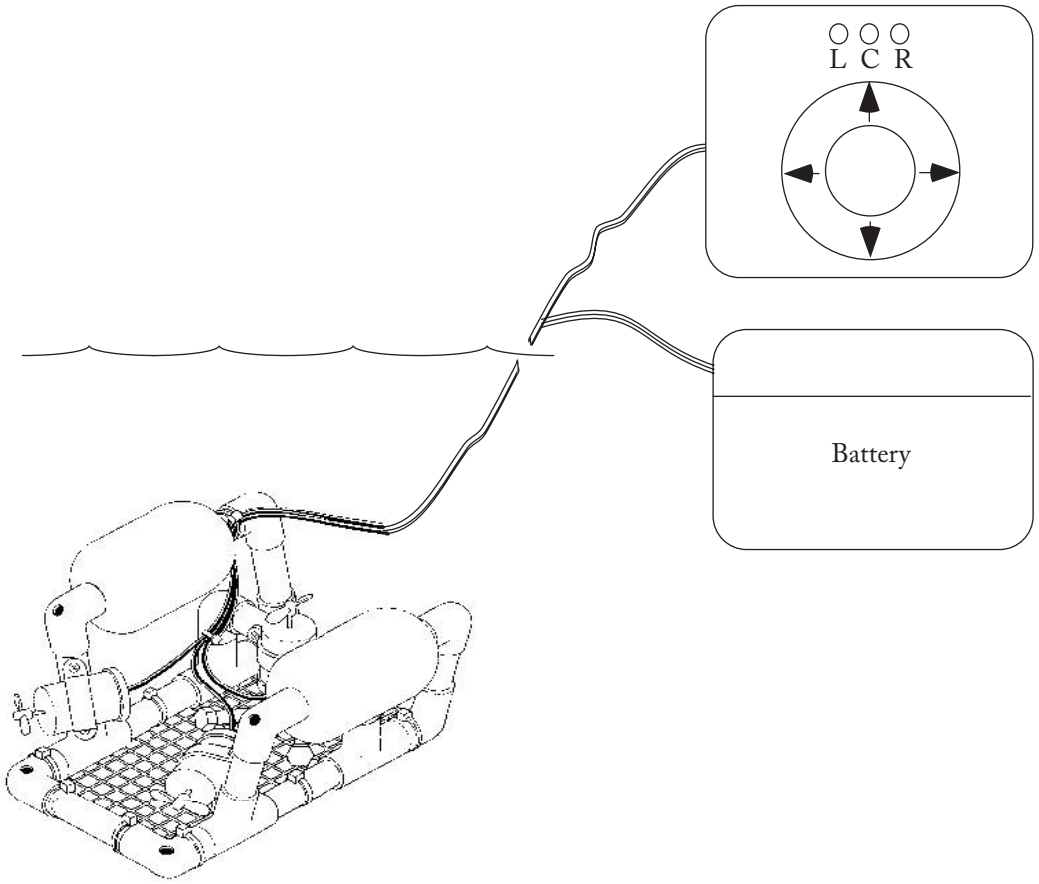


Figure 4.6: Power and control are provided remotely to the SeaPerch ROV. (Adapted and used with permission of Bohm and Jensen, West Coast Words Publishing.)

buoyancy the structure floats. With neutral buoyancy the structures hovers beneath the surface. With negative buoyancy the structure sinks. A more positive buoyancy way be achieved by attaching floats or foam to the structure tubing. A more negative buoyancy may be achieved by adding weights to the structure [Bohm and Jensen].

4.3.4 STRUCTURE CHART

The SeaPerch structure chart is provided in Figure 4.8. The SeaPerch control system will accept input from the five position joystick (left, right, select, up and down). We use the Sparkfun thumb joystick (Sparkfun COM-09032). The joystick schematic and connections to the Arduino UNO R3 are provided in Figures 4.9 and 4.10.

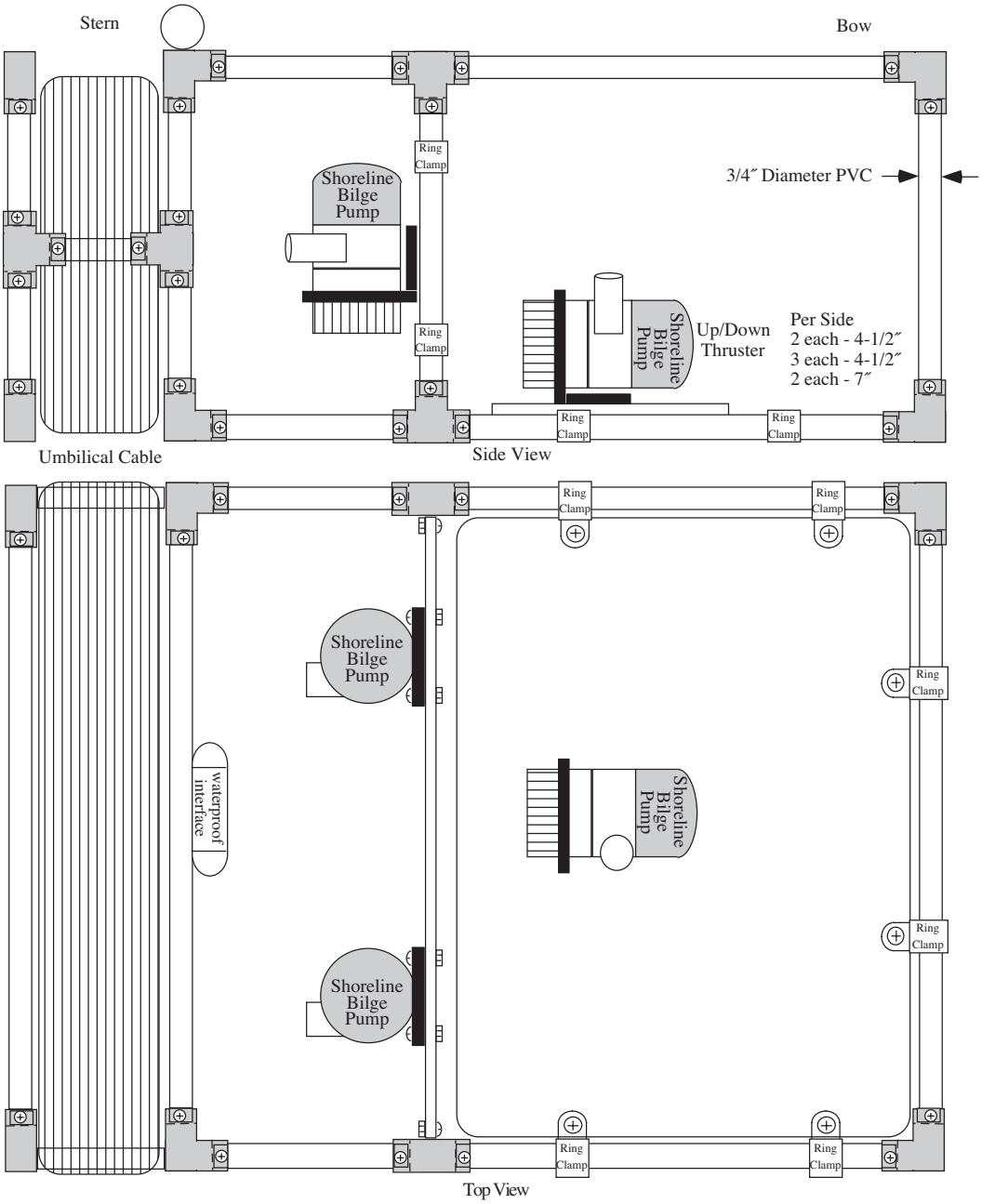


Figure 4.7: SeaPerch ROV structure.

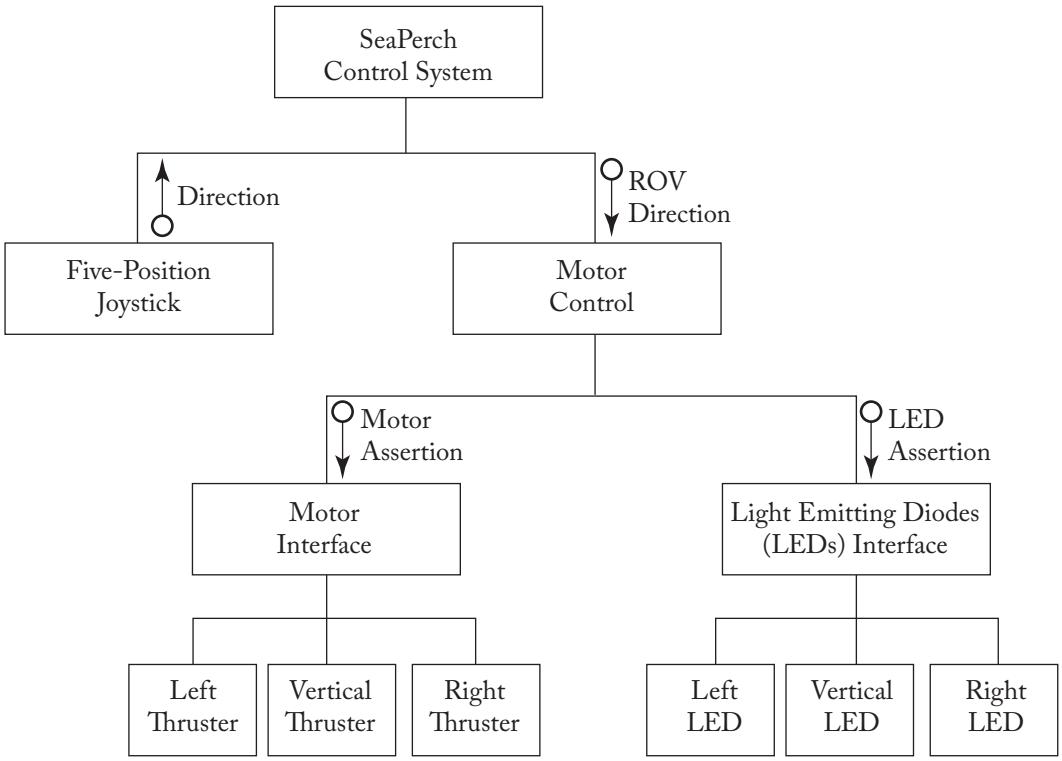


Figure 4.8: SeaPerch ROV structure chart.

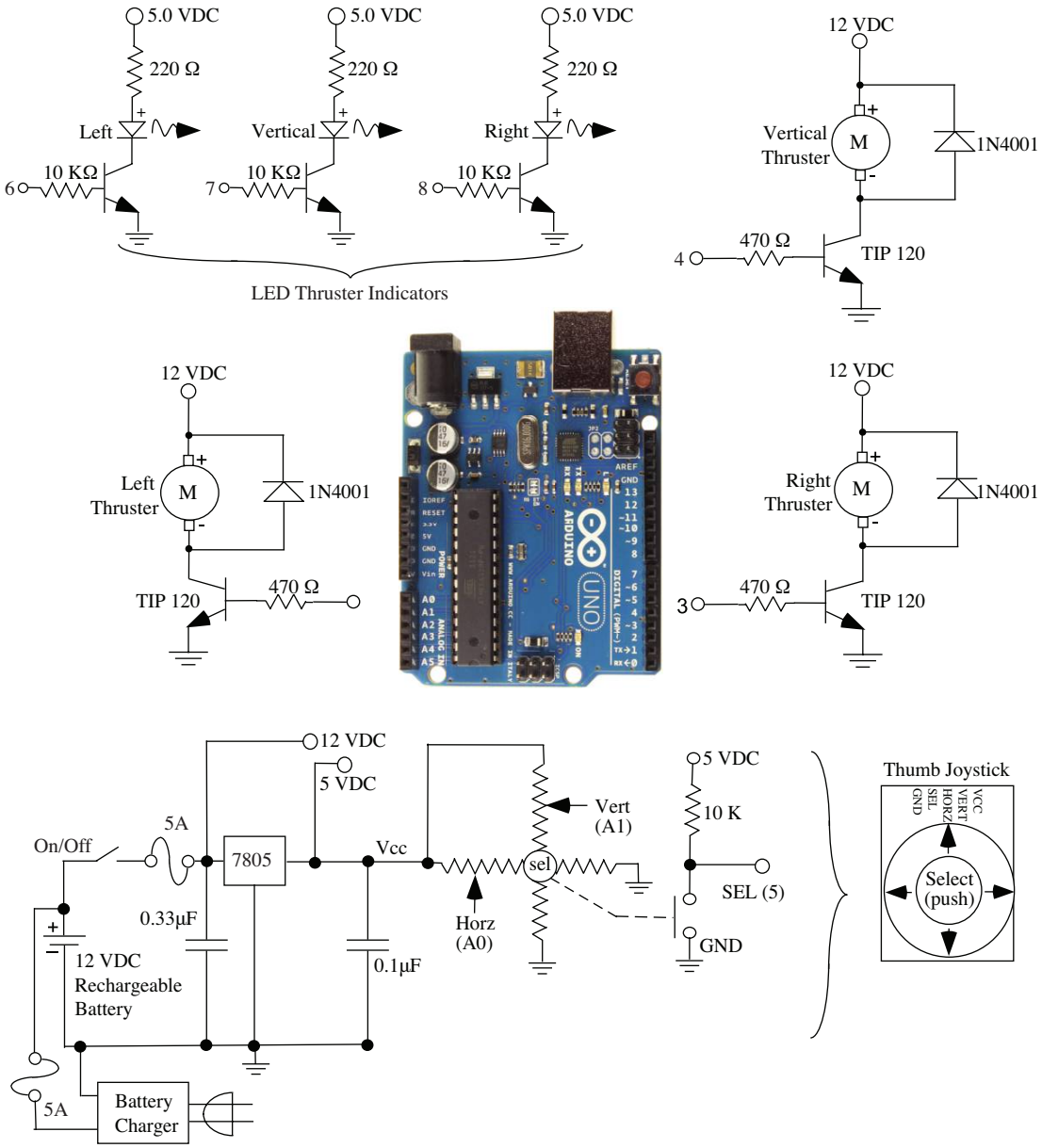


Figure 4.9: SeaPerch ROV interface control. UNO R3 illustration used with permission of the Arduino Team (CC BY-NC-SA; www.arduino.cc).

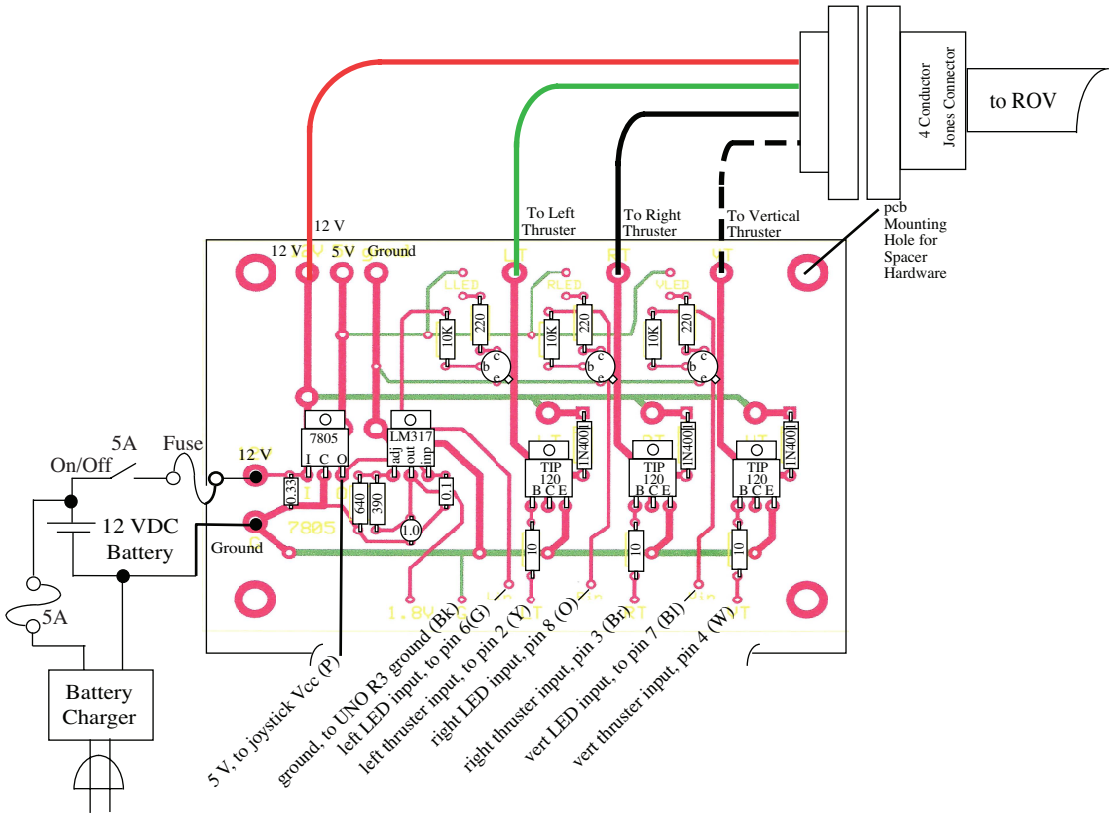


Figure 4.10: SeaPerch ROV printed circuit board interface.

In response to user joystick input, the SeaPerch control algorithm will issue a control command indicating desired ROV direction. In response to this desired direction command, the motor control algorithm will issue control signals to assert the appropriate thrusters and LEDs.

4.3.5 CIRCUIT DIAGRAM

The circuit diagram for the SeaPerch control system is provided in Figure 4.9. The thumb joystick is used to select desired ROV direction. The thumb joystick contains two built-in potentiometers (horizontal and vertical). A reference voltage of 5 VDC is applied to the V_{CC} input of the joystick. As the joystick is moved, the horizontal (HORZ) and vertical (VERT) analog output voltages will change to indicate the joystick position. The joystick is also equipped with a digital select (SEL) button. The SEL button is used to activate an ROV dive. The joystick is interfaced to Arduino UNO R3, as shown in Figure 4.9.

There are three LED interface circuits connected to the Arduino UNO R3 header pins 6, 7, and 8. The LEDs illuminate to indicate the left, vertical, and right thrusters have been asserted. As previously mentioned, the prime mover for the ROV are three bilge pumps. The left and right bilge pumps are driven by pulse width modulation channels (pins 2 and 3) via power NPN Darlington transistors (TIP 120), as shown in Figure 4.9. The vertical thrust is under digital pin control 4 equipped with NPN Darlington transistor (TIP 120) interface.

The interface circuitry between the Arduino UNO R3 and the bilge pumps is mounted on a printed circuit board (PCB) within the control housing. The interface between Arduino, the PCB, and the umbilical cable is provided in Figure 4.10.

4.3.6 UML ACTIVITY DIAGRAM

The SeaPerch control system UML activity diagram is provided in Figure 4.11. After initializing the UNO R3 pins the control algorithm is placed in a continuous loop awaiting user input. In response to user input, the algorithm determines desired direction of ROV travel and asserts appropriate control signals for the LED and motors.

4.3.7 ARDUINO UNO R3 SKETCH

In this example we use the thumb joystick to control the left and right thruster (motor or bilge pump). The joystick provides a separate voltage from 0–5 VDC for the horizontal (HORZ) and vertical (VERT) position of the joystick. We use this voltage to set the duty cycle of the pulse width modulated (PWM) signals sent to the left and right thrusters. The select push-button (SEL) on the joystick is used to assert the vertical thruster. The analog read function (analogRead) is used to read the X and Y position of the joystick. A value from 0–1023 is reported from the analog read function corresponding to 0–5 VDC. After the voltage readings are taken they are scaled to 5 VDC for further processing. Joystick activity is divided into multiple zones (0–8), as shown in Figure 4.12. The joystick signal is further processed consistent with the joystick zone selected.

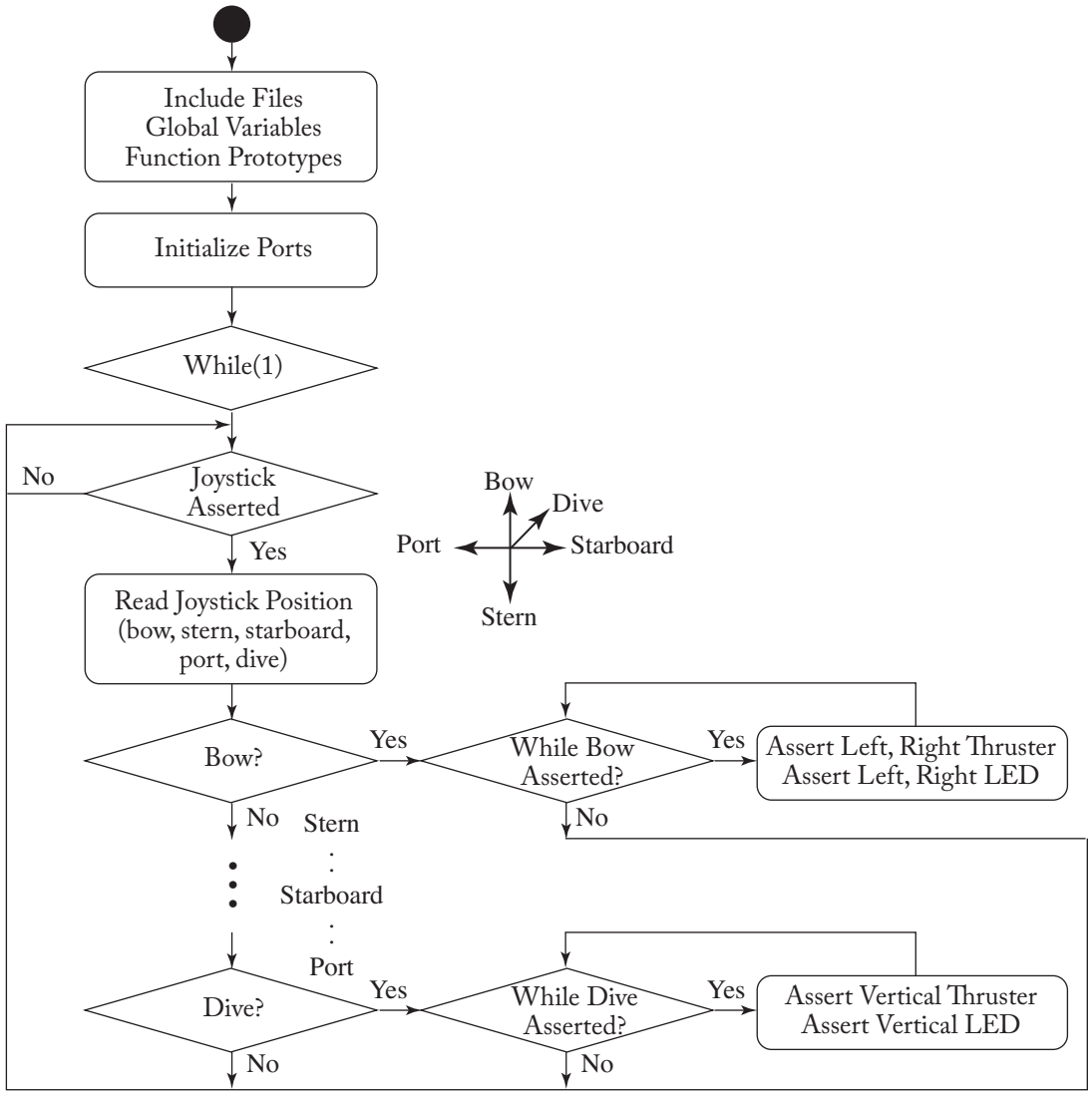


Figure 4.11: SeaPerch ROV UML activity diagram.

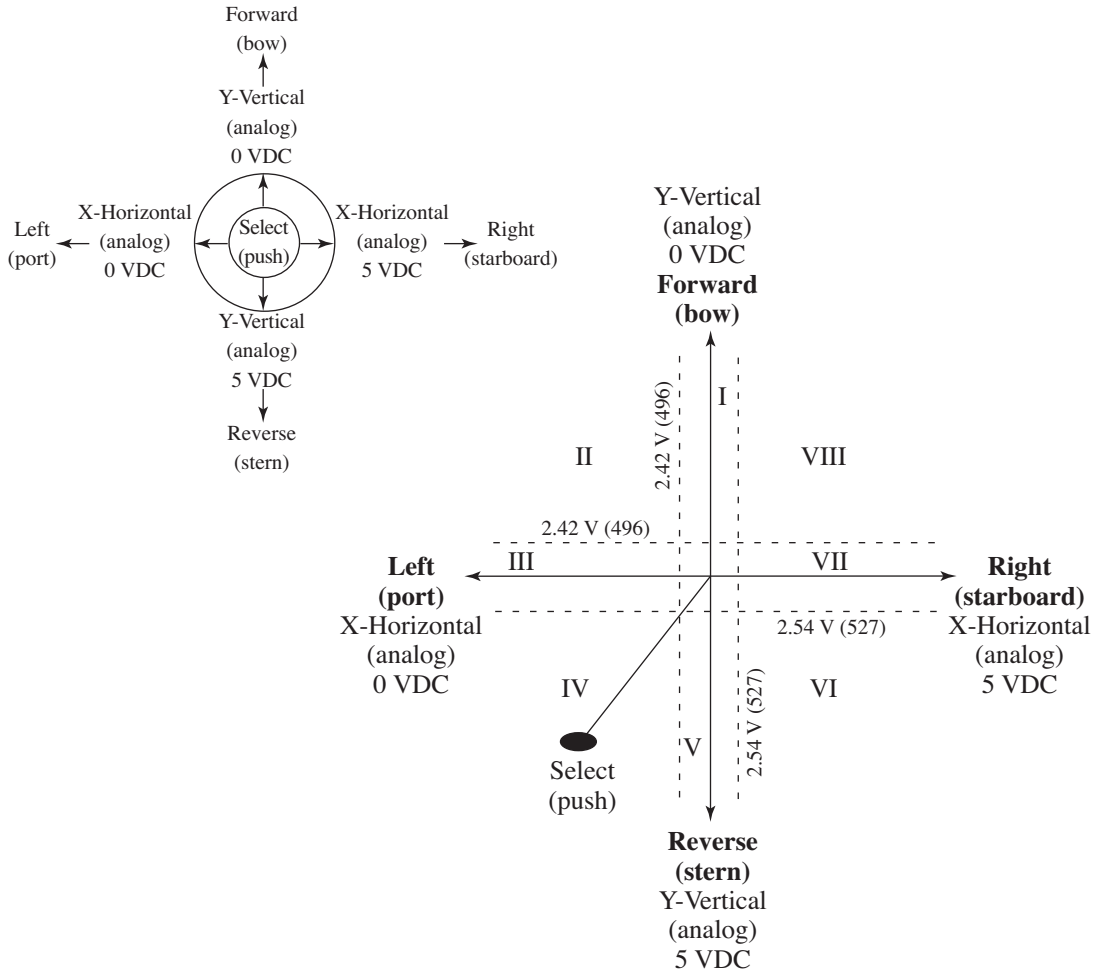


Figure 4.12: Joystick position as related to thruster activity.

```
//*********************************************************************
//ROV
//In response to joystick input, the SeaPerch control algorithm issues
//a control command indicating desired ROV direction.  In response to
//desired direction command, the motor control algorithm issues
//control signals to assert the appropriate thrusters and LEDs.
//*********************************************************************

//analog input pins
```

```
#define joystick_hor      A0      //analog pin - joystick horizontal in
#define joystick_ver      A1      //analog pin - joystick vertical in

//digital input pin
#define joystick_sel       5      //digital pin - joystick select in

//digital output pins - LED indicators
#define left_LED           6      //digital pin - left LED out
#define vertical_LED       7      //digital pin - vertical LED out
#define right_LED          8      //digital pin - right LED out

//thruster outputs
#define left_thruster      2      //digital pin - left thruster
#define right_thruster     3      //digital pin - right thruster
#define vertical_thruster  4      //digital pin - vertical thruster

int joystick_hor_value;           //horizontal joystick value
int joystick_ver_value;           //vertical joystick value
int joystick_sel_value;           //joystick select value
int joystick_thrust_on;           //1: thrust on; 0: off
int troubleshoot = 1;             //1: serial monitor prints

void setup()
{
//LED indicators
pinMode(left_LED,     OUTPUT);    //config pin for digital out -
//                                                  left LED
pinMode(vertical_LED, OUTPUT);    //config pin for digital out -
//                                                  vertical LED
pinMode(right_LED,    OUTPUT);    //config pin for digital out -
//                                                  right LED

//joystick select input
pinMode(joystick_sel, INPUT);     //config pin for digital in -
//                                                  joystick sel

//thruster outputs
pinMode(left_thruster,     OUTPUT);  //config digital out -
//                                                  left thruster
```

```
pinMode(vertical_thruster, OUTPUT);  //config digital out -
//                                                    vertical thruster
pinMode(right_thruster,    OUTPUT);  //config digital out -
//                                                    right thruster

//Serial monitor - open serial communications
if(troubleshoot == 1) Serial.begin(9600);
}

void loop()
{
//set update interval
delay(1000);

//turn off LEDs
digitalWrite(left_LED,     LOW);        //left LED     - off
digitalWrite(vertical_LED, LOW);        //vertical LED - off
digitalWrite(right_LED,    LOW);       //right LED    - off

//read hor and vert joystick position
//analog read returns value between 0 and 1023
joystick_hor_value = analogRead(joystick_hor);
joystick_ver_value = analogRead(joystick_ver);

if(troubleshoot == 1) Serial.println(joystick_hor_value);
if(troubleshoot == 1) Serial.println(joystick_ver_value);

//Convert 0 to 1023 to 0 to 5 VDC value
joystick_hor_value = ((joystick_hor_value/1023.0) * 5.0);
if(troubleshoot == 1) Serial.println(joystick_hor_value);

joystick_ver_value = ((joystick_ver_value/1023.0) * 5.0);
if(troubleshoot == 1) Serial.println(joystick_ver_value);

//Read vertical thrust
joystick_thrust_on  = digitalRead(joystick_sel); //vertical thrust?

//*************************************************************
//vertical thrust - active low pushbutton on joystick
```

```
//*************************************************************
if(joystick_thrust_on == 0)
  {
  digitalWrite(vertical_thruster, HIGH);
  digitalWrite(vertical_LED,  HIGH);
  if(troubleshoot == 1) Serial.println("Thrust is on!");
  }
else
  {
  digitalWrite(vertical_thruster, LOW);
  digitalWrite(vertical_LED,  LOW);
  if(troubleshoot == 1) Serial.println("Thrust is off!");
  }

//*************************************************************
//*************************************************************
//process different joystick zones
//*************************************************************
//Case 0: Joystick in null position
//Inputs:
//  X channel between 2.42 to 2.54 VDC - null zone
//  Y channel between 2.42 to 2.54 VDC - null zone
//Output:
//  Shut off thrusters
//*************************************************************

if((joystick_hor_value > 2.42)&&(joystick_hor_value < 2.54)&&
   (joystick_ver_value > 2.42)&&(joystick_ver_value < 2.54))

  {
  if(troubleshoot == 1) Serial.println("Zone 0");

  if(troubleshoot == 1)
  {
  if(troubleshoot == 1) Serial.println(joystick_hor_value);
  if(troubleshoot == 1) Serial.println(joystick_ver_value);
  if(troubleshoot == 1) Serial.println(joystick_thrust_on);
  }
```

```
  //assert thrusters to move forward
  analogWrite(left_thruster,  0);
  analogWrite(right_thruster, 0);

  //assert LEDs
  digitalWrite(left_LED,  LOW);       //de-assert left LED
  digitalWrite(right_LED, LOW);       //de-assert right LED
  }

//*************************************************************
//*************************************************************
//process different joystick zones
//*************************************************************
//Case 1:
//Inputs:
//  X channel between 2.42 to 2.54 VDC - null zone
//  Y channel <= 2.42 VDC
//Output:
//  Move forward - provide same voltage to left and right thrusters
//*************************************************************

if((joystick_hor_value > 2.42)&&(joystick_hor_value < 2.54)&&
    (joystick_ver_value <= 2.42))
  {
  if(troubleshoot == 1) Serial.println("Zone 1");

  //scale joystick vertical to value from 0 to 1
  joystick_ver_value = 2.42 - joystick_ver_value;
  if(troubleshoot == 1) Serial.println(joystick_hor_value);
  if(troubleshoot == 1) Serial.println(joystick_ver_value);
  if(troubleshoot == 1) Serial.println(joystick_thrust_on);

  //assert thrusters to move forward
  analogWrite(left_thruster,  joystick_ver_value);
  analogWrite(right_thruster, joystick_ver_value);

  //assert LEDs
  digitalWrite(left_LED, HIGH);       //assert left LED
  digitalWrite(right_LED,HIGH);       //assert right LED
```

```
  }

//*************************************************************
//*************************************************************
//Case 2:
//Inputs:
//  X channel <= 2.42 VDC
//  Y channel <= 2.42 VDC
//Output:
//  Move forward and bare left
//  - Which joystick direction is asserted more?
//  - Scale PWM voltage to left and right thruster accordingly
//*************************************************************

if((joystick_hor_value <= 2.42)&&(joystick_ver_value <= 2.42))
  {
  if(troubleshoot == 1) Serial.println("Zone 2");

  //scale joystick horizontal and vertical to value from 0 to 1
  joystick_hor_value = 2.42 - joystick_hor_value;
  joystick_ver_value = 2.42 - joystick_ver_value;

  if(troubleshoot == 1)  Serial.println(joystick_hor_value);
  if(troubleshoot == 1)  Serial.println(joystick_ver_value);
  if(troubleshoot == 1)  Serial.println(joystick_thrust_on);

  //assert thrusters and LEDs
  if(joystick_hor_value > joystick_ver_value)
    {
    analogWrite(left_thruster, (joystick_hor_value - joystick_ver_value));
    analogWrite(right_thruster, joystick_hor_value);

    //assert LEDs
    digitalWrite(left_LED,  HIGH);        //assert left LED
    digitalWrite(right_LED,  HIGH);       //assert right LED
    }
  else
    {
    analogWrite(left_thruster, joystick_ver_value);
```

```
    analogWrite(right_thruster, (joystick_ver_value - joystick_hor_value));

    //assert LEDs
    digitalWrite(left_LED,  HIGH);        //assert left LED
    digitalWrite(right_LED,  HIGH);       //assert right LED
    }
  }

//************************************************************
//************************************************************
//Case 3:
//Inputs:
//  X channel <= 2.42 VDC
//  Y channel between 2.42 to 2.54 VDC - null zone
//Output:
//  Bare left
//************************************************************

if((joystick_hor_value <= 2.42)&&(joystick_ver_value > 2.42)&&
    (joystick_ver_value < 2.54))
  {
  if(troubleshoot == 1)  Serial.println("Zone 3");

  //scale joystick vertical to value from 0 to 1
  joystick_hor_value = 2.42 - joystick_hor_value;

  if(troubleshoot == 1)  Serial.println(joystick_hor_value);
  if(troubleshoot == 1)  Serial.println(joystick_ver_value);
  if(troubleshoot == 1)  Serial.println(joystick_thrust_on);

  //assert thrusters
  analogWrite(left_thruster, 0);
  analogWrite(right_thruster, joystick_hor_value);

  //assert LEDs
  digitalWrite(left_LED,  LOW);        //de-assert left LED
  digitalWrite(right_LED,  HIGH);      //assert right LED
  }
```

```
//*************************************************************
//*************************************************************
//Case 4:
//Inputs:
//  X channel <= 2.42 VDC
//  Y channel >= 2.54 VDC
//Output:
//  Bare left to turn around
//*************************************************************

if((joystick_hor_value <= 2.42)&&(joystick_ver_value >= 2.54))
  {
  if(troubleshoot == 1) Serial.println("Zone 4");

  //scale joystick horizontal and vertical to value from 0 to 1
  joystick_hor_value = 2.42 - joystick_hor_value;
  joystick_ver_value = joystick_ver_value - 2.54;

  if(troubleshoot == 1)  Serial.println(joystick_hor_value);
  if(troubleshoot == 1)  Serial.println(joystick_ver_value);
  if(troubleshoot == 1)  Serial.println(joystick_thrust_on);

  //assert thrusters and LEDs
  if(joystick_hor_value > joystick_ver_value)
    {
    analogWrite(left_thruster, 0);
    analogWrite(right_thruster, (joystick_hor_value-joystick_ver_value));

    //assert LEDs
    digitalWrite(left_LED,  LOW);         //de-assert left LED
    digitalWrite(right_LED,  HIGH);       //assert right LED
    }
  else
    {
    analogWrite(left_thruster, 0);
    analogWrite(right_thruster, (joystick_ver_value-joystick_hor_value));

    //assert LEDs
    digitalWrite(left_LED,  LOW);         //de-assert left LED
```

```
    digitalWrite(right_LED,  HIGH);       //assert right LED
    }
   }

//*************************************************************
//*************************************************************
//Case 5:
//Inputs:
//  X channel between 2.42 to 2.54 VDC - null zone
//  Y channel >= 2.54 VDC
//Output:
//  Move backward - provide same voltage to left and right thrusters
//*************************************************************

if((joystick_hor_value > 2.42)&&(joystick_hor_value < 2.54)&&
   (joystick_ver_value >= 2.54))
  {
  if(troubleshoot ==1) Serial.println("Zone 5");

  //scale joystick vertical to value from 0 to 1
  joystick_ver_value = joystick_ver_value - 2.54;

  if(troubleshoot == 1)  Serial.println(joystick_hor_value);
  if(troubleshoot == 1)  Serial.println(joystick_ver_value);
  if(troubleshoot == 1)  Serial.println(joystick_thrust_on);

  //assert thrusters
  analogWrite(left_thruster, 0);
  analogWrite(right_thruster, joystick_ver_value);

  //assert LEDs
  digitalWrite(left_LED,  LOW);         //de-assert left LED
  digitalWrite(right_LED,  HIGH);       //assert right LED
  }

//*************************************************************
//*************************************************************
//Case 6:
//Inputs:
```

```
//  X channel >= 2.54 VDC
//  Y channel >= 2.54 VDC
//Output:
//  Bare left to turn around
//************************************************************

if((joystick_hor_value >= 2.54)&&(joystick_ver_value >= 2.54))
  {
  if(troubleshoot == 1) Serial.println("Zone 6");

  //scale joystick horizontal and vertical to value from 0 to 1
  joystick_hor_value = joystick_hor_value - 2.54;
  joystick_ver_value = joystick_ver_value - 2.54;

  if(troubleshoot == 1)  Serial.println(joystick_hor_value);
  if(troubleshoot == 1)  Serial.println(joystick_ver_value);
  if(troubleshoot == 1)  Serial.println(joystick_thrust_on);

  //assert thrusters and LEDs
  if(joystick_hor_value > joystick_ver_value)
    {
    analogWrite(left_thruster, (joystick_hor_value-joystick_ver_value));
    analogWrite(right_thruster, 0);

    //assert LEDs
    digitalWrite(left_LED,  HIGH);        //assert left LED
    digitalWrite(right_LED,  LOW);        //de-assert right LED
    }
  else
    {
    analogWrite(left_thruster, (joystick_ver_value-joystick_hor_value));
    analogWrite(right_thruster, 0);

    //assert LEDs
    digitalWrite(left_LED,  HIGH);       //assert left LED
    digitalWrite(right_LED,  LOW);       //de-assert right LED
    }
   }
```

```
//*****************************************************************
//*****************************************************************
//Case 7:
//Inputs:
//  X channel >= 2.54 VDC
//  Y channel between 2.42 to 2.54 VDC - null zone
//Output:
//  Bare right
//*****************************************************************

if((joystick_hor_value >= 2.54)&&(joystick_ver_value > 2.42)&&
   (joystick_ver_value < 2.54))
  {
  if(troubleshoot == 1) Serial.println("Zone 7");

  //scale joystick vertical to value from 0 to 1
  joystick_hor_value = joystick_hor_value - 2.54;

  if(troubleshoot == 1)  Serial.println(joystick_hor_value);
  if(troubleshoot == 1)  Serial.println(joystick_ver_value);
  if(troubleshoot == 1)  Serial.println(joystick_thrust_on);

  //assert thrusters
   analogWrite(left_thruster, joystick_hor_value);
   analogWrite(right_thruster, 0);

  //assert LEDs
  digitalWrite(left_LED,  HIGH);       //assert left LED
  digitalWrite(right_LED,  LOW);       //de-assert right LED
  }

//*****************************************************************
//*****************************************************************
//Case 8:
//Inputs:
//  X channel >= 2.54 VDC
//  Y channel <= 2.42 VDC
//Output:
//  Move forward and bare right
```

```
//  - Which joystick direction is asserted more?
//  - Scale PWM voltage to left and right thruster accordingly
//*************************************************************

if((joystick_hor_value >= 2.54)&&(joystick_ver_value <= 2.42))
  {
  if(troubleshoot == 1) Serial.println("Zone 8");

  //scale joystick horizontal and vertical to value from 0 to 1
  joystick_hor_value = joystick_hor_value - 2.54;
  joystick_ver_value = 2.42 - joystick_ver_value;

  if(troubleshoot == 1)  Serial.println(joystick_hor_value);
  if(troubleshoot == 1)  Serial.println(joystick_ver_value);
  if(troubleshoot == 1)  Serial.println(joystick_thrust_on);

  //assert thrusters and LEDs
  if(joystick_hor_value > joystick_ver_value)
    {
    analogWrite(left_thruster, joystick_hor_value);
    analogWrite(right_thruster, (joystick_hor_value-joystick_ver_value));

    //assert LEDs
    digitalWrite(left_LED,  HIGH);        //assert left LED
    digitalWrite(right_LED,  HIGH);       //assert right LED
    }
  else
    {
    analogWrite(left_thruster, (joystick_ver_value-joystick_hor_value));
    analogWrite(right_thruster, joystick_ver_value);

    //assert LEDs
    digitalWrite(left_LED,  HIGH);        //assert left LED
    digitalWrite(right_LED,  HIGH);       //assert right LED
    }
  }
}

//*************************************************************
```

4.3.8 CONTROL HOUSING LAYOUT

A Plano Model 1312-00 water-resistant field box is used to house the control circuitry and rechargeable battery. The battery is a rechargeable, sealed, lead-acid battery, 12 VDC, with an 8.5 amp-hour capacity. It is available from McMaster-Carr (#7448K82). A battery charger (12 VDC, 4–8 amp-hour rating) is also available (#7448K67). The layout for the ROV control housing is provided in Figure 4.13.

The control circuitry consists of two connected plastic panels, as shown in Figure 4.13. The top panel has the on/off switch, the LED thruster indicators (left, dive, and right), an access hole for the joystick, and a 1/4-inch jack for the battery recharge cable.

The lower panel is connected to the top panel using aluminum spacers, screws, and corresponding washers, lock washers, and nuts. The lower panel contains Arduino UNO R3 equipped with the thumb joystick on a DIY Adafruit Proto Shield (#2077). The UNO R3 is connected to the lower panel using a Jameco standoff kit (#106551). The UNO R3 is interfaced to the thrusters via interface circuitry described in Figures 4.9 and 4.10. The interface printed circuit board is connected to the four-conductor thruster cable via a four-conductor Jones connector.

4.3.9 FINAL ASSEMBLY TESTING

The final system is tested a subassembly at a time. The following sequence is suggested.

- Recheck all waterproofed connections. Reapply waterproof caulk as necessary.
- With the thumb joystick on the breakout board disconnected from UNO R3, test each LED indicator (left, dive, and right). This is accomplished by applying a 5 VDC signal in turn to the base resistor of each LED drive transistor.
- In a similar manner each thruster (left, right, and vertical) may be tested. If available, a signal generator may be used to generate a pulse width modulated signal to test each thruster.
- With power applied, the voltage regulators aboard the printed circuit board should be tested for proper voltages.
- The output voltages from the thumb joystick may be verified at the appropriate header pins.
- With the software fully functional, the thumb joystick on the breakout board may be connected to UNO R3 for end-to-end testing.

4.3.10 FINAL ASSEMBLY

The fully assembled ROV is shown in Figure 4.14.

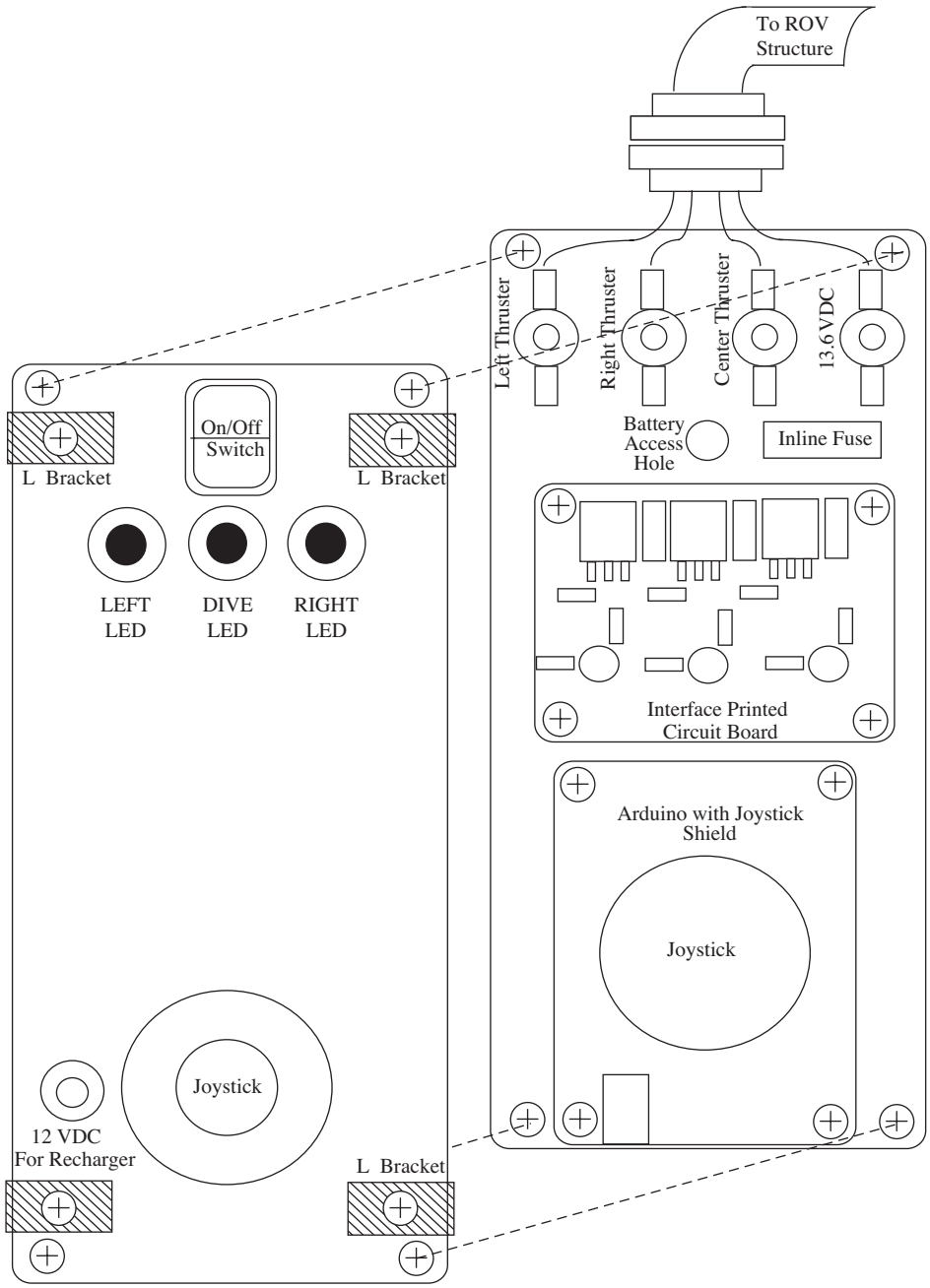


Figure 4.13: ROV control housing layout.

Figure 4.14: ROV fully assembled (photo courtesy of J. Barrett).

4.4 SUMMARY

In this chapter, we have provided design details for two Arduino-based systems: a remote weather station and an ROV.

4.5 REFERENCES

[1] Earl, B. (2018). Adafruit Data Logger Shield.

[2] Bohm, H. and Jensen, V. (1997). *Build Your Own Underwater Robot and Other Wet Projects*. Westcoast Words.

[3] SeaPerch, `www.seaperch.org`. 171, 173

[4] *LM34 Precision Fahrenheit Temperature Sensors,(SNIS161D)*. Texas Instruments: 2016. 156

4.6 CHAPTER PROBLEMS

4.1. Provide the complete sketch for the remote weather station.

4.2. It is desired to updated weather parameters every 15 minutes. Write a function to provide a 15-minute delay.

4.3. Add one of the following sensors to the weather station:

- barometer
- hygrometer

You will need to investigate background information on the selected sensor, develop an interface circuit for the sensor, and modify the weather station code.

4.4. For the SeaPerch ROV provide a powered dive and surface thruster. To provide for a powered dive and surface capability, the ROV must be equipped with a vertical thruster equipped with an H-bridge to allow for motor forward and reversal.

4.5. For the SeaPerch ROV provide left and right thruster reverse. Currently, the left and right thrusters may only be powered in one direction. To provide additional maneuverability, the left and right thrusters could be equipped with an H-bridge to allow bi-directional motor control.

4.6. For the SeaPerch ROV provide proportional speed control with bi-directional motor control. Both of these advanced features may be provided by driving the H-bridge circuit with PWM signals.

4.7. Develop an embedded system controlled dirigible/blimp (www.microflight.com, www.rctoys.com).

4.8. Develop a trip odometer for your bicycle (Hint: use a Hall Effect sensor to detect tire rotation).

4.9. Develop a timing system for a four-lane pinewood derby track.

4.10. Develop a playing board and control system for your favorite game (Yahtzee, Connect Four, Battleship, etc.).

4.11. You have a very enthusiastic dog that loves to chase balls. Develop a system to launch balls for the dog.

Author's Biography

STEVEN F. BARRETT

Steven F. Barrett, Ph.D., P.E., received a B.S. in Electronic Engineering Technology from the University of Nebraska at Omaha in 1979, an M.E.E.E. from the University of Idaho at Moscow in 1986, and a Ph.D. from The University of Texas at Austin in 1993. He was formally an active duty faculty member at the United States Air Force Academy, Colorado and is now the Associate Dean of Academic Programs at the University of Wyoming. He is a member of IEEE (senior) and Tau Beta Pi (chief faculty advisor). His research interests include digital and analog image processing, computer-assisted laser surgery, and embedded controller systems. He is a registered Professional Engineer in Wyoming and Colorado. He co-wrote with Dr. Daniel Pack several textbooks on microcontrollers and embedded systems. In 2004, Barrett was named "Wyoming Professor of the Year" by the Carnegie Foundation for the Advancement of Teaching and in 2008 was the recipient of the National Society of Professional Engineers (NSPE) in Higher Education, Engineering Education Excellence Award.

Index