8-2012

Satisfying STEM Education Using the Arduino Microprocessor in C Programming

Brandyn Moore Hoffer
East Tennessee State University

Satisfying STEM Education Using the Arduino Microprocessor in C Programming

A thesis

presented to

the faculty of the College of Business and Technology

Department of Engineering Technology, Surveying and Digital Media

In partial fulfillment

of the requirements for the degree

Masters of Science in Technology

with a concentration in Engineering Technology

by

Brandyn M. Hoffer

August 2012

Keywords: Arduino, microprocessor, technology education, programming, computer science, STEM

ABSTRACT

Satisfying STEM Education Using the Arduino Microprocessor in C Programming

by

Brandyn M. Hoffer

There exists a need to promote better Science Technology Engineering and Math (STEM) education at the high school level. To satisfy this need a series of hands-on laboratory assignments were created to be accompanied by 2 educational trainers that contain various electronic components. This project provides an interdisciplinary, hands-on approach to teaching C programming that meets several standards defined by the Tennessee Board of Education. Together the trainers and lab assignments also introduce key concepts in math and science while allowing students hands-on experience with various electronic components. This will allow students to mimic real world applications of using the C programming language while exposing them to technology not currently introduced in many high school classrooms. The developed project is targeted at high school students performing at or above the junior level and uses the Arduino Mega open-source Microprocessor and software as the primary control unit.

DEDICATION

This thesis is dedicated to my family who has continued to encourage me to pursue higher education and a successful future. First, I would like to dedicate this to my mother Carolyn Hoffer who supported my curiosity in technology at a young age despite the damage it caused to numerous electronic devices. Her continuous support of my decisions and encouragement over the years kept me in school despite changing majors and universities several times. Her willingness to let me make my own decisions in life has allowed me to develop into the person I am today and I could not ask for better. I would like to thank my father Brett Hoffer who has supported me in my decisions and assumed much of the financial burden of the extreme price of putting a kid through college. He has taught me the value of hard work and dedication and been an ever present example of how those qualities directly affect success in the “real world” and personal life. I would also like to thank my grandparents Rolland Vogt and the late Eva Vogt who supplied me with magnifying glasses, VCRs, and other electronics that originally sparked my interest in science and technology. Their praise for learning how to hook up and fix various devices and computers made education in the field of technology an obvious and satisfying choice once I was ready to make it.

ACKNOWLDEGEMENTS

I would like to thank Dr. Paul Sims for the education and opportunities he has provided me with over the years. His interest in my education is the reason I joined the M.S. of Engineering Technology program at ETSU, and I am very thankful for his knowledge and generosity.

I would like to thank Mr. Garth Gearing for volunteering to oversee all of my independent studies and Topics in Technology courses. His help has allowed me to gain valuable hands-on experience and provided me a continuous outlet for exploration in the field of Electronic Technology.

I would like to thank Mr. William Hemphill for dedicating his time and knowledge in helping create the Trainers. If it wasn’t for his valuable input, the prototype trainer would be a dysfunctional wreck comprised of plastic, superglue, and duct tape.

I would also like to thank Matt Crum and Benjamin McMurry for their assistance with the creation of the first prototype trainer as well as their input and knowledge on mechanical engineering and design.

Finally I would like to thank the Arduino open source development team and all of the programmers who either contributed to my knowledge or provided code used in the project created for this thesis. These people are listed in Appendix A.

CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In an effort to create an educational tool that would provide students with a fundamental understanding of C programing as well as exposure to electronic technology, two trainers and associated labs were created. This project is meant to assist high school students performing at or above the junior level of education. The Trainers consist of two separate units each comprised of multiple components designed around several laboratory assignments. The laboratory assignments each aim to progress with more complicated code once the student has used the previously introduced code commands in various programs (referred to as sketches by the Arduino software). The resulting prototypes and lab assignments are constructed in a way that makes them easy to use and easy to complete during allotted class time. The result of completion of the labs using the associated trainers will be a proficiency in the basics of C programing as well as exposure to multiple technology components, key ideas in using both analog and digital electronics, and the ability to interface with various sensors and displays.

The Arduino microprocessor was chosen for its ease of use, its capabilities, its relatively cheap purchase price, and the fact that it is programmed in the C language; making the knowledge gained by programming applicable to multiple areas in computer technology. The Arduino is also an "open source" device consisting of a hardware unit available in multiple configurations for various technology applications. The Arduino Mega 2560 unit was chosen as the key piece of hardware for use with the trainers because of its abundance of inputs and outputs and its ability to demonstrate all necessary C commands. The Arduino units are programmed by any computer with a standard operating system using open source software. The associated trainers are used by simply connecting different plugs to various inputs and outputs to the Arduino unit. All necessary connections to make the hardware and electrical components function have already been made allowing students to focus solely on their understanding of the programs being created and the technology concepts introduced.

The lab manual introduces key commands that work together to accomplish specific tasks with the developed trainers or with a computer. The lab manual begins by illustrating in detail how to connect the Arduino to a computer with a Windows 7 operating system, and begin communication. The lab assignments build in a logical order continuing to introduce new commands and techniques while using previously introduced commands. Students are limited to using only those commands introduced in each lab section as well as those introduced previously in an effort to guide students programming; forming easily gradable and consistent finished sketches. Examples of functioning sketches for each finished lab assignment have also been complied to have a functioning program for educators to grade with. Each lab also introduces key concepts in electronics to better aid students in understanding the tasks they are performing as well as educate them to these concepts.

The trainer and associated labs meets several of the required course content standards for education in C programing as well as others in math and technology courses as defined by the Tennessee Board of Education. The individual components were chosen for their ability to demonstrate successful use of C code. LEDs serve as indicators of producing successful control statements that turn on various outputs. The LEDs provide an easy to notice indication that a particular control statement was correctly formed or an output was properly specified. These are used in a variety of ways to demonstrate binary numbers, analog outputs through dimming, and simple output control. A potentiometer is used as the first analog input. This device allows for the voltage on an analog pin to be easily changed simply by twisting a knob. A photo resistor was used as a light sensor to vary the analog input as a function of the presence of light in the room. These allow for control statements to be used with variable inputs and require proper data formatting.

To further demonstrate the idea of resistance as well as provide a lab dealing with more complicated control statements several resistors were used to form multiple voltage divider circuits to measure the resistance of an external resistor. The photoresistor was used again in a second trainer to measure lighting levels. This was chosen for its ability to demonstrate output control based on analog inputs and to expose students to a transducer device. Simple push buttons were used as the primary means to demonstrate input control statements and simulate digital electronics. These are featured on the first trainer to illuminate LEDs and to provide readout on the computer through the manipulation of binary numbers.

More complicated electronics such as the matrix keypad, the LCD screen and the Ultrasonic distance sensor expose students to highly useful components. The keypad and LCD screen also allow for the use of library or header files in C programming, allowing students to call data from a prepared file. Through various labs these electronics allow students to develop algorithms and use multiple data types including characters and arrays. Controlling these devices requires students to use more complicated mathematical functions and conditional statements allowing for exposure to many of the basics of C programming.

The prototypes and associated lab manuals as well as their design specifications and components will be kept by East Tennessee State University. These prototypes will hopefully be adopted by the college of Engineering Technology for minor scale production and sold or donated to local area High Schools as a learning tool should they show interest. Mr. Garth Gearing's automation and robotics lab has the necessary tools and equipment to meet small scale production of these units and is currently (summer semester of year 2012) being converted into a manufacturing work cell with multiple robots on a conveyer system as well as an automatic storage and retrieval palletizing device.

These trainers and associated labs will aid students gaining a greater understanding of electronics and programming through hands-on application. The resulting education should spark students' interest in computer science, technology, and engineering majors improving education in Tennessee and meeting the much needed demand for STEM graduates. Members of the "Rising Above the Gathering Storm" (2010, p. ix) committee discussed the current state of STEM education and its impact; "participants expressed concern that the weakening of science and technology in the United states would inevitably degrade its social and economic conditions, and in particular erode the ability of its citizens to compete for high-quality jobs". To produce the best possible trainer for distribution to schools, revisions to the final prototypes may be made to creating a more manufactureable trainer unit. The current design meets all necessary functionality and the associated labs introduce a basic proficiency in using a variety of C programming commands to meet various tasks and control a variety of inputs and outputs.

CHAPTER 2

MEETING EDUCATIONAL NEEDS

The Purpose of the Project

The initial goal of the project was to design an educational tool that allowed students to investigate their interest in science and technology courses in high school in a unique way. In order to properly meet course standards for the various technology related courses, high schools often segment classes into highly specific curriculum. While these classes must meet or exceed the standards for particular states educational requirements they often do not allow for students to observe the way in which many of these courses work together. The resulting project was to allow for students to easily use technology in a hands-on approach while meeting standard course requirements.

In order to narrow the scope of the project a series of laboratory assignments were created for use in a high school class room. These labs were to help meet the core requirements defined by the Tennessee Department of Education curriculum standards for C programming. These labs were accompanied by two educational trainers that allowed for technology and engineering ideas to be applied through the use of C programming. The trainer exposes students to various technologies and concepts that can be used in related STEM fields, thus promoting education and interest in Science Technology Engineering and Math. This integrated approach will show students how various fields and knowledge in these areas work together to provide “real world” solutions.

The Arduino microprocessor is programmed in the C language. It can interface with any analog or digital device that does not exceed its specifications. Any device that can be powered by 5V and requires less than 40 mA of current can be used without the need for separate power supplies. In order for students to use this device for C programming the trainers were developed to incorporate electronics that could teach key concepts of the language. The trainers use simple 8 pin stackable headers as plugs to interface with the Arduino. These plugs correspond with the electronics on the trainers to produce easily readable outputs showing proper completion of a program. All electronics were chosen to demonstrate key programming or technology concepts.

Students first learn how to specify and format outputs through the use of LEDs. The first trainer contains two sets of red, yellow, and green LEDs in a stoplight configuration. These are used in the second lab teaching students about the basic structure of programming. These also teach students to activate digital five volt outputs on the trainer that turn the LEDs on and off. If the program is written correctly the LEDs simulate a stoplight. This allows for the output to be observed by simulating a device most are familiar with. Errors in the programming will also be easily noticed so the program can be debugged and retested.

Conditional statements are used in C to perform specific actions based on some variable or event. "There are two forms of conditional statement: with or without an else clause" (Harbison & Steele, 1984, p. 202). The "if…else" statement is used with momentary push buttons to act as inputs and LEDs to act as outputs. Four buttons and four LEDs are featured on the first trainer to teach students how to trigger digital outputs based on the condition of digital inputs.

To further the understanding of conditional statements and introduce the concepts of Binary and ASCII, a keypad was featured on the first trainer. This keypad sends a 4-bit binary output to various digital pins to be interpreted by a program. The binary value read by the Arduino is then displayed. Student use the Boolean operators with conditional statements to interpret signals from the keypad. The keypad is also used to introduce different variables and data types. In one lab students write a program that displays the key pressed in its binary for on four LEDs. This is done by reading the keypad's output and using math to reduce the ASCII encoded number to its actual decimal value.

The first trainer features two analog input devices. These are a rotary potentiometer and a photo sensor. Interfacing with these devices requires more complicated control statements and the use of variables. These also demonstrate the principals of the analog to digital converter used by the Arduino to interpret these signals. Another set of two ultra-bright LEDs are used to visually demonstrate an analog output.

The second trainer uses an LCD screen to make the device portable. To control the LCD screen a header (.h) file is used to display characters on the screen. Header files, discussed more in depth later, are used to simplify code and retrieve data. The screen works in combination with the other devices on the second trainer to display the value of variables and print specific characters based on the condition received. A matrix keypad is featured on the second trainer. This was used to interface with the LCD. Students use various buttons on the keypad to specify what task the Arduino should be performing and send data to the Arduino.

In the first lab using these two devices, students are instructed to print characters to the LCD screen based on the buttons pressed. This introduces character arrays and how to use them in a program to produce certain actions. Students then create a simple calculator using these two devices. This allows for numerical arrays to be used for calculations. Students develop an algorithm to do math by calling the data stored in the array and performing the specified math function.

Two safety sockets are located at the bottom of the second trainer. These use a voltage divider in series with an external resistor to display the resistance measured in ohms. Students will use this to compare measured and theoretical values of several resistors. The code to perform this function is added onto the previous program using the keypad and LCD. Loops are introduced to demonstrate how to execute only a portion of C code under certain conditions. Students will also use loops with the photoresistor on the second trainer to calculate an average reading over time. The analog reading will then be converted to a measure of luminance and displayed on the screen.

The previous labs introduce the knowledge necessary to interface with the ultrasonic sensor. By the time students write C code using this sensor, they will have reasonably developed programming skills and know how to handle multiple variables and data types. This sensor is used to calculate distance in a program based on the time in milliseconds read by the Arduino. Students are instructed to interpret the received data and display the measurement in inches and centimeters. Students will work in a group to interface with this device allowing them to present a solution using the knowledge gained in C code. A picture of the two trainers can be seen in Figure 1.

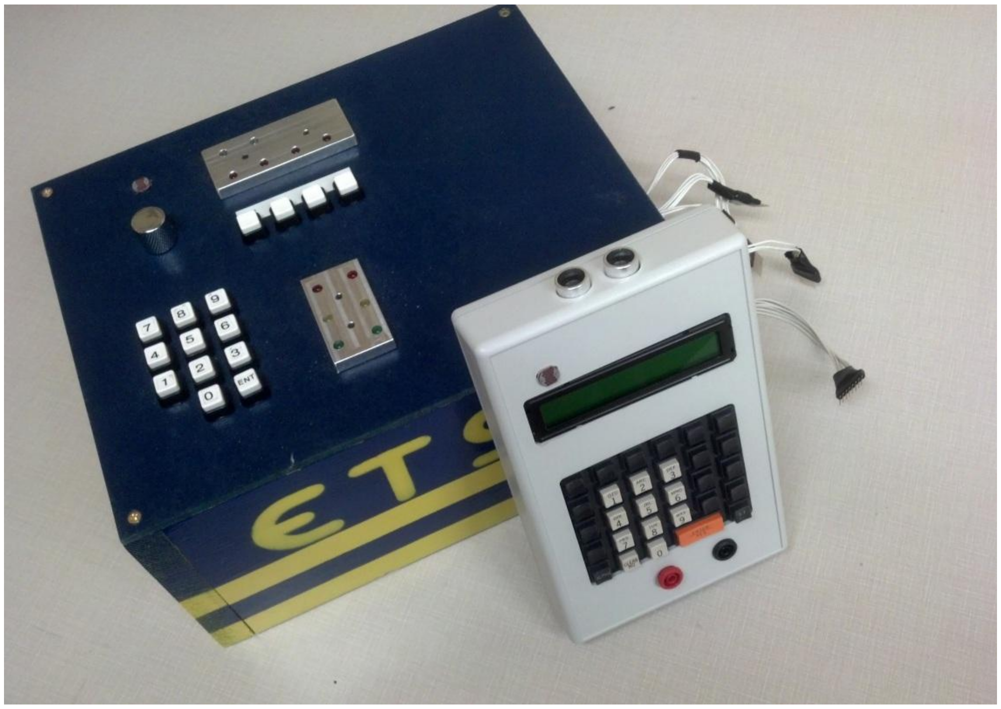


Figure 1. Arduino Trainers

Interfacing with these devices allows for a unique educational opportunity in C programming. Currently there are issues with the way STEM education is being taught at the high school level. These issues are defined in the next section of this paper. This approach to C programming aims to better education in these fields by incorporating new ways of teaching. The exposure to technology and math in the lab assignments should satisfy several of the needs to better STEM education in Tennessee high schools.

STEM Needs in Education

The National Science Board (NSB) of the U.S. National Science Foundation clearly states the need to promote STEM education in high school to encourage an educated workforce in these fields. “Almost 30 percent of students in their first year of college are forced to take remedial science and math classes because they are not prepared to take college-level courses” (National Science Board, 2007, p. 2-3). The United States has historically been a source for some of the most innovative science and technology advances and should strive to maintain that stance. This is not possible if STEM education is not properly taught. While there are many factors that contribute to the lack of properly educated students in the U.S., sparking students’ interest in these fields and teaching subjects in new ways can improve students’ abilities in these subjects.

The global market is now making the technology market far more competitive than in previous years. In order to compete and keep its position as a world leader for technology and innovation it is important that the U.S. must properly educate students to be competent in the STEM fields that drive innovation in this new age. Students must be exposed to this education during high school or earlier in order to spark their interest and ensure they have what it takes to receive a valued college degree in these areas. Unfortunately the U.S. is falling behind in education. “The World Economic Forum ranks the United States 48th in quality of mathematics and science education” according to a 2010 report evaluating the competitiveness of the U.S. in the global market (By Members of the 2005 “Rising Above the Gathering Storm” Committee et al., 2010, p. 6). While there are many social, political, and economic factors that affect education in the United States, the way in which students are being taught plays a big role in education.

Currently many teachers in the U.S. are not fully capable of teaching the STEM fields effectively. Classes are often taught in a subject by subject basis with little interdisciplinary approach. Teachers are often not given the tools and flexibility needed to become educated on the subjects enough to teach them effectively. The STEM national action plan of the NSB states that "the Nation faces a chronic shortage of qualified teachers who are adequately prepared and supported to teach STEM discipline effectively" (NSB, 2007, p. 5). This is often the case because of the inability to attract highly qualified professionals into teaching jobs. There is too little flexibility in teacher compensation and other professional positions offer much higher salaries and conditions than most teaching careers. The question remains: if there is a need to improve STEM education, how is that done?

An Interdisciplinary Approach to C Programming

In order to Improve STEM education it is important to recognize that these fields do not operate independently in real world applications. Many high school classes focus on teaching the theory behind the subjects with little real world application. Individual classes often do not incorporate concepts from other related subjects. The STEM fields in particular all work together to form real world problem solving skills. Princeton University has created an undergraduate college program known as Integrated Science that incorporates multiple aspects of various science disciplines into a singular course without the single subject barriers of traditional schools. David Botstein, the director of the Lewis-Sigler Institute for Integrative Genomics at Princeton states that "Any budding researcher needs a foundation in several fields to be able to work on the most important problems confronting scientists today" (Princeton University, 2012).

There are many expert opinions on how to better education that range from increasing salaries, to making mandatory, across the country, benchmark standards. These ideas, however, fall outside of the scope of the created project for this thesis and will not be discussed. Douglas, Iversen, and Kalyandurg (2004) of the American Society for Engineering Education (ASEE) have published a paper entitled Engineering in the K-12 Classroom that discusses several of the issues with science and engineering education. Among the six major recommendations made to improve science and engineering education in the U.S., three can be improved directly at the classroom level. The first of which is to use more hands-on learning that will make curriculum less theory-based and more context-based. The second is to take interdisciplinary approaches to subjects. The last recommendation that falls within the scope of the classroom is to use and improve K-12 teachers. While this project cannot improve teachers in these subjects, it can provide them with a tool that will make their jobs easier and help ease the perception that engineering and science are hard subjects.

Many teachers were surveyed in the previously mentioned ASEE report on their views of engineering and associated education. Some of the key questions in this survey also illustrate what is needed to better engineering education. According to the paper "teachers believe that majoring in engineering in college is harder than majoring in many other subjects" …furthermore "39.1% strongly agreed and 51.2% agreed with the statement, 'Understanding more about engineering can help me become a better teacher'" (Douglas et al., 2004, p.10). While the STEM fields most certainly are not easy, the perception that they are too difficult to enter can be changed with the proper approach to education. A series of hands-on assignments that teaches the fundamentals of these subjects while demonstrating the ideas behind them can do a lot to better understanding amongst both students and teachers.

The ASEE idea of improving hands-on learning goes hand in hand with the problem of discontinuity between public education and workforce requirements. The NSB agrees that more should be done to provide the type of useful education that has real world applications rather than using memorization and theory alone to teach subjects. “All stakeholders should make a serious effort to minimize the current disconnect between high school graduation requirements and the skills and knowledge required to succeed in higher education and the workforce” (NSB, 2007, p. 19). Providing students with skills that will help in college level classes and their future careers will not only provide a more useful education to the students but ease the number of remedial class enrolments and training hours spent on new employees in related fields.

In order to satisfy the principals for improving STEM education in the high school setting the created project was designed with these ideas in mind. Rather than abiding by the traditional way C programming is taught in high schools, an interdisciplinary hands-on approach was taken. C programming is typically taught through the use of lectures and reading for education where the fundamental functions and commands of the language are to be memorized and demonstrated through a few programs. Students do get some “hands-on” experience in the class room using a computer to develop these basic programs, but this method only exposes students to the aspects of computer science for this language. While C is typically used for programming computer applications, it has other applications as well. The Arduino Microprocessor chosen for demonstration of the language is a valuable technology and engineering tool that can accurately show the usefulness of C beyond simply sitting in front of a computer. Using the Arduino to control various electronics on the trainer demonstrates these key commands and how they can be used in control systems and electronic devices. This in turn bridges the gap between a computer language and the technology that can be driven by it.

Meeting Standards While Incorporating Other Disciplines

While many of the recommendations made by the NSB and the ASEE would improve education, the coursework and classes introduced in high schools must still meet the curriculum standards set out by each state. As previously mentioned, in an effort to narrow the scope of this project it was created to meet several of the standards defined out by the Tennessee Department of Education (TDOE) for C programming. While this project may meet course standards in other states, they were not considered in the development of this project. An interdisciplinary class covering many aspects of different subject would likely not meet these standards as a result of not teaching any particular subject with enough depth. The created project satisfies the core requirements set out by TDOE for standards 2.0 - 4.0 in the subject of C Programming Applications. Standard 1.0, however, deals with the history and background of computers and programming and will still be met by standard classroom instruction.

As it stands the TDOE sets core requirements for every class that must be met under state guidelines in order for a high school class to be considered adequate. These standards are divided into sections and defined by learning expectations and performance indicators. The individual standards for each class are listed on the TDOE site at www.tn.gov/education/curriculum.shtml and are available in Adobe PDF format for anyone to view. C Programming Applications is listed under the Computer Technology courses subsection for grades 9-12. There is a prerequisite for the class of Algebra I, meaning most students will not encounter this class until their 10th or 11th year. The class C Programming Applications has four standards as previously mentioned. These are all satisfied by the education students will receive using the created project, excluding the history requirements. The lab manual provides instruction on how to use the individual commands as well as the previously mentioned concepts for technology education. This project is not meant to replace current classroom materials and instruction but rather supplement them allowing for an understanding of C programming applications that are not currently taught in most Tennessee school class rooms. The electronics on the trainers correspond with specific eight pin headers that have been labeled. This makes interfacing with the electronics as simple as plugging them into the Arduino board. Having a simple interface allows for programming to be the main focus of the class and prevents students from having to build circuits.

The following standards were taken from the TDOE website for C Programming Applications core standards. The standards are listed in order and the way in which the created project supports these standards is listed below each one.

Tennessee State Board of Education, C Programming Application Core Standard 1.0 (2005):

Standard 1.0

1. The student will gain competency in the background knowledge of computers and programming.

Learning Expectations

The student will

1. Discuss the history of computers and programming languages.
2. Describe the purposes of the computer and the C language.
3. Discuss the architecture of the computer.
4. Summarize the characteristics of the C programming language.
5. Critique the role of the computer in society.

This standard is satisfied by traditional classroom instruction. While the created project does teach some of the purposes of C programming and many of the characteristics of the language, historical knowledge cannot be taught through the use of the trainer. The standards recommend that students develop a timeline with dates demonstrating their proficiency of the historical knowledge of computers and programming languages.

Tennessee State Board of Education, C Programming Application Core Standard 2.0 (2005):

Standard 2.0

2. The student will use system operations as they relate to C programs on the computer.

Learning Expectations

The student will

1. Demonstrate computer start-up procedures.

2. Discuss the basic structure of the C language.

3. Explain C program entry, listing and editing as it relates to the operating system.

4. Discuss the execution of programs.

5. Explain the storage, retrieval and deletion of programs.

Performance Indicators: Evidence Standard Is Met

The student is able to

• demonstrate the use of a prepared C program on the computer.

This standard is satisfied within the first few lab assignments. The introduction to the crated lab manual describes in detail how begin connecting the Arduino Mega microprocessor and load a basic prepared program. This introduces the start-up procedures and the basic format of the language (requirement 1 and 2). The second requirement is for the basic structure of the C language is discussed. Students will see the basic programming structure for various commands as they are introduced throughout the labs. In Lab 2 students edit the prepared program in the introduction to perform a more complex task by simulating a stoplight rather than blinking a single LED with the prepared program. This also allows students to be familiar with program entry, editing, and execution (requirements 3 and 4). To satisfy requirement 5, the lab manual suggests that programs should be saved often with understandable naming conventions. Code that does not work should be deleted to prevent using it in future labs. This standard suggests that students execute a prepared program then edit the program and execute the new one as a sample performance task, which is performed in lab 2.

Tennessee State Board of Education, C Programming Application Core Standard 3.0 (2005):

Standard 3.0

3. The student will write and document an executable program in C

Learning Expectations

The student will

1. Identify names for variables and their data types.

2. Recognize the symbols for operations and use them in evaluating data.

3. Demonstrate the various methods of obtaining input/output and formatting output.

4. Analyze the task and develop an algorithm.

5. Demonstrate control statements.

6. Identify, illustrate and perform operations on data types in arrays.

7. Identify and use functions.

8. Read and/or write data files for input/output purposes.

9. Debug the program and verify the output of the program.

Performance Indicators: Evidence Standard Is Met

The student is able to

• analyze, design and write a minimum of two executable programs in C for each of the Learning Expectations.

This standard has the widest range of learning expectations for the class and seems to be the educational focus of C Programming applications. As a result the individual requirements were broken down, and the ways in which they are satisfied were discussed.

Requirement 1

Identify names for variable types and their data types. The "int" or integer variable is introduced in lab 3. Integers are used to call digital pins by recognizable names rather than numerical address. Lab 4 introduces the data types "DEC, BIN, HEX," and "OCT". These are the commands used to format a variable as a decimal, binary, hexadecimal, and octal number. Lab 6 introduces the float variable used to express decimal or fractional numbers. Lab 10 introduces the data type byte and the variable "char" or character. These are used to specify characters for an LCD display. Finally lab 11 introduces the "long" variable type that can store much larger numbers than the "int" variable type.

Requirement 2

Recognize the symbols for operations and use them in evaluating data. All Boolean and arithmetic math operations are introduced in lab 3. These are used with control statements to produce the proper output. Lab 4 then introduces mathematical operations to be used with the variable types. Finally in Lab 11, where students create a calculator with basic functions, almost all variable types and most of the Boolean operators are used to perform the required functions.

Requirement 3

Demonstrate the various methods of obtaining input, output, and formatting output. In the first lab students use the serial monitor or text display on the computer as an output. Lab 2 students activate 6 LEDs using digital outputs. Lab 3 further builds on this concept using digital inputs in the form of push buttons. The analog input and output functions are introduced in labs 6 and 7. These are capable of reading or writing variable 5 V signals. All of these inputs and outputs are used in many labs and students must format them properly in order to complete each assignment. If the variable types are not formatted correctly, their sketch will not work.

Requirement 4

Analyze the task and develop an algorithm. When students create a calculator they must form algorithms to handle the mathematical function input on the keypad. These algorithms will allow the Arduino to interpret the incoming variables and perform the correct mathematical function. Other labs require algorithms to be developed to convert an analog input measurement into usable data, or a physical measurement such as the ohm or voltage.

Requirement 5

Demonstrate control statements. In lab 3 where students first used digital inputs and outputs together, control statements allow for the proper output to be formed. These statements become progressively more complicated when handling different variables and data types. The “if” command is used heavily throughout the programs to specify an action to be performed based on different inputs.

Requirement 6

Identify, illustrate, and perform operations on data types in arrays. The calculator lab uses arrays to store numbers input with the keypad. These numbers are then used in the students’ calculations. Character arrays are specified in lab 10 where students learn how to interface with the matrix keypad. These arrays allow for various characters to be received by the Arduino. Control statements are used to specify what should be done when receiving each character.

Requirement 7

Identify and use functions. Functions must be used in all labs to create working programs. Functions let the Arduino know how to handle individual portions of the code. The necessary basic functions for programming the Arduino are the setup and loop functions. These are introduced in the very first lab.

Requirement 8

Read and/or write data files for input and output purposes. Header files, discussed in more detail in Chapter 3, are used as the primary data files. The LCD screen used first in lab 9 uses the library files to display characters to the LCD screen output. A header file is also used to control the matrix keypad used in lab 10. This file allows for data from the keypad to be used as an input for various functions. These files are used in the remaining labs.

<u>Requirement 9</u>

Debug the program and verify the output of the program. Completion of any lab assignment is done by creating the proper output. As such, students must debug their programs to correct errors and produce the desired output. Programs become progressively more complicated increasing the likelihood of errors. Each completed program will give students more experience in debugging these problems. The state of any variable, input, or output can be viewed on the computer's serial monitor allowing for students to see where problems occur. More complicated labs encourage students to use the monitor to verify that calculations are correct.

The sample performance task to meet this standard recommends students convert one unit of measurement to another. This occurs in the final lab where students will view distance measurements on the LCD screen expressed both in centimeters and inches. Students convert analog measurements in several labs prior to the final lab. These measurements are converted to usable data to control outputs based on specific conditions. In lab 13 students will convert an analog measurement into a measurement of illumination. In lab 12 students will use a voltage divider to convert an analog measurement to ohms.

Tennessee State Board of Education, C Programming Application Core Standard 4.0 (2005):

Standard

4. The student will work as a team member to develop an integrated application using C .

Learning Expectations

The student will

1. Define the roles of the team members.

2. Solve a complex task using C .

3. Compare and contrast the advantages of working as a group.

Performance Indicators: Evidence Standard Is Met

The team is able to

• analyze and present the solution of the task.

Upon completion of the Lab 13 students will have a detailed knowledge of the basics of C programming and Arduino specific commands. Lab 14 is designed to be a coordinated group project where student will work in a group to develop a distance measuring device using an ultrasonic range finder. The output of the device will be measured, and students will plot their readings against a ruler of known accuracy. They will use this information to determine an equation for the line of best fit. There measurements will be displayed in centimeters and inches, and measurements must be accurate for completion of the assignment. Students will have their measurements checked against the ruler to verify that the proper calculations were performed.

The labs and the trainers also introduce concepts for the Principles of Technology I and use math in a way that the Common Core State Standards Implementation Plan recommends. According to the plan math should be used to link major topics within grades and provide conceptual understanding with application. Multiple Labs use mathematics in the programming of a sketch to solve algebraic equations. Students will use the calculator created in Lab11 to use the sin, cosine, and tangent functions to verify their lab was created successfully. Many other equations are introduced to handle variables and to produce readable and accurate measurements for the various sensors introduced.

In order to explain the function of the keypads and other electronics, the concept of bits and bytes is introduced. These data types use binary to represent numbers or characters. Binary is explained in Lab 4 where students convert decimal numbers into several other formats. Binary is a base two counting system that uses either a 1 or a 0 to signify high or low, on or off. This lab includes the equation to convert decimal to binary. A Byte consists of eight binary bits that are commonly used to Store American Standard Code for Information Interexchange (ASCII). Computers communicate using binary numbers and encode all readable characters to integers for storage. ASCII translates these to readable characters for the serial monitor from an eight-bit binary code (Johnsonbaugh & Kalin, 1997, p. 15).

A/D converters measure an analog input in and convert it to a binary number in equal steps. “Each unique digital code corresponds to a small range of analog input voltages. This range is 1 LSB [least significant bit] wide” (Lundberg, p. 2). The concept of resolution is also mention. The resolution of an A/D converter is equal to its full scale voltage divided by the number of bits used to represent the voltage. This in turn determines the accuracy of an A/D converter and the accuracy of the voltage students read in the serial monitor.

The A/D converter is used to measure the voltage from either a potentiometer or the various voltage divider circuits. The voltage divider equation is used for measuring resistance in multiple labs. The most notable is Lab 12 where students will use resistor color codes to predict the value of the resistor. They will then use the ohm meter created to measure the resistance of these components and compare them to their predicted values. This satisfies the TDOE standards

3220.3.4.a, 3220.3.4.b, and 3220.3.4.c for Technology I. These standards say students should learn about electrical resistance, compare resistor values to measured values, and calculate resistance (Tennessee State Board of Education, 2008).

Both analog and digital devices are used in the trainer and students will use both types in their labs. Analog outputs use 5V pulse width modulation (PWM) at a rate of 490 hertz (Hz). The analog output is therefore a square wave that “appears” to be changing in voltage as a result of the duration of the pulse. One Hz is one cycle per second, or the amount of time a signal takes to go from high to low and back to high again. The amount of time it takes for the signal to make one cycle is the period of the signal. The frequency is defined as: Frequency = 1 / Period (Davies, 1998, p. 275). Introducing these concepts satisfy the TDOE core standard 3220.3.3.b for the class Technology 1 that applies to understanding frequency and period (Tennessee State Board of Education, 2008).

In order to introduce hands-on technology application, a variety of electrical components and sensors are used to demonstrate the labs. The use of a microprocessor allows students to understand both digital and analog inputs and outputs. The more complex sensors in the trainer, as well as variable voltage circuits, illustrate to students how to setup control programs using technology concepts. Each of the more complex components of the trainer is described in the lab, as well as the fundamental concepts for controlling them. Using these pieces of technology commonly used in Electrical Technology and Engineering classes allows for the interdisciplinary approach described by the NSB and ASEE.

In order to address the issue of helping teachers with complicated subjects the labs were designed to introduce all knowledge pertaining to the assignment. A brief explanation of each command used in the lab is provided, as well as an explanation of how they will be used in the sketch. All the internal connections have already been made and students will only be required to connect labeled plugs to the Arduino Mega microprocessor simplifying the technology side of the trainers. Each lab has already been completed in a sketch that will be provided in digital format along with the trainers. These completed sketches have all been tested to work in the way the labs suggests and use only the commands students are intended to use. They are commented in detail to explain the function of each command set in the sketch so teachers will have no problem understanding how the program controls the various components. The Arduino and associated program is an open source technology that has a variety of information available on the web to explain in detail any concepts or issues not understood by the instructor. This in turn should make the instructor’s job much easier in being able to explain these concepts to students.

CHAPTER 3

THE LAB MANUAL

Format and Reasoning

The Lab manual was created to provide personal instruction to students and teachers on how to complete each task assignment. It is divided into 14 labs that each introduces new C programming commands and key concepts related to the language and the operation of the created trainers. There is also an introductory section that describes how to get the free Arduino software, connect the device, and use it to execute various programs. Each lab is broken down into four sections. These sections are Purpose, Equipment, Commands and Description, and Procedure. The lab manual is written in the second person narrative to reflect that these are instructions provided to the student for completion of the assignment.

The use of the second person narrative was chosen to involve the student in the writing of the labs. The use of the pronouns “you” and “your” reflect that the assignments are the creation of individual students and that instruction is directed at them. These also take on the same narrative that is used in classroom instruction. They can be read aloud by an instructor or fellow student and do not sound as removed as the third person narrative. While the second person narrative is rarely appropriate, it can be used to better provide instruction so long as the specific audience is known. Owens Community College (2012, p. 1) agrees “Second person is effectively used when writing directions; in this case the audience is clearly identified and is seeking the author’s advice.”

The different sections of each lab are consistent throughout the Lab manual. The author of this thesis based the format on a common format observed in the majority of classes containing lab instructions while pursuing a M.S in Technology. This format was also designated and used by the author to provide instructional labs to undergraduate students in the classes of Industrial Electronics, Electronic Communication, and Electronic Principals at East Tennessee State University. A similar format was approved and accepted by Mr. David Jones and Dr. Zhibin Tan while the author was assisting with their laboratory classes. The format differs slightly for these classes based on the specific instructional needs for these classes. For example, rather than a Commands and Description section, a Schematic section was included. This serves the same purpose of defining what would be needed and how it will be used to complete the lab. The instructional labs for these classes also included a question and answer section where students would answer questions to determine their grade for the assignment. This section was omitted in the majority of the labs designed for this project because completion will be defined by a working program in most cases. Grades can be assigned by instructors based on students' adherence to the format preferred by the instructor as well as the logical structure of the commands in the program and their readability.

Each lab is introduced by number and title name. Immediately following the name is the Purpose section. This section describes the goal of the lab and what technology or concepts will be introduced. The next section is the Equipment section. This is simply a list of the equipment that will be needed to for proper completion of the lab. All equipment for the labs is included with the trainers created for the project. Common equipment used in every lab is the Arduino Mega, the Arduino USB cable, and a computer. The Commands and Description section follows and provides a brief description of each new C programming command used in the assignment arranged in an easy to read table. Below the table supplementary information is provided on the various commands to better describe their use in the assignment. These are provided for the student as an easy reference while writing sketches. Any technology concepts or math used in the assignment is also introduced in this section. The Procedure section is a step-by-step instructional set for the student that allows them to complete the assignment. This section contains recommended variable names, tips for completion of the assignment, and the guidelines for the general structure of the sketch. Any connections that need to be made are also clearly explained and pictures and other illustrations are used to provide a visual representation for more complicated ideas.

The step-by-step instructions provided in each lab are written in a way to guide students programming so they produce easy to grade, somewhat consistent sketches. There are a variety of ways to produce the same outcome using different commands and even programs that have few similarities between their commands and structures. Often times C programming commands are interchangeable to perform the same task. In order to minimize the variability in students completed sketches examples are provided and specific variable names are recommended. There is no national guideline for the indentations or spacing used in professional programming, so this was left to the teacher’s discretion. In order to further limit the commands used in a program and prevent variability in the commands used to accomplish specific tasks, students are limited to using only those commands that are listed in the Commands and Description section as well as those introduced in previous labs. The completed sketches included with the created project should be used by instructors as an example of one way in which the sketch can be completed. Students may not produce identical sketches, but they will contain the same commands and same basic structure to accomplish the assigned task.

Lab Descriptions and Concepts

The Lab manual introduces new commands to interface with various components on the trainers in a logical manner that covers many of the basic C programming commands. In this section an excerpt of two labs will be introduced. The full lab Manuel can be viewed in Appendix B, and a brief explanation of each can be found in Appendix C. The C commands learned in each lab will be discussed as well as the concepts introduced. This will show the logical structure of each lab and the reason for introducing the individual commands and technology concepts. All Arduino specific library commands used in the sketches have been defined by the Arduino language reference site at http://arduino.cc/en/Reference/HomePage.

The first section is First Time Setup. This section covers downloading the free Arduino software from the website which comes in a .zip file. Instructions on how to unzip and install the file are also listed. Next, this section covers how to install the Arduino drivers. This occurs automatically on some machines; however, computers running certain antivirus software and firewalls can make this a bit difficult. The user must manually install the drivers for the Arduino Mega 2560. To make this simple, pictures and file paths are included in the information. Next the setup section shows how to connect the computer to the Arduino by specifying the specific board and communication port. To verify that everything was performed correctly, a prewritten example program that blinks an onboard LED on the Arduino is loaded and run to show students how programs are used.

Before explaining the commands used in the labs it is necessary to explain what a header or library file is. These files introduce new commands into the C programming language that make more complicated programming tasks executable through a single command. There are several Arduino specific commands included in every sketch that allow for digital and analog control of the I/O pins. Other standard library files can also be included by depositing them in the library folder of the Arduino program. These files are designated with the .h extension derived from the header file name. According to Johnsonbaugh and Kalin (1997, p.33) these files “request some action before the program is actually translated into machine code.” It is important to note that these files are widely used to simplify the C programming language. Arduino uses many of the basic C commands and automatically loads several header files to simplify I/O functions, timers, and other specific functions commonly used in sketches. This in turn allows students to form programs that would be much more complicated without the use of these files.

Lab 8: Maintaining Lighting Levels Using a Photoresistor

Lab 8 is a good example of a fairly complex program done on the first trainer. This lab was chosen for demonstration because it uses most of the electronics on the first trainer. All of these devices, excluding the photoresistor, have been used in previous labs and should be understood. An excerpt of the purpose section of the lab describes what will be done and how this builds on previous labs.

Lab 8: Maintaining Lighting Levels Using a Photoresistor

Purpose

Now that you can use the Arduino with both analog inputs and outputs you will be using both of them in a sketch. In the previous lab you dimmed a light using the keypad buttons; however, most dimmers you see in houses and lighting devices use a knob (potentiometer) to control the amount of light rather than a keypad. Another common use for dimming lights is to maintain a certain brightness level in an area by supplementing artificial light when natural light

> begins to disappear. Windows and skylights allow for sunlight to illuminate a room without the need for electricity, but the sun eventually sets, and it would be much easier if the lights just came on without you having to move to the nearest light controls to gradually make them brighter as this happens. In this lab you will be programming the Arduino to control the same LEDs used in the last lab. The "0" and "ENT" buttons will be used to select one of two different lighting modes. The first lighting mode will make the LEDs become brighter as the room becomes darker through the use of a Photo Resistor. A photo resistor is a device that has a changing resistance that varies in the presence of light. The second lighting mode will allow for manual adjustment of the LEDs through use of a potentiometer. This setup allows for both manual and automated lights in the home or work environment. You will also need to program a way to make your light sensor more or less sensitive based on the desired lighting levels for the environment.

This lab is a good demonstration of solving a real world problem by using C programming and the technology introduced. It states how this program can be used in a work area, and the analog devices on the trainer all work together to form two specific outputs. One is controlled by a variable resistance from the photoresistor, the other is manually controlled. The next section is the "Equipment" section and lists all the equipment that will be used for the lab. It also mentions optional equipment that may be useful but in no way is required.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Power Cable (Optional)

Arduino Trainer

Light Source

The following section describes the commands used in the program and their function. These are presented in a table and can be used as an easy reference during programming. Table 1 is the excerpt from this section:

Table 1.

Lab 8: Commands and Description Section

Command	**Description**
map (*Intiger, fromLow, fromHigh, toLow, toHigh*) *fromLow, fromHigh, toLow, toHigh* = any number used for calculation	The map( ) function can be used to change the value of a variable or integer to another more appropriate value. Because the Arduino reads analog value from 0 to 1023 these are often used as the *fromLow* (0) and *fromHigh* (1023) numbers. Since analog outputs are 0 to 255 these are often used as the *toLow* (0) and *toHigh* (255). Examples are below.
constrain(*Value, Min, Max*) *Value* = any integer, variable, or other data type. *Min* = the minimum value *Max* = the maximum value	This function "constrains" a value to a particular set of numbers. In other words the *Min* value specified is the lowest number the value can take. Even if the sketch tells it to go lower the number will stay the value specified by *Min*. Similarly *Max* provides an upper limit.

Because some of these commands are fairly complicated they require better explanation. Students are also exposed to the voltage divider equation to understand how the photo resistor works. The photo resistor, and how it functions, is explained in an easy to understand way below the commands and description table. This lab requires multiple statements from a previous lab to be used. As a result a computer tip is added at the end.

More on the Introduced Commands:

In the last sketch the math to “analogWrite( )” the value of the LEDs was performed manually. The “map( )” function allows the Arduino to calculate these values for you. For example, if the integer “x” had a value set by an analog input (0-1023), and the expression was “x = map(x, 0, 1,023, 0, 255);” the integer x would take the value of 255 when the analog input tells it to be 1,023. In other words:

$$x_{final} = x_{initial} \times \frac{225}{1{,}023}$$

Similarly, if x was to take on the value 560 by an analog input, it would be set to ≈139. It is always set to the nearest whole number. If this still does not make sense look at the expression “x = map(x, 0, 100, 0, 1,000);” assuming x is an integer. In this case if x = 70, then x is set to the value of 700. If x = 65 then x is set to the value of 650 by the equation $x_{final} = x_{initial} \times \frac{10}{100}$.

It will be necessary to understand how the photo resistor in the trainer works in order to perform this lab. This resistor responds to a change in light by varying its resistance. The brighter it is the less resistance the photo resistor has, and conversely the darker it is, the more resistance the photo resistor has. This particular photo resistor has an 8K Ω resistance in the light and a 1M Ω resistance in the dark. The particular photo sensor in the trainer may go beyond these ratings in extreme darkness or extreme light; however, these are the ratings for general operation. This photo resistor is in a voltage divider configuration with a 100k Ω resistor that is connected to the output that goes into the Arduino. The equation for the output voltage is:

$$V_{out} = V_{in}\left(\frac{100{,}000}{(R + 100{,}000)}\right)$$

Where “R” is the resistance of the photo resistor and "V_{in}" is the 5V input from the Arduino. If you use the 8K Ω rated value of the photo resistor (exposed to light) and solve the equation you get the output voltage of ≈4.63 V. Similarly, if you solve the equation for the 1M Ω resistance (in the dark) you get an output voltage of ≈0.45V. In other words, more light = lower resistance = higher voltage. This is all a function of Ohm’s Law.

The “constrain( )” command is useful when limiting sensors inputs or integers to a particular set of values. In this lab you will need to figure out how to set various sensitivity levels for the automated lighting. This means you will need to constrain an integer to specific values to make it more or less sensitive based on preference. Say you want to make the LEDs begin to come on when only a little amount of sunlight is lost in the room. In this case you would want to use the “map ( )” command introduced in this section to respond to a somewhat small set of values read from the analog input. Assume you discover that the analog input reads 700 when the room is only somewhat dark, and this is when you want the lights to come full on. Also assume the variable x is assigned a value by the photo sensor through the “analogRead( )” command. In this case you would need to use the command “x = map(x, 700, 1,023, 255, 0);”. This would make the device more sensitive to lighting levels in the specified range (700 to 1,025). In order to prevent an error where the analog input goes "out of bounds" so to speak, you would need to constrain the analog input from the photo sensor. In this case you would need to insert "*PhotoResistor* = constrain(*PhotoResistor*, 700, 1,023);". This prevents a number less than 700 from being read. More information will be in the procedure of this lab.

****Pro-Tip: The sketches are only going to get longer and longer. Use copy and paste as much as you can to make these things take less time. Do this using the right-click on the mouse or by pressing the keys "ctrl+c" to copy and "ctrl+v" to paste; "ctrl+x" is the cut command. If you have the code in a previous lab, copy and paste to save time.**

The procedure section begins by testing the room for specific lighting levels to ensure the photoresistor will work correctly. It describes the steps necessary to use the proposed commands and includes information on how to make all necessary connections. Keep in mind that this is a later lab for the first trainer and students already understand how to make some of these connections and are familiar with the "serial monitor". This is the text display on the computer used for reading information sent from the Arduino.

<u>Procedure</u>

1. Begin by opening up a new sketch in the Arduino program or by opening up a previous sketch you wish to modify. The lighting levels in every room differ so you will need to discover what analog input values you will read for the particular room you are in. Lab 6 can allow you to do this without writing any code. In a separate sketch window, open up Lab 6 and modify the program to change the

Analog input from “A7” for the potentiometer to “A6” for the photo resistor. Make sure you change all values of “A7” to “A6” throughout the whole sketch or your readings will not be correct.

2. Connect the Arduino to the computer and upload your modified Lab 6. Next connect the power plug from the trainer labeled “POWER” to the appropriate pins on the Arduino with the label facing outward. Also connect the module 5 plug labeled “5” to pins A0-A7 with the label facing outward. Open up the serial monitor and observe the input. If lighting is not sufficient in the room you are in, you may wish to use a flashlight or other direct lighting source to shine directly on the top of the Arduino trainer. Sufficient light should produce a voltage value of greater than 4.25V. If the maximum value in direct light is not 4.25V or greater, you will need a better lighting source. Record the highest voltage produced in direct light and the analog input value displayed in the serial monitor.

3. Next turn off the lights in the room for a moment and observe the voltage value and analog input value of the sensor in a dark room. If the lights cannot be turned off simply cup your hand around the photo resistor located above the potentiometer knob and to the left of LEDs. It’s the squiggly looking thing. Once the values in the serial monitor have become somewhat constant, record these values and turn the lights back on or remove your hand. If multiple people are sharing the same trainer feel free to share your measurements with others, as the values should not differ so long as the trainer stays in the same lighting source.

The next few steps describe what needs to be programmed to accomplish the task proposed for the lab. Variable names are recommended to make students sketches similar and easy to grade. This naming convention can be seen in Table 2. All connections needed for testing the final sketch are also described. Students are reminded of the limitations of several recently introduced commands in an effort to minimize errors during learning. Students are also instructed to test their first section of code to verify that it has been formed appropriately. Steps 4-9 are listed below, with the final step describing exactly how the LEDs should react to various inputs.

> 4. Now that you have your light and dark measurements close lab 6. Begin in the new sketch by creating two integers to monitor the values of the photo sensor and one for the potentiometer. Two will be used to directly record analog measurements, and the other one will be used to map the values for both modes of operation. You may wish to name these integers “PhotoResistor”, “PotValue”, and “x”. Add the “Mode” integer to select between manual and automated lights with a default value of 0. You can also create integers to monitor the digital and analog inputs and outputs used in this lab to make your program easier to read. This is not necessary if you feel confident that you can understand the program simply by referencing the pin numbers within the sketch. If you do wish to name these pins you can use the following for easy reference and naming convention:

Table 2.

Lab 8: Naming Convention

Pin Number	**Integer Name**	**pinMode**
14	Bit0	INPUT
15	Bit1	INPUT
16	Bit2	INPUT
17	Bit3	INPUT
A6	PhotoPin	INPUT
A7	PotPin	INPUT
6	LED1	OUTPUT
5	LED2	OUTPUT

5. In the setup of your program be sure to declare all of the appropriate inputs and outputs listed in step 4. If you want a little extra information you can also begin serial monitoring to keep track of the analog values within your sketch. Begin the loop part of your program by inserting a control statement to set the mode based on weather "0" or "ENT" has been pressed. Remember the binary value of 0 is 1010 and ENT is 1100. You can copy these from any previous keypad sketch so long as the input pins match this one. Make the 0 button set manual controls and the ENT button set automatic controls using a conditional statement.

6. Write another statement for manual controls that reads the analog value of the potentiometer and set this to the variable you chose to store the value. This value must also be assigned to the main control variable (suggested to be "x").

Remember that “x” will be used to set the brightness of the LEDs, and that the analogWrite function uses the values 0-255. This is where the map command comes in handy.

7. You may wish to test your code by this point to verify you have used the map function correctly. The power and module 5 plugs will need to be connected the same as in step 2. The module 3 plug will need to be connected from digital pins 14-21 with the label facing the outside edge of the Arduino, and the module 4 plug will be connected to digital pins 0-7 with the same orientation. If it does not work correctly read the “More Helpful Information” section under the commands and pay attention to the example code and the way it is written.

8. Next create another condition statement for mode 2 or automatic controls. Create a similar section of code to assign values read from the analog photoresistor pin to the variable chosen for these values. When this mode is activated the brightness of the LEDs (“x”) should be set from the photo resistor. In order for the photo resistor to function properly for the room you are in you must constrain them to the values discovered earlier for a light room and a dark room. Map these constrained values to the analog output for the LEDs. Make sure you invert the analog output values so that the LEDs come on when it is dark, rather than when it is light.

9. Once you believe the code is correct, upload it to the Arduino device and plug in the appropriate connections from the Arduino trainer the same as you did before; listed in steps six and two. If everything works correctly the LEDs should

be off in the light and on in the dark. You should be able to switch between manual and automatic controls using the 0 and ENT buttons, and the potentiometer should vary the LEDs brightness.

The following code is provided for instructors to review a working program. There are many comments in the program to detail the function of any statements not previously introduced. Variable names match those suggested in the lab for easy reference. This code in Figure 2 has been tested and works in the exact way described by the assignment.

```
int PhotoResistor = 0; //variable used to assign analog inputs
 int PotValue = 0;     //variable used to assign analog inputs
 int x = 0;            //variable used to map and write analog outputs
 int Mode = 0;   //variable used to select between manual and automatic controls
 const int Bit0 = 14;
 const int Bit1 = 15;
 const  int Bit2 = 16;
 const  int Bit3 = 17;
 int PhotoPin = A6;
 int PotPin = A7;
 int LED1 = 6;
 int LED2 = 5;

 void setup() {
  pinMode(Bit0, INPUT);    //these are the inputs for the keypad
  pinMode(Bit1, INPUT);
  pinMode(Bit2, INPUT);
  pinMode(Bit3, INPUT);
 pinMode(PhotoPin, INPUT); //these are the inputs for the controls

 pinMode(PotPin, INPUT);

 pinMode(LED1, OUTPUT); //these are the analog outputs

 pinMode(LED2, OUTPUT);

 Serial.begin(9600);

}


void loop() {

  if (digitalRead(Bit0) == LOW && digitalRead(Bit1) == HIGH  && digitalRead(Bit2) == LOW &&
digitalRead(Bit3) == HIGH) {             // if 0 is pressed

    Mode = 0;

  }

  if (digitalRead(Bit0) == LOW && digitalRead(Bit1) == LOW  && digitalRead(Bit2) == HIGH &&
digitalRead(Bit3) == HIGH) {             // if enter is pressed

    Mode = 1;

  }

  if (Mode == 0) {

    PotValue = analogRead(PotPin);

    x = PotValue;

    x = map(x, 0, 1023, 0, 255);

    analogWrite(LED1, x);

    analogWrite(LED2, x);

  }

  if (Mode == 1) {

    PhotoResistor = analogRead(PhotoPin);

    x = PhotoPin;

    x = constrain(PhotoResistor, 671, 1012);

    x = map(x, 671, 1012, 255, 0);

    analogWrite(LED1, x);

    analogWrite(LED2, x);
  }
    Serial.println(x);
}
```

Figure 2. Lab 8 Code

Lab 10: Interfacing With a Matrix Keypad

Lab 10 was chosen because it instructs students on how to interface the keypad on the second trainer with the LCD screen. These two devices are used in all other labs, and understanding how to use these devices in crucial to completion of labs 10-14. It begins by explaining the new C commands used including the “char” or character variable and the byte. The equipment list is also displayed as with the previous lab.

Lab 10: Interfacing With a Matrix Keypad

Purpose

In this lab you will learn how to interface with the matrix keypad on the second trainer. A matrix keypad is a type of keypad that closes the circuit between two pins when a button is pressed rather than writing a pin HIGH as with the previous keypad. The keypad library will be included in this sketch, which sends pulses through the input pins and reads which output pins go HIGH to determine which key has been pressed. You will also learn about the varable “char” or character. This data type allows a number, letter, or symbol to be used in conditional statement and commands. You will also learn the byte data type. A byte is 8 binary bits that are used to store data. ASCII uses one byte of data for each character. Using the new data types and a few other library specific keypad commands you will be able to begin more complicated programming using the keypad, rather than a serial monitor, to control what prints.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Trainer II

The commands in this section are then introduce in Table 3. Several of them are specific to the keypad library.

Table 3.

Lab 10: Commands and Description Section

Command	**Description**

char *NAME = #* or ‘*CHARICTER*’
char *NAME*[#] ={‘*CHARICTER*’}
char *NAME*[#][#] = {
{‘*CHARICTER*’}
{‘*CHARICTER*’}
}
	The first command allows for one character to be specified by a particular name, much like integer. The ASCII number must be used to specify the character, or a single character can be placed between two apostrophes such as ‘A’. The second command is used for a single

	line of characters where the number specifies the number of characters. The third specifies an array of characters where the first is the number of rows, and the second is the number of columns
byte	8 bits of data used to store a single ASCII character in its binary form.
key	A library specific command used in the program to refer to a specific key
NAME.getKey()	A library specific command that is a variation of the C command getcar(). The name is whatever you choose to name the keypad, and signifies that the command will get the key from that device. The .getKey() command reads the character specified for that key and "gets" the one you press to use in your program.
Keypad	This command is used to declare the name of the keypad for use with the .getKey() command.
makeKeymap(*Charicters), RowPins, ColumnPins, CharicterRows, CharicterColumns*)	The *Charicters* are the characters specified for the keypad array. The *RowPins* in this lab will be 30, 32, 34, 36, and the *ColumnPins* will be

	38, 40, 42, 44, 46, 48, 50, 52. The *CharicterRows* and *CharicterColums* are the number of rows and columns in the character array for the keypad.

The Keypad.h library is not included with the Arduino software and must be downloaded. This library is available free through the Arduino website, and students are instructed how do get this library and include it in their program. This further demonstrates the open source nature of the Arduino and its programming environment and allows students to take full advantage of this technology in the future, should they choose to continue using the Arduino. A more in-depth explanation of the "char" variable and how to use it to interface with the keypad is also provided in Table 4. Figure 3 shows the character array for the keypad used in this lab while Table 5 shows the actual keypad buttons.

More on Keypad.h and char

Keypad.h is not a preloaded library file. In order to get the file you must download it from http://arduino.cc/playground/Code/Keypad#Download. Once you download the file you must extract the file and move it to the proper location before opening the Arduino sketch program. Extract the file the same way you extracted the Arduino software. Then copy the file into the "libraries" folder within the "arduino-1.0" folder. Once this is done you can open the sketch program and use this library with the "#include<>" command and you will have access to several new examples.

This library file uses characters in unique ways. For the purpose of this sketch you will be specifying a keypad array that uses a single character for each key. First, here are some examples of how to use char. Note the word “Character” is what I have chosen to use as a name.

Table 4.

Lab 10: Character Print Examples

Command Entered	**Output on the LCD**

char Character = 'A';
lcd.print(Character);
	
	A

char Character[3] = {'A', 'B', 'C'};
lcd.print(Character[1]);
lcd.print(Character[2]);
lcd.print(Character[0]);
	
	B
	C
	A

char Character[2][3] = {
  {'A', '1', 'B'},
  {'2', '3', 'C'},
  };
  lcd.print(Character[0][0]);
  lcd.print(Character[0][2]);
  lcd.print(Character[1][2]);
	
	A
	B
	C

The keypad on the Arduino trainer is a 4x8 keypad. In other words there are four columns and eight rows in the internal circuitry. You will need to create a character array similar to the final example for your keypad to display each number or mathematical function pressed. Later you will use these characters to perform various math functions. The character array for the keypad looks like this:

```
{'^','/','9','8','7',' ','=','Q'},

{'T','X','6','5','4',' ','0',' '},

{'C','-','3','2','1',' ','A','N'},

 {'S','+',' ',' ','.','c',' ',' '},
```

Figure 3. Character Array

Each key can only hold one character, so functions such as “Sin” must be represented by a single letter “S”. Some of the characters have been omitted and will be added in later labs. Table 5 shows how each letter corresponds with the button pressed. Functions with longer names have parenthesis around the brackets used to display those characters.

Table 5.

Lab 10: Keypad Buttons

Cal(c)				+	(S)in
(L)ight	1	2	3	-	(C)os
(O)hm	4	5	6	X	(T)an
(D)ist.	7	8	9	/	^
(A)lpha	Clear/(N)o	0	Enter/Yes		S(Q)RT

The procedure for this lab is broken up to make sure students are creating the code correctly. First students are introduced to the keypad library in the first few steps and told in detail how to use the newly introduce commands to interface with the keypad. Suggested variable names are used, and students are instructed to look at an example code provided with the keypad library to compare their code to. Often times seeing a similar working program, and comparing it to one created will clear up any syntax errors and improve understanding of the new commands. The students are instructed to first print the characters from the keypad to the serial monitor to verify this portion of the code works before attempting to form conditional statements using the characters. This was done so students know exactly where the error in their program occurs should one arise.

Procedure

1. Open up your previous sketch and copy all the code used to initiate the LCD screen. These are the "#include<>" function, the LCD pins, the digital output used to power the LCD and the "lcd.begin()" commands. Also include the new keypad library "Keypad.h".

2. Before the setup of your program include two constant bytes. Declare one for "Rows" and one for "Columns". These will be used to create the character array for the 4x8 keypad. Make the character array mentioned in the Commands and Description section. Name the character something easily recognizable like "keys".

3. Next you will need a number array to specify the pins used by the keypad. This will be an array of bytes and is entered "byte rowPins[Rows] = {30, 32, 34, 36};". These pins are also mentioned in the Commands and Description section. The name "Rows" in the brackets specifies that the number of rows in the keypad is 4 from your previously declared byte. You will also need to add one for "columnPins".

4. To map the appropriate Arduino pins to the character array you created name your keypad using the Keypad declaration and specify the row and column pins. If you used all suggested variable names the code should appear as "Keypad keypad = Keypad(makeKeymap(keys), rowPins, columnPins, Rows, Columns);". If any of this is confusing you may wish to consult one of the examples that came with the Keypad to see how your code should appear. One good example is "CustomKeypad" that is used to create a 4x4 keypad.

5. Now within the loop of your program you will need to assign one more character to the key pressed using the "getKey()" function introduced. To verify that all of your declarations and arrays have been created correctly, begin serial communication and print the values of the keys to the serial monitor with a short delay to prevent the monitor from scrolling too quickly. Connect the module 7

plug on the keypad to digital pins 52-30 with the wire strip hanging over the outside edge of the Arduino. Verify that each printed character matches the key pressed.

Next students are told to use the knowledge gained from the last lab, where they interfaced with the LCD screen, to print the various mathematical functions. This lab provides the basic code for the next lab where they will create a calculator. In order to ensure this code can be used effectively students are told to use separate conditional statements to print the mathematical functions. Students are then instructed on how to connect the devices and what output they should expect on the screen for a successful program.

6. Once you have gotten the proper keys printing, you will need to print these to the LCD screen to properly display them. The character "c" for Cal(c)ulator will later be used to initialize a calculator function. Using the knowledge gained in the last lab, create a conditional statement that tests to see if the key received is "c". If this character is received, clear the LCD screen to get rid of any unwanted characters, print the word "Calculator" in the top left corner, and clear it again after 2 seconds. The character "N" is used to represent the "Clear/(N)o" button. Make this button simply clear the LCD.

7. Use conditional statements to print all numbers to the screen. Try using the Boolean operator "||" or to do this in a single statement. In separate statements you will need to print the mathematical operators "+, -, /, ^, =" and "X" used for multiplication, as these will be used for more complex functions in the future. Also print the decimal, (".") for using decimal numbers in a separate statement.

8. For the tragicomic functions “S” for sin, “C” for cosine and “T” for tangent, you will need to print “sin(, cos(, tan(“. Rather than simply displaying the character.

9. Once you believe your code is correct, connect the module 6 plug in the second trainer to the GND-41 digital pins as you did in the previous lab. Also connect the keypad to digital pins 52-30 as you did in step 5. These two should overlap slightly. Connect the USB cable through the battery door of the second trainer and close the box. If everything has been performed correctly, you should be able to print any mathematical operation or number. The clear button should clear the screen, and pressing the Calc button should display the word Calculator.

As for every lab, a tested working code will be provided to the instructors to clear up any confusion. The code compiled for this lab can be seen in figure 4. This code is also heavily commented to explain the purpose of each section. It is well organized with indentations and spaces between different sections of code to be easily read.

```
#include <Keypad.h>
#include <LiquidCrystal.h>

LiquidCrystal lcd(51, 49, 47, 45, 43, 41);

const byte Rows = 4; //four rows
const byte Columns = 8; //eight columns

char keys[Rows][Columns] = {
 {'^','/','9','8','7',' ','=','Q'},

 {'T','X','6','5','4',' ','0',' '},

 {'C','-','3','2','1',' ','A','N'},

 {'S','+',' ',' ','.','c',' ',' '},

}; //This is the character array for the keypad


byte rowPins[Rows] = {30, 32, 34, 36}; //Connect to the row pin outs of the keypad

byte columnPins[Columns] = {38, 40, 42, 44, 46, 48, 50, 52}; //Connect to the column pin outs of
the keypad


Keypad keypad = Keypad( makeKeymap(keys), rowPins, columnPins, Rows, Columns); // Maps
the character array to the keys


void setup(){

 pinMode(53, OUTPUT);

 digitalWrite(53, HIGH);    // sends power to the LCD

 delay(50);                 // Allows LCD some time to warm up

 lcd.begin(24,2);           // Begins LCD library

}


void loop(){

 char key = keypad.getKey();   // Defines "key" as a character and sets it to the character pressed


 if (key == 'c') {           // This is the Calculator button

   lcd.clear();             //Clears anything to remove characters

   lcd.print("Calculatior");

   delay(2000);             // Displays Calculator and waits 2 seconds

   lcd.clear();             // Clears again for input

 }
```

```
  if (key == 'N') {       //This is the Clear key on the keypad and should clear the LCD

    lcd.clear();

  }


  if (key == '0' || key == '1' || key == '2' || key == '3' || key == '4' || key == '5' || key == '6' ||
key == '7' || key == '8' || key == '9') {

   lcd.print(key);           // This prints the key if a number is pressed

  }




  //The next few keys are written separately to accommodate for future calculations. These are
basic math functions.

  if (key == '+') {

    lcd.print(key);

  }


  if (key == '-') {

    lcd.print(key);

  }


  if (key == 'X') {

    lcd.print(key);

  }


  if (key == '/') {

    lcd.print(key);

  }


 //These are the more complex math functions

  if (key == '^') {
        lcd.print(key);
      }

      if (key == 'S') {
        lcd.print("sin(");
      }

      if (key == 'C') {
        lcd.print("cos(");
      }

      if (key == 'T') {
        lcd.print("tan(");
      }

      if (key == 'Q') {
        lcd.print("sqrt(");
      }

      //These will be used to perform calculations
      if (key == '=') {
        lcd.print("=");
      }
       if (key == '.') {
        lcd.print(".");
      }
    }
```

Figure 4. Lab 10 Code

Both of these labs provide easy reference for all commands used and explain other more complicated concepts in a way students will be able to comprehend. The information given on the keypad and analog lab allow students to have a basic understanding of the electronics used without going into as much depth as a technology or electronics lab. This allows the programming, or interface with these devices, to be the main focus of the labs. Students are given clear step by step instruction on how to form the code leaving enough room for them to form their own individual programs. All other labs introduce concepts in similar ways, with new concepts being explained in depth using examples. The labs require students to use their knowledge of previous concepts to finish the assignment. The example code provided will be similar to any sketch created for the same assignment. Variable names and basic structure of the code may differ from student to student; however, all programs will serve the same function as the examples provided.

CHAPTER 4

DESIGN CONSIDERATIONS

The Arduino Mega

“Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments” (Arduino, 2012, p. 1). The Arduino Mega 2560, and the associated software was chosen as the key technology for the development of the created project for several reasons. The open source nature of Arduino, the programming language used, the interface, and applications all make it the ideal tool in meeting the previously defined STEM educational need. The hardware is reasonably priced and the software is free making the Arduino a very affordable teaching tool.

The benefit of the programming the Arduino with open source software is that all students have free access to the software. Students and schools will pay nothing to use the Arduino programming environment. Students can download the software on home computers to use personal time to advance their work, or simply play around. There are a variety of programs and libraries available through the Arduino website to further education and expand the capabilities of the device. The software is also cross-platform, meaning it is available for most major operating systems. The programming environment is simple and straightforward making it an ideal choice for beginners. The example codes and libraries made available through the site are free, easy to understand, and tested working by the Arduino users.

The open source hardware of Arduino boards means that preassembled boards can be purchased for well under $50. This also means the Arduino can be built by hand and expanded upon to improve its capabilities. “The Arduino is based on Atmel’s ATMEGA8 and ATMEGA168 microcontrollers. The plans for the modules are published under a Creative commons license, so experienced circuit designers can make their own version of the module, extending it and improving it” (Arduino, 2012, p. 2). The Arduino Mega 2560 was chosen for this project because of its abundance of digital inputs and outputs and its operating characteristics. Table 6 contains a summary of the characteristics of the Arduino Mega.

Arduino, Arduino Mega 2560 Summary (2012, p. 2):

Table 6.

Arduino Mega 2560 Summary

Summary	
Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA

DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by boot loader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

The Arduino Mega 2560 can be powered either by a standard USB printer cable or a 2.1mm center-positive power jack. The power jack can either be connected to a AC to DC adapter or battery pack. The external supply can be anywhere from 6-20V, although 7-12V is recommended for optimal performance. The “Vin” pin of the Arduino can also accept a stable 5 V power signal for operation. The Mega 2560 also has two voltage output pins consisting of a 5V and a 3.3V regulated output. The 5V I/O pins make the hardware ideal for operating with a variety of low power digital and analog devices, and the device even has sufficient current capabilities to drive small electric servo or DC motors. Figure 5 shows a picture of the Arduino Mega 2560.

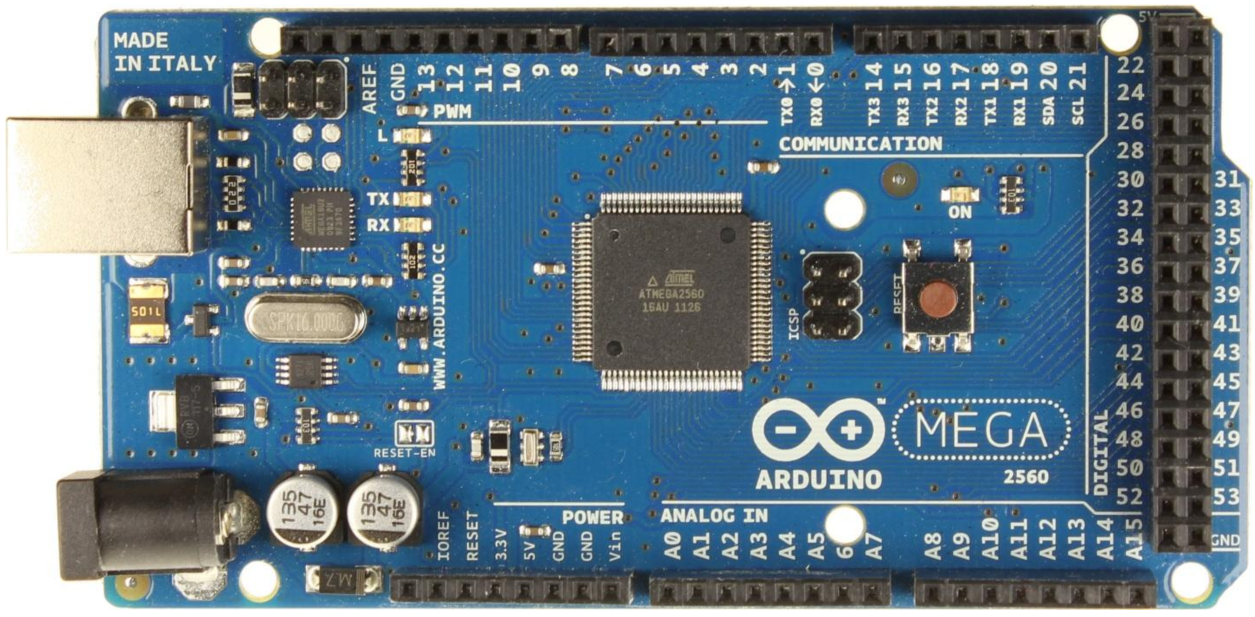


Figure 5. Arduino Mega 2560 (Arduino, Arduino Mega 2560, 2012)

Other Electronic Components and Devices

The electronics in the trainers were chosen for their ability to demonstrate the concepts introduced for the lab assignments. They were also chosen for their operating characteristics and compatibility with the voltage and current capabilities of the Arduino Mega 2560. The fact that all exposed electronic elements operate on a low current 5V power supply means the risk of electrocution or shock to students or teachers is nearly nonexistent. The more rugged components featured in the first of the Arduino trainers develop have the ability to withstand the abuse common to newly educated students exposed to their first electrical devices.

All electronic components and devices featured in the trainer including resistor configurations operate at or below a 40 mA current. They are all powered either by the 5V internal voltage output pin on the Arduino or directly from an analog or digital pin. Ground connections were made to the common ground pins of the Arduino. All resistors are rated for up to 500 mW or greater and are either metal film or carbon film through-hole resistors. Individual datasheets for the various components used in the trainer can be seen in Appendix C. The datasheets for the resistors have been omitted because their relevant characteristics have already been described. The datasheet for the matrix keypad has also been omitted because it could not be found.

All electronic components used to build the prototypes of the trainers are listed in Table 7 below. The majority of the components were purchased from Newark, the online electronic components store at http://www.newark.com/. A few components were purchased from Jameco Electronics at http://www.jameco.com/, and Robot Shop at http://www.robotshop.com/. Some of the components used in the trainers were items from the department of Business and Technology at ETSU. Items labeled with a "*" were identical in function to items used in the prototype trainers but differed by model. These were items from ETSU. The price for the Matrix Keypad denoted with "**" is for a similar item with fewer keys than the actual keypad used in the trainer. No keypad with the same functionality and number of keys could be found on any of the three sites listed. The keypad used was one of many found at ETSU and contained no model number that could be matched to the product in a web search.

Table 7.

Electronics Parts List

Component	Quantity	Unit Price	Total	Price From
5mm Red LED	6	$0.21	$1.25	Newark
5mm Green LED	2	$0.21	$0.42	Newark
5mm Yellow LED	2	$0.23	$0.46	Newark
Ultra-bright White LED	2	$0.69	$1.38	Newark
10K Ohm Rotary Pot	1	$1.41	$1.41	Newark
10k Tuner Pot	1	$0.75	$0.75	Jameco
SPST Pushbutton Switch	15	$1.44	$21.60	Newark
100 Ohm Resistor	1	$0.01	$0.01	Newark*
330 Ohm Resistor	20	$0.10	$1.94	Newark
1K Ohm Resistor	1	$0.09	$0.09	Newark*
10K Ohm Resistor	1	$0.04	$0.04	Newark*
100k Ohm Resistor	3	$0.04	$0.11	Newark*
1M Ohm Photoresistor	2	$1.79	$3.58	Jameco
Black Safety Socket	1	$3.59	$3.59	Newark
Red Safety Socket	1	$3.59	$3.59	Newark
Diode	15	$0.05	$0.75	Jameco
Arduino Mega 2560	1	$47.91	$47.91	Newark
24x2 LCD Display	1	$21.92	$21.92	Newark*
Ultrasonic Range Finder	1	$15.00	$15.00	RobotShop
Matrix Keypad	1	$29.99	$29.99	Newark**
Totals:			$155.78	

Two sets of red, yellow, and green LEDs were arranged in a stoplight configuration, connected in series with 330 ohm current limiting resistors to ground. These can be seen in Figure 5 connected to I/O pins 09-16 which represent the Module 1 plug of the trainer. Figure 5 and all other schematics were created using Multisim 11.0. Another series of 4 LEDs used to represent binary numbers have been connected in the same configuration. These are LEDs 07-10 of the schematic. Four single pole single throw (SPST) momentary buttons located below these LEDs are connected from the 5V power supply in series with 330 ohm resistors connected to ground to produce a readable current for the Arduino. I/O pins 01-08 are the Module 2 plug, and connect these components to the Arduino.

The 10K ohm rotary potentiometer is connected from power to ground and sends the wiper to I/O pin 25 for the Arduino. The 1M ohm photoresistor is in a voltage divider configuration with a 100k ohm resistor to produce a varying voltage to be read by the Arduino. The output of the voltage divider is connected to I/O 26. Together these two pins make up the Module 4 plug that is the plug for the only devices on the first trainer that send an analog signal to the Arduino. Finally LEDs 11 and 12 represent the ultra-bright LEDs and are connected to I/O pins 18 and 19 of the module 5 plug. These LED's are the only LEDs that are connected with the 330 ohm current limiting resistor going to ground. This is so the full 5V signal from the Arduino can be felt by these LEDs allowing maximum illumination. This configuration can also be seen in Figure 6.

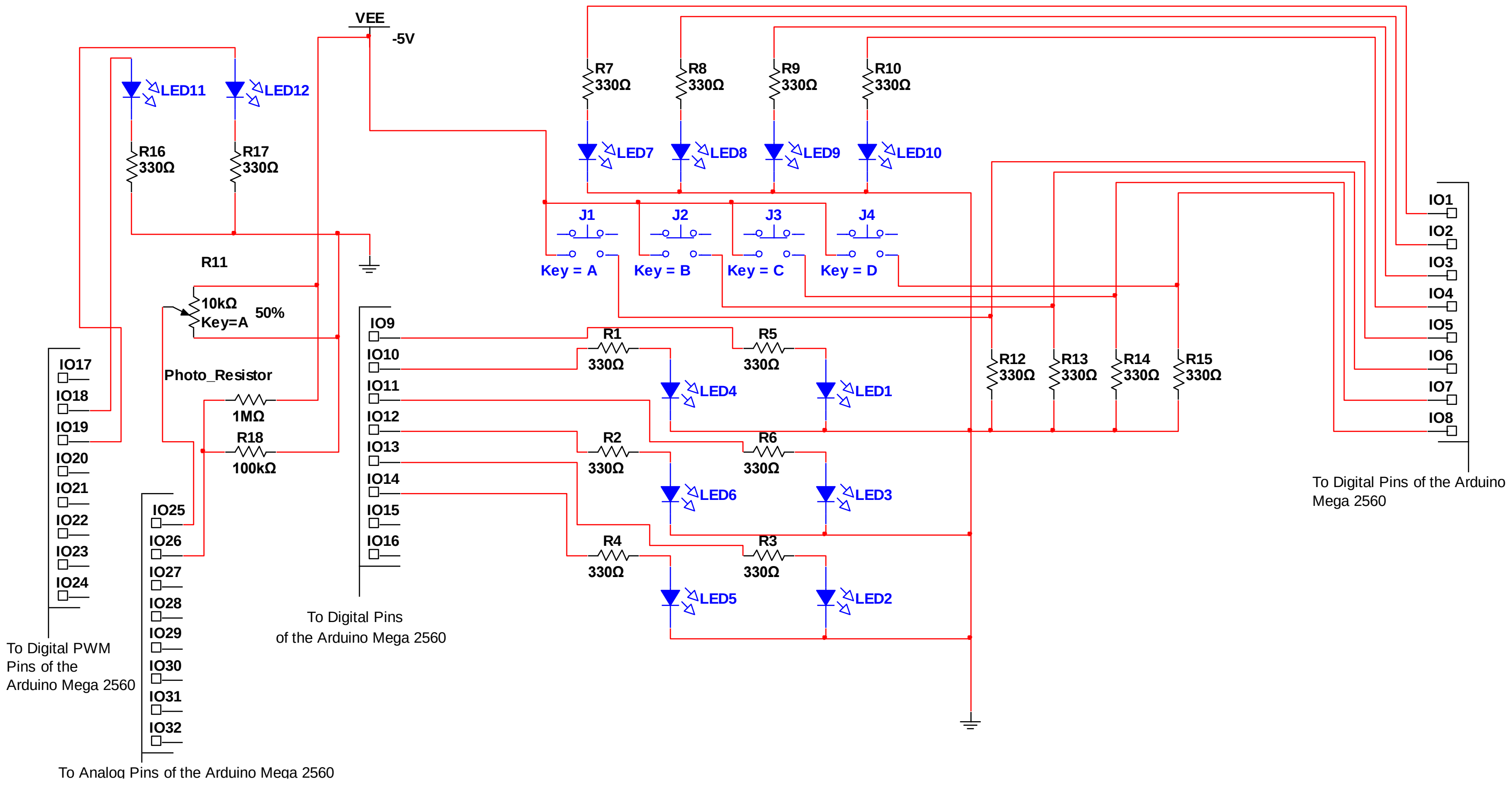


Figure 6. Trainer 1 Schematic 1

Figure 7 shows the schematic of the binary keypad created for the first Arduino trainer. This was placed in a separate schematic to make it more distinguishable from the other components in the first trainer, and because of the number of components to create a properly functioning keypad. Eleven SPST momentary push buttons were used with 10 of them forming the numbers 0-9 and 1 being used as an “Enter” Button. Each of these buttons is connected to a series of diodes to produce their associated binary number on I/O pins 1-4 for the module 3 plug with I/O 1 being the least significant bit. The diodes were used to prevent feedback from one button being pressed and falsely supplying voltage to another. Keys 1, 2, 4, and 8 were not connected with diodes as these are represented by single bytes and will not be affected by feedback. Fifteen diodes were used to create the binary output for the circuit. In order to create a proper reading for the 0 button this was wired to I/O pins 2 and 4 forming the binary number 10. Similarly the Enter button is represented by the binary value of 12. Four 330 ohm resistors were connected to I/O pins to ground to create a readable current for the Arduino to process.

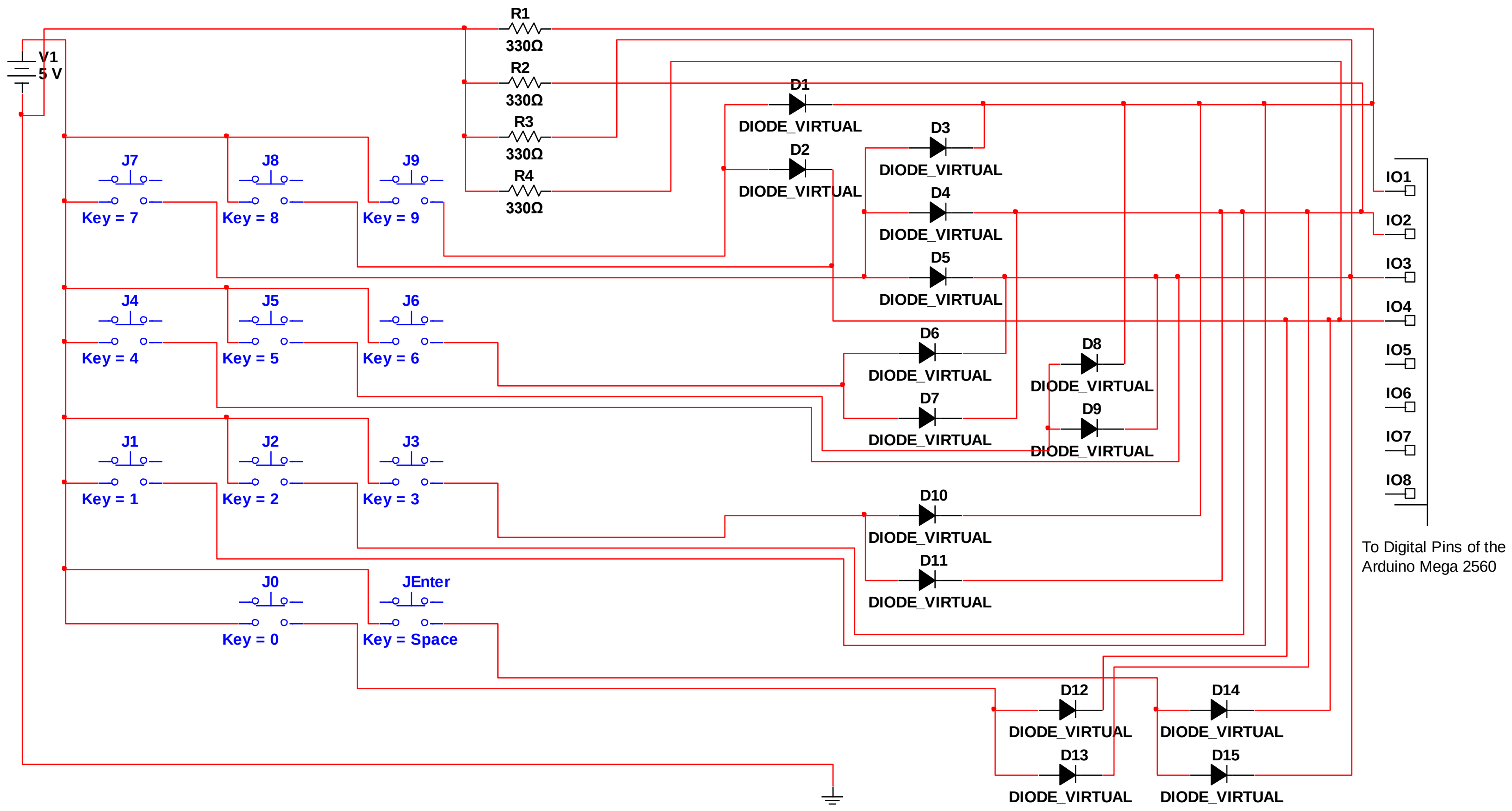


Figure 7. Trainer 1 Schematic 2

The second trainer’s schematic shows the internal connections made in Figure 8. The module 1 plug consists of I/O pins 1-8 and connects the LCD screen pins necessary for four bit operation to the Arduino. I/O pin 1 is connected to one of the common grounds of the Arduino and I/0 pin 2 provides power to the device. A 10K ohm tuner potentiometer is connected from power to ground with the wiper going to the contrast adjustment for the LCD. This potentiometer is inside the second trainer, as it does not need adjustment after initial tuning has been done.

The I/O pins named Red and Black represent the safety sockets used to connect leads for measuring resistance. The 100, 1k, 10k, and 100k ohm resistors will be used in a voltage divider configuration with power coming from only one I/O pin at a time. A photoresistor voltage divider circuit is connected in the same configuration as previously introduced; with I/O pin 15 providing power and I/O pin 14 receiving the analog signal. These make up the module 3 plug for the second trainer. The “UltraSonic” I/O pins represent what their names suggest and are connected to the module 4 plug using I/O pins 17-24. The I/O pin 18 will once again be connected to one of the common grounds on the Arduino with pin 19 supplying power and 20 receiving the signal.

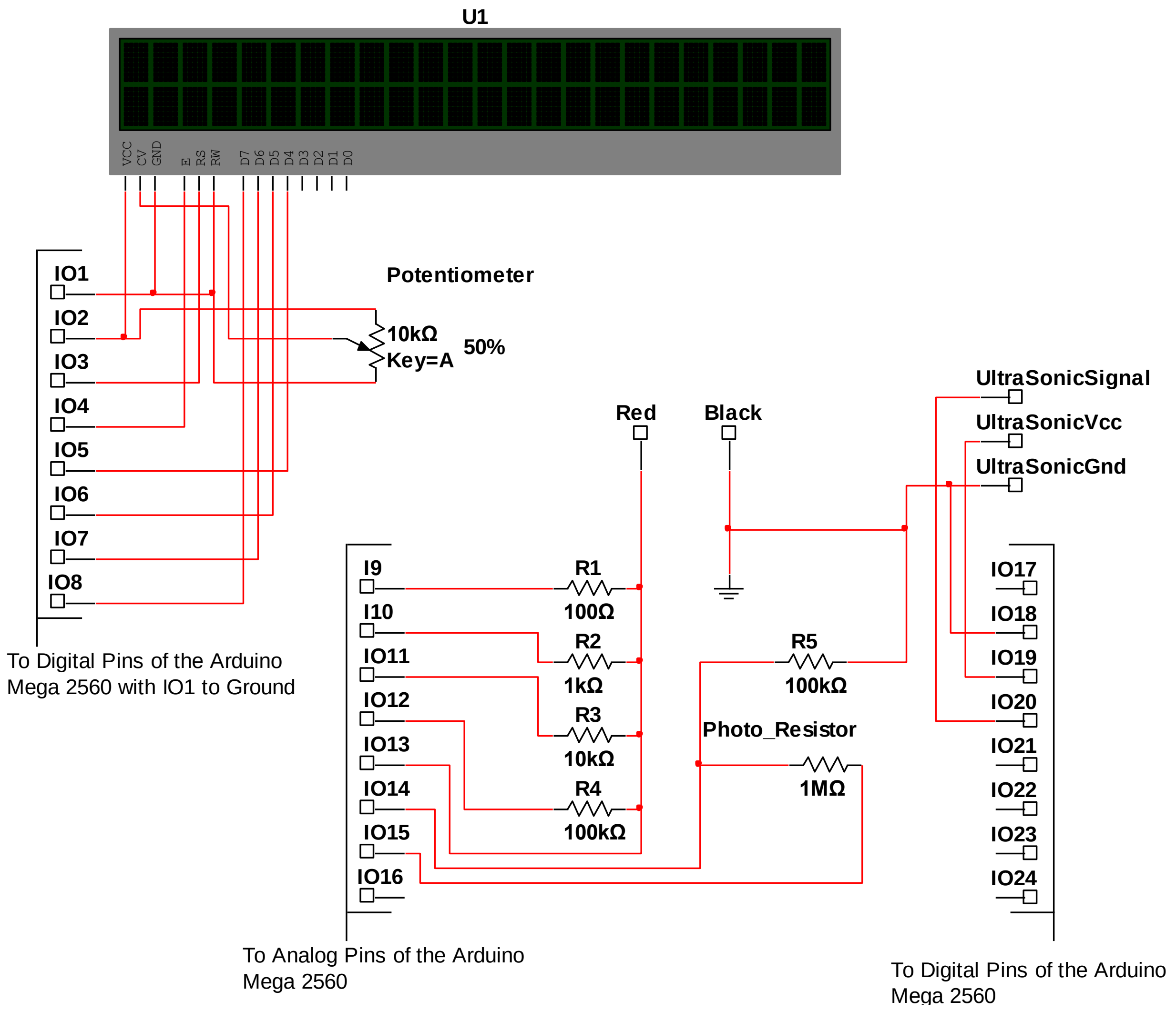


Figure 8. Trainer 2

The Module 2 plug for the keypad has been omitted from the schematic drawings. The keypad consists of 12 I/O pins that are connected directly to a single row 12 pin headers that make up the module 2 plug. The datasheet for the keypad could not be found so its operation was determined by taking apart the device and using a digital multi-meter to test for conductivity through the pins. The device has five rows and six columns of buttons with an "Enter/Yes" button taking up two key spaces. The keys 0-9, "Clear/No", and "Alpha" were labeled and left as they appeared. The "S.F." key and other blank keys were labeled for use with the labs. It was determined that the keypad functions as a 4x8 matrix keypad. A Matrix keypad closes the circuit between the row and column pins based on the button pressed, with each button creating a different closed circuit. Table 8 shows the closed circuit made when each button is pressed with pin 1 being the pin farthest to the left of the plug. Table 9 shows the labels of the keys. In both cases a single box on the table represents a physical key on the keypad.

Table 8.

Pin Connections

4+10	4+9	4+8	4+7	4+6	4+5
3+10	3+9	3+8	3+7	3+6	3+5
2+10	2+9	2+8	2+7	2+6	2+5
1+10	1+9	1+8	1+7	1+6	1+5
3+11	3+12	2+11	1+11		1+12

Table 9.

Key Labels

Calc				+	Sin
Light	1	2	3	-	Cos
Ohm	4	5	6	X	Tan
Dist.	7	8	9	/	^
Alpha	Clear/No	0	Enter/Yes		SQRT

Materials and Construction

Many of the material used to create the trainers were items already available in the Business and Technology department of ETSU. Others more critical to the operation of the trainer were purchased from electronics stores. The construction of the frame and physical design of the first prototype trainer was done by graduate students Matt Crum and Benjamin McMurry during the fall semester of 2011. All design information including dimensions, construction techniques, and dimensional drawings for the first trainer were provided by Crum and McMurry in a report prepared for ENTC 5900. The second prototype trainer was designed and constructed by the author in the following semesters.

The rectangular body of the first trainer is constructed primarily from wood. This material was chosen by Crum and McMurry because it is an easy to work with material that is somewhat light and provides insulation for the internal electric components, as well as its availability. The trainer consists of a body casing and faceplate fashioned with aluminum housings for onboard LEDs. The wood for the lower body casing was cut using a radial arm saw and table saw. The box measures 12 in. wide, 10.25 in. in length, and 6 in. high. The box also has an internal wooden divider to separate the electronics from a 1 in. diameter hole where the plugs for the trainer exit. The body was fashioned together using finishing nails.

The faceplate of the trainer was designed in AutoCAD 2007 by Crum and McMurry and cut out of .25 in. thick MDF on a CNC router table using a .25 in two flute end mill to create the holes for the buttons and the potentiometer and the outer perimeter of the faceplate. The screw holes and led holes were cut out using a .125 in drill bit on the CNC router. An excerpt from their report outlines the construction of the aluminum led faceplate housings:

Crum & McMurray (2012)

> The two led faceplates were cut using a standard milling machine and a .375 two flute end mill. The two pieces were first machined square to a size roughly larger than the placement of the LEDs. They were then surface machined mainly for aesthetic reasons, as surface machining aluminum leaves a desirable pattern in the surface of the material. The next step was to simply lay the led faceplate onto the box faceplate and transpose the location of the LEDs onto the aluminum using a punch. Once this was done clearance holes were drilled into the aluminum for the LEDs to be installed into. The LED's bodies are slightly tapered or larger on the end where the wires exited the plastic than the top. This allowed for the majority of the LED's body to pass through the aluminum, but not all the way through. Two holes were then drilled and tapped into the aluminum to accept a 10-24 socket head screw, and clearance holes were drilled into the MDF for the same 10-24 socket head screws. These two screws are to hold the aluminum faceplates in place and secure the LEDs to the faceplate.

The dimensional drawing of the faceplate provided by Crum and McMurray can be seen in Appendix D. A revision was later made to the faceplate of the box to include a photoresistor. Two small holes were added to the faceplate using a drill press fitted with a 1/8 in drill bit. The photo resistor was placed 1.5 in away from the top edge of the aluminum led housing.

The internal electrical components were soldered to a breadboard connecting to the various component mounted in the faceplate. Wires were feed from the proto board through the 1 in. hole in the side of the box and connected to 8 pin stackable headers using a hot glue gun to hold them firmly in place and provide insulation to the electric connections. The headers were then insulated with electrical tape and labeled with their respective number. All components were inserted into their respective place in the faceplate, and a metal knob was secured to the potentiometer. Internal wiring connections were made using silicon filled 3 terminal connectors which provide insulation and require less time to make connections then conventional soldering techniques. The box was painted with blue and yellow pant to provide a more finished look. A picture of the first prototype trainer can be seen in Figure 9 below.

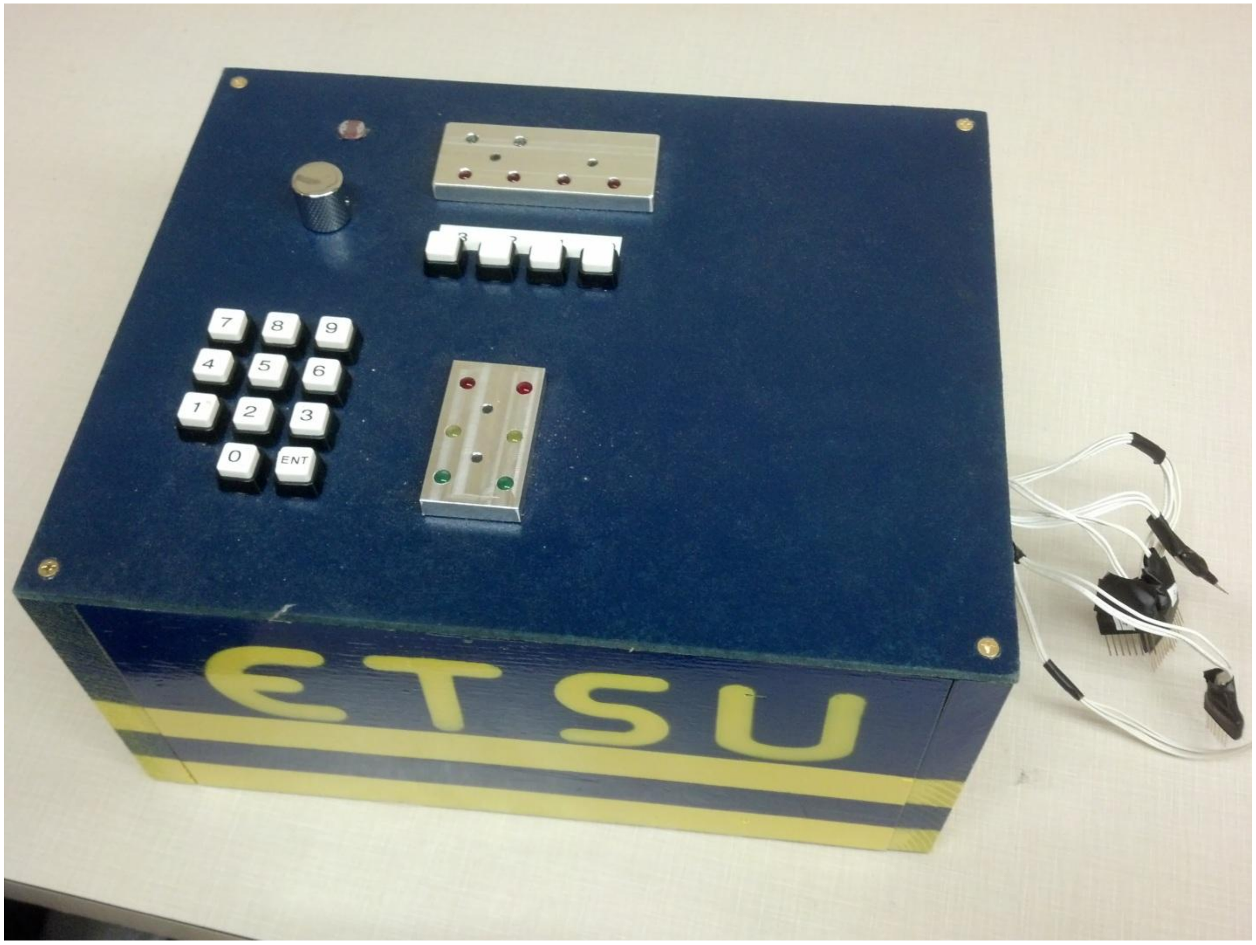


Figure 9. Prototype Trainer 1

The second prototype trainer was created under the guidance of Mr. William Hemphill. It was housed in a prefabricated project box produced by Hammond Manufacturing. The project box is formed from ABS plastic and measures 8.661 in. wide, 8.12 in. long and 1.575 in. thick. This project box was chosen for its lightweight relatively small size allowing the second trainer to be portable. The project box also features a sliding battery door section that provides access to internal components without having to open the box. Several project boxes were considered, but this one allowed for the proper amount of clearance between the internal and external hardware and had sufficient size to allow for all electronic components to be mounted appropriately.

Once again AutoCAD was used to form the dimensional drawing for the cutouts that were to be made to seat various external components. The second prototype trainer AutoCAD was also used to model the components edges to verify that they would have proper placement in the project box without interfering with the internal screw mounts. The project box also contained eight small plastic stubs that needed to be avoided when cutting out the holes for the electronics. The project box outside can be seen in Figure 10 and the inside can be seen in Figure 11. Several revisions were made to the original drawing to ensure all components would be placed properly.

Figure 10. Project Box Outside

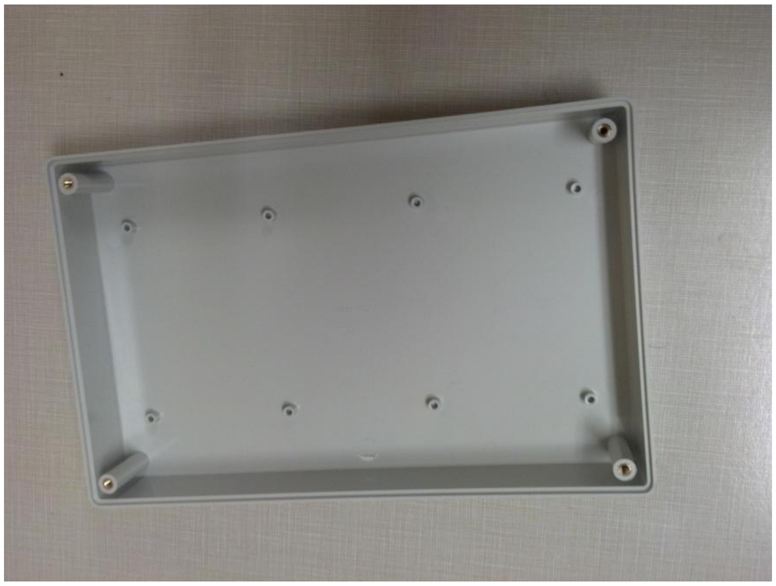


Figure 11. Project Box Inside

An insert was created with the help of Mr. Hemphill to allow the project box to directly rest on a surface and ensure pressure was applied to the inside of the box. This reduced the likelihood of the box chattering when the holes were milled out using a CNC router. This also prevented chipping and other damage when milling out the project box. The insert was model in AutoCAD and the tool path was also created. A piece of 0.75 in. thick MDF board was used as the primary material to rest firmly against the interior of the box. All cuts and holes were made using a 0.25 in. drill bit on the CNC router. A second piece of 0.25 in. thick MDF board was cut to the inner dimensions of the box without the holes made to compensate for the plastic stubs. Both of these were placed inside the box allowing the MDF to have a small clearance to support the full weight of the project box allowing pressure to be applied to the front, which was to become the faceplate for inserting the various components. The insert created can be seen in Figure 12.

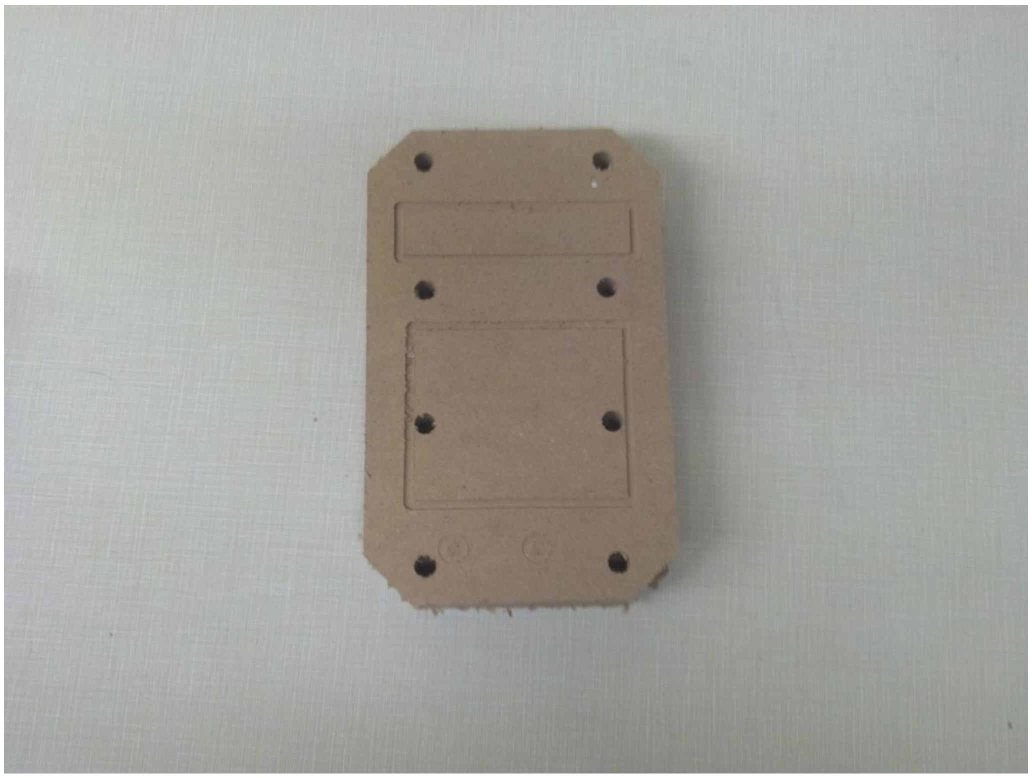


Figure 12. MDF Support Insert

A tool path was created in AutoCAD for a 0.125 in. drill bit to make the proper cuts to the faceplate. The inserts were used and the CNC router made the specified cuts to the project box. Two holes were made in the front of the box to the diameter of the Ultrasonic range finder using a standard drill press. Below is Figure 13, a picture of the box after all cuts were made.

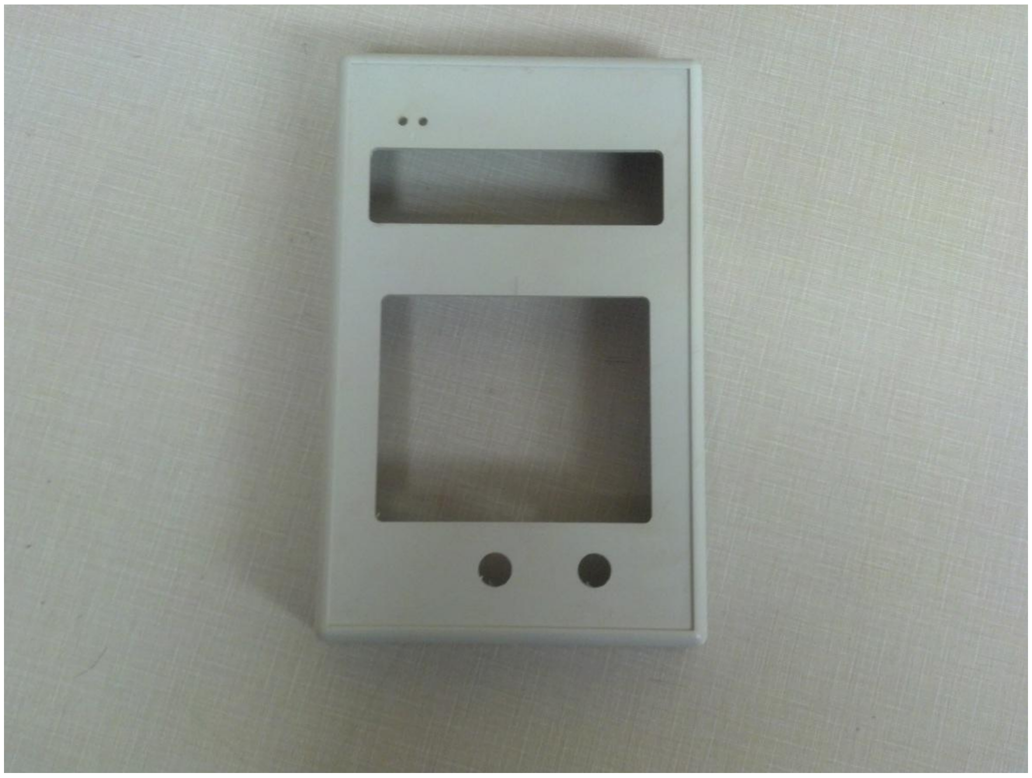


Figure 13. Project Box After Milling

This project box is closed and secured by using the hardware provided with the box. External electrical components were secured using small sized screws for the LCD screen, ultrasonic range finder, and keypad. The safety sockets were secured using their provided hardware. The photoresistor is held in place using a small amount of hot glue to secure it. Very few resistors were needed to ensure proper operation of this equipment, and these resistors were soldered in line with the wiring to the various components. The same 8 pin stackable headers were used to secure the wires and provide a simple plug connection to the Arduino. The keypad used a single row 12 pin header connected to the Arduino. All wires were held in place using hot glue, then insulated with electrical tape and labeled with their respective number. These headers were bent at a 90 degree angle to ensure clearance for the limited space within the project box. A picture of the second prototype trainer can be seen in Figure 14.

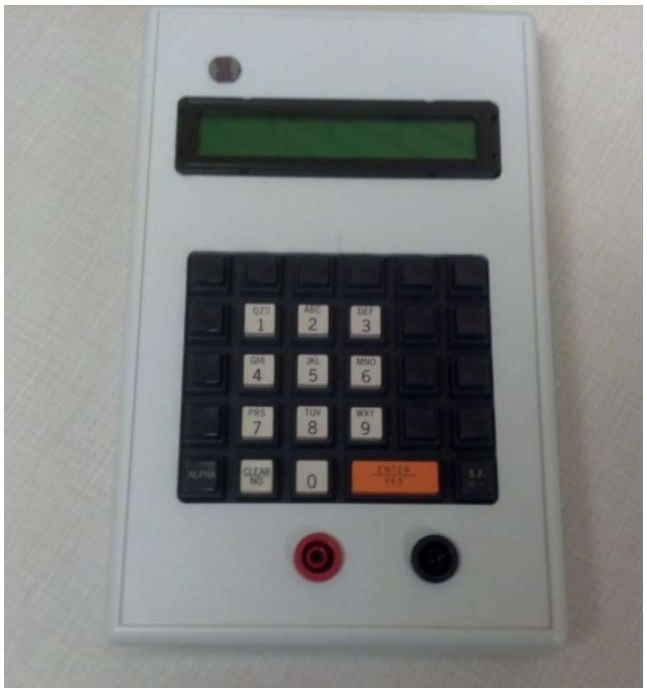


Figure 14. Prototype Trainer 2

Table 10 shows the price of the various construction materials used. Items denoted with a “*” signify that the entire quantity of the item was not used and the amount used was estimated. Items such as wire and three terminal connections could only be purchased in certain amounts and therefore the price included was for the amount purchased. Items denoted with “**” are items that were simply found available for use at ETSU and the price was estimated from similar items. The price of screws, nails, and hot glue was omitted as very few of these were used and it would be difficult to quantify the price of four small wood screws (or other items) when packs containing minimal amounts still greatly exceed what was needed. The previously mentioned suppliers were used as well as Sparkfun Electronics at http://www.sparkfun.com/, and Lowes at http://www.lowes.com/.

Table 10.

Hardware and Construction Materials

Component	Quantity	Estimated Amount	Unit Price	Total	Price From
Single Header Strip	1 Pack	25%	$1.29	$0.32	L&S Electronics*
White Project Case	1		$12.32	$0.00	Newark
3 Terminal Connectors	1 BOX	50%	$27.23	$13.62	Newark*
8 Pin Stackable Header	9		$0.50	$0.00	Sparkfun
Proto Board	1		$4.95	$0.00	Jameco**
22AWG Solid White Wire 25'	1	50%	$3.49	$1.75	Jameco*
20AWG Stranded Black Wire 25'	1	25%	$4.40	$1.10	Newark**
20AWG Stranded RED Wire 25'	1	25%	$4.40	$1.10	Newark**
Jumper Wire Kit	1	10%	$11.95	$1.20	Jameco*
Electrical Tape 60'	1	50%	$1.35	$0.68	Jameco*
2"x6'x8' Treated Lumber	1	30%	$5.57	$1.67	Lowes**
0.5"x24"x48" MDF Board	1	10%	$5.73	$0.57	Lowes**
0.75"x24"x48 M"DF Board	1	10%	$8.44	$0.84	Lowes**
0.125"x4'x8' MDF Board	1	10%	$9.96	$1.00	Lowes**
Totals:				$23.84	

In actuality many of the items listed in were not used in entirety, and several of the trainers could be made using the materials marked with asterisks. With the electronics cost at $155.78, the total equipment and material price for producing the Arduino Trainers amounts to an estimate of $179.62. This price was lower in actuality as many of the purchased items were already available for use and came at no cost.

CHAPTER 5

CONCLUSION AND RECOMENDATIONS

It is recommended that ETSU work with local area highs schools to investigate their interest in using the created project as an educational tool to satisfy previously educational needs. ETSU has already provided educational tools to David Crockett High School (DCHS) through the Gear Up program, and has multiple contacts with that school. Gear Up is a “discretionary grant program, is designed to increase the number of low-income students who are prepared to enter and succeed in postsecondary education” (U.S. Department of Education, 2012, p. 1). Past projects between DCHS include constructing and automated greenhouse, a solar powered outdoor media classroom, and a video capturing wildlife observation system. It is recommended that administrators of that school be given a demonstration of the created project and a copy of the educational materials for their consideration. Should the GearUp grant be renewed, it is recommended that any future schools be informed of this project for their consideration.

A web search on many of the local area high schools in east Tennessee reviled that many do not currently teach the class C Programming Applications, although many do have a programming class of some nature. This could cause poor adoption of the project created for this thesis. As a result it is also suggested that afterschool and summer school project coordinators be given a demonstration of the trainer. One such afterschool project is LEAPs, which is funded by the Tennessee lottery. “The overall goal of LEAPs is to provide Tennessee students with academic enrichment opportunities that reinforce and complement the regular academic program” (TDOE, 2010, p. 4). The Lottery for Education: Afterschool Programs is designed to use revenue generated by the lottery to further educate students classified as “at risk”. The criteria for being at risk, however, are rather broad. A survey of 27,113 students reviled that 83% were considered at risk for criteria involving poverty, dysfunctional living situations, enrolment in a school with inadequate yearly progress, or at risk of being behind in classes (TDOE, 2010, p. 3).

Organizations involved with LEAPs should be contacted, or a grant proposal for LEAPs funding should be made. In order to qualify for a grant at least 50% of students must qualify as at risk by the previously mentioned standards. The goal of the grant proposal must be education, and grant proposals must be clear and well defined with a reasonable budget. The priorities of education must include an average student participation of 15 hours a week. The program must also develop reading, math, science, or computer literacy skills. There must be academic mentoring and the program has to have some sports or leisure time (TDOE, 2012). Other avenues of reaching students include various clubs and college center learning programs for students offered by the University of Tennessee and University School at ETSU.

While the prototype trainers and Lab manual was developed for the previously mentioned purpose of satisfying STEM educational need for high school students in Tennessee, the project can serve other educational uses for ETSU. Dr. Zhibin Tan has requested a copy of the finished lab manual and access to the prototype trainers for possible uses in ENTC 4337 – Microprocessors. While the coursework may not be college level material, it can be used to introduce the concept of how microprocessors work and be used as an introduction to more complicated systems.

Mr. Garth Ghearing will also be reviewing the lab manual and trainer to investigate its uses in ENTC 4957/5957 – Introduction to Mechatronics. The created project will most likely not be used as required course material but rather to provide individual students with a knowledge basis of the Arduino's capabilities and applications in automated and robotic systems. Many of the sensors and programs used in the trainers have already been used in robotic and automation classes to provide a simple and cheap interface between inputs and outputs. Mr. Ghearing has also state that he will further the development of the prototype and associated labs in an effort to produce pilot models of the trainers in an automated production line using the equipment at ETSU. These pilot trainers would include improvements on the original design making them more capable of being manufactured using the automatic storage and retrieval system in combination with the Denso robots mounted on the conveyer belt work cell at ETSU.

Should local area high schools show an interest in this project, it is recommended that the pilot models of the trainers be put into production to not only help educate students in high schools but also provide valuable teaching opportunities to those enrolled in Engineering Technology at ETSU. Constructing a pilot version of the trainers will allow college students to have hands-on experience in constructing and using various electronic circuits. Producing these on an automated line will allow for education in programming and robotic control. Creating an easily manufactured pilot trainer will expose college students to the concepts of design for manufacturing and optimizing production.

Some design changes would need to be made for the pilot trainers to make them more easily produced on an automated line and to improve the functionality. It is recommended that a project box be used for the first trainer in order to provide a more consistent design with less inherent variability to each created trainer. This will also reduce construction time. The first trainer should also use printed circuit boards rather than prototype boards to mount electronic components. This will vastly reduce wiring time and errors in the circuits. This will also eliminate the need for the three terminal connectors used in the prototype and allow for pick and place robotic construction of many of the circuits. The LEDs should also be replaced with more compact flatter LEDs that will not protrude from the top of the project box. This will allow for the same protection as the aluminum hosing on the prototype trainer without the need for creating the aluminum faceplates.

It is recommended that a single type of stranded 20 gauge wire be used for all wiring connections. This will allow the wiring to be more flexible, and using a single type will reduce the cost for purchasing multiple rolls. This type of wire will not easily connect with the 8 pin stackable headers used in the prototype and the headers do not have a desirable appearance. It is recommended that prefabricated jumper wires with the appropriate headers be used, or one of the many shield kits available for the Arduino be used for the connections. The author of this thesis will make every effort possible to continue the development and distribution of the created project in the future.

The prototypes and lab manual teach many of the basic C programming commands and introduce many other library specific commands. They do this in a way that provides hands-on experience with technology that is used in many other related fields. The labs also include key concepts for understanding the fundamentals of technology and incorporate high school appropriate math. The Lab assignments provided meet standard core requirements 2.0-4.0 for the class of C Programming Applications as defined by the Tennessee State Department of Education. This is done in an interdisciplinary way with plenty of information provided to allow teachers to educate students with ease. In conclusion the created prototype trainers and lab assignments meet the goals set out by this thesis by creating an educational tool that will meet the STEM educational needs in Tennessee high schools.

REFERENCES

Arduino (2012). Arduino - Home Page. Retrieved 06/25/2012 From: http://arduino.cc/en/

Arduino (2012). Arduino Mega 2560. Retrieved 06/26/2012 From: http://arduino.cc/en/Main/ArduinoBoardMega2560

Berry, R.E., & Meekings B.A.E. (1984). A book on C. Southampton, Great Britian: Camelot Press. ISBN 0-333-36821-5

By Members of the 2005 “Rising above the gathering storm” Committee, Prepared for the Presidents of the National Academy of Sciences, National Academy of Engineering, Institute of Medicine. (2010). Rising Above the Gathering Storm, Revisited, Rapidly Approaching Category 5. The National Academies Press. Retrieved 06/12/ 2012, From: http://www.uic.edu/home/Chancellor/risingabove.pdf

Crum, M., & McMurray, B. (2012) Design planning and prototype creation for the Arduino Training Module. Johnson City, TN: East Tennessee State University. Prepared for: ENTC 5900: Independent Study.

Davies, A. (1998). Handbook of condition monitoring: Techniques and methodology. London SE1 8HN, UK: Chapman & Hall, an imprint of Thomson Science. ISBN 0-412-61320-4

Douglas, J., Iversen, E., & Kalyandurg C. (2004). Engineering in the K-12 classroom, an analysis of current practices & guidelines for the future. The American Society for Engineering Education. Retieved 06/18/2012 From: http://www.asibei.org/oddi/libros/Engineering%20in%20the%20K-12%20classroom.pdf

Harbison, S.P., & Steele, G.L. Jr.(1984). C: A reference manual. Englewood Cliffs, NJ: Prentice-Hall. ISBN 0-13-110016-5

Johnsonbaugh, R. & Kalin, M. (1997) C for scientists and engineers. Upper Saddle River, NJ: Prentice-Hall. ISBN 0-02-361136-7

Kernighan, B.W., & Ritchie, D.M. (1978). The C programming language. Englewood Cliffs, New Jersey: Prentic-Hall, Inc. ISBN 0-13-110163-3

Lundberg, K.H. Analog to digital converter testing. Retrieved 06/25/2012 From: http://web.mit.edu/klund/www/papers/UNP_A2Dtest.pdf

National Science Board. (2007). National action plan for addressing the critical needs of the U.S. science, technology, engineering, and mathematics education system. (NSB-07-114) October 30, 2007. Retrieved 06/11/2012, From: http://www.nsf.gov/nsb/documents/2007/stem_action.pdf

Owens Community College (2012) Audience and person. The writing center. Retrieved 6/23/2012 From: https://www.owens.edu/writing/audience.html

Princeton University. (2012). Integrated science. (Last Updated March 1, 2012) Retrieved 06/18/2012 From: http://www.princeton.edu/integratedscience/

Tennessee Department of Education. (2010). Lottery for education: Afterschool programs (LEAPs). Retrieved 7/3/2012 From: http://www.tn.gov/education/safe_schls/learning/doc/2010-11LEAPsAnnualReport.pdf

Tennessee Department of Education. (2012). Lottery for education: afterschool programs (LEAPs). Retrieved 7/3/2012 From: http://www.tn.gov/education/safe_schls/learning/leaps.shtml

Tennessee State Board of Education. (2005) Core standards for C programming applications. Tennessee Department of Education. Retrieved 6/20/2012 From: http://www.tn.gov/education/ci/computer/doc/cprogapps.pdf

Tennessee State Board of Education. (2008). Principles of technology I, principles of technology II, biology for technology curriculum standards. Tennessee Department of Education. Retrieved 6/20/2012 From: http://www.tn.gov/education/ci/computer/doc/cprogapps.pdf

APPENDICES

Appendix A: Arduino Programmers

The following people contributed to the programming used in the created project or provided code that educated Brandyn M. Hoffer allowing him to create the project. The People listed here are given credit for their work, and this list is intended in no way to imply that they endorse the created project or this thesis.

The following people are the developers of the Arduino project and further contributed through providing example sketches used for reference when investigating the Arduino's capabilities:

Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis

Credit is given to DojoDave, http://www.0j0.org, 2005, for creating the "Button" sketch used for learning the Arduino inputs and outputs later modified by Tom Igore.

Credit is given to Nicholas Zambetti, 2006, for creating the "ASCIITable" sketch later modified by Tom Igore, which was the basis for the creation of Lab 4: Better Serial Communication.

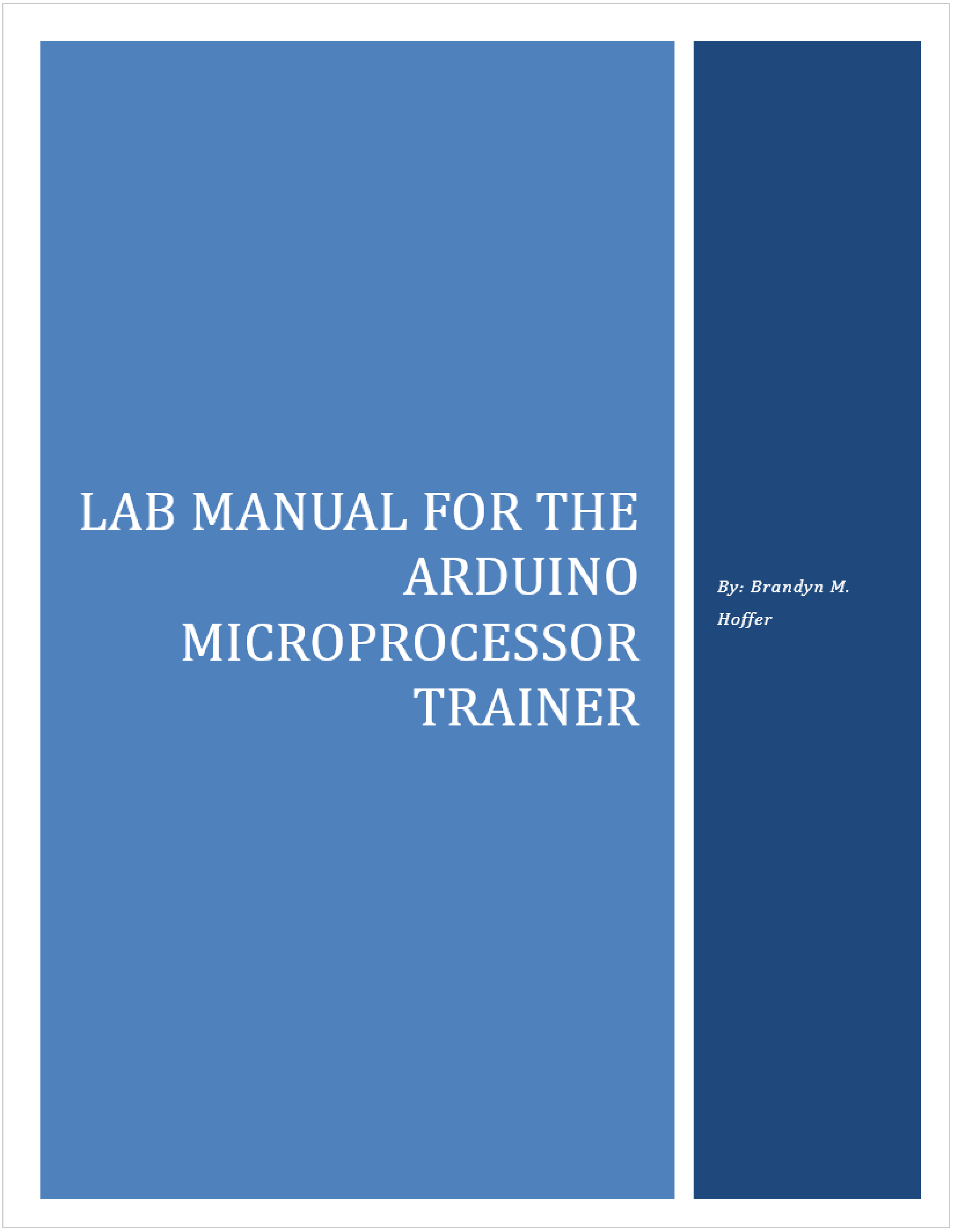
Credit is given to Mark Stanley, and Alexander Brevig for creation of the "keypad.h" header file used in Labs 10-14, and for the example sketches provided with the library which demonstrated its uses. The "CustomKeypad" sketch was modified to provide the basis for Lab 10: Interfacing With a Matrix Keypad.

Credit is given to Limor Fried, 2009, who modified the LiquidCrystal.h library created by the developers, which was used in Labs 9-14.

Credit is given to Hernando Barrang who developed “Wiring”, “an-open source programming framework for microcontrollers” (http://wiring.org.co/) which the Arduino syntax is based on.

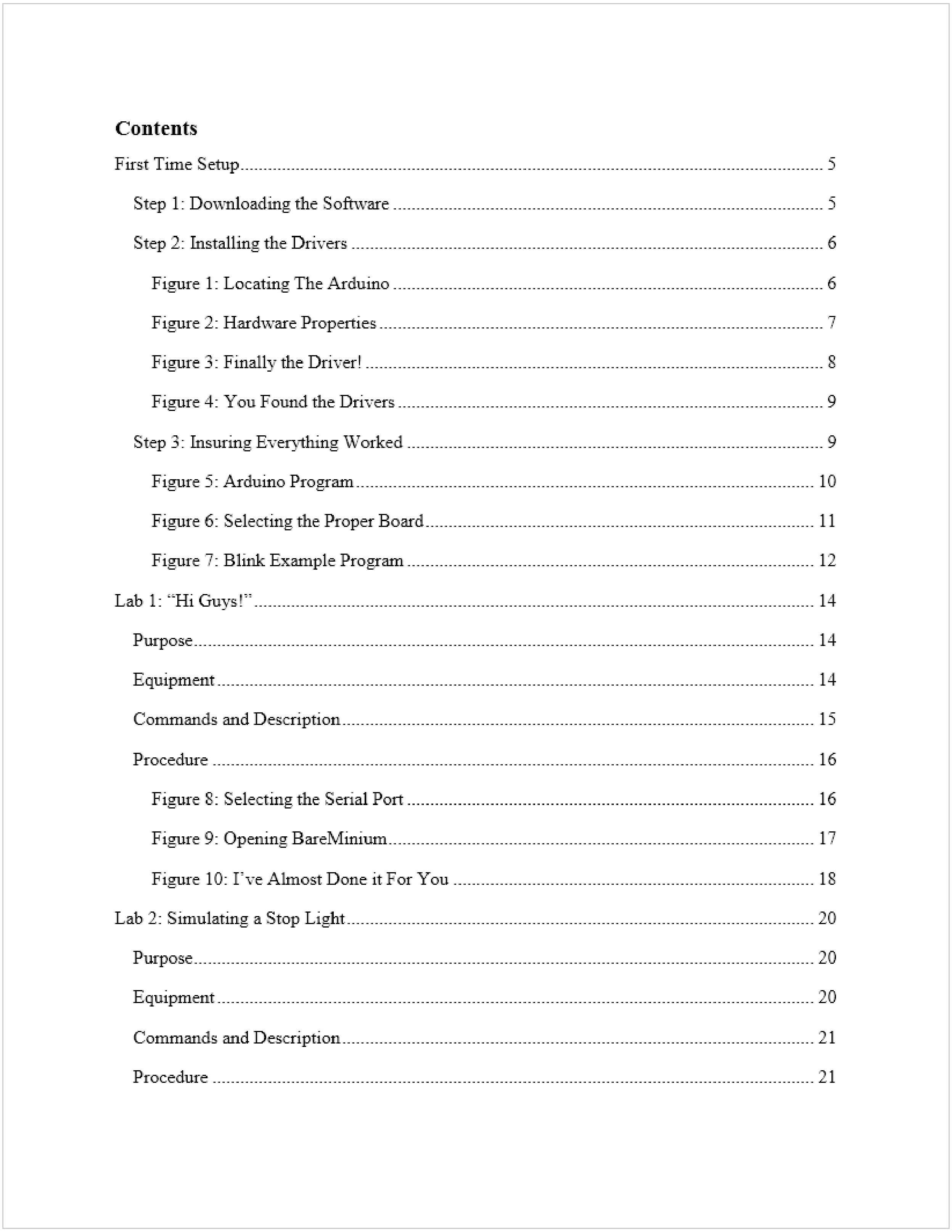
Credit is given to Ben Fry and Casey Reas for developing Processing “an open source programming language and environment” (http://processing.org/) which the Arduino environment is based on.

Appendix B: The Lab Manuel

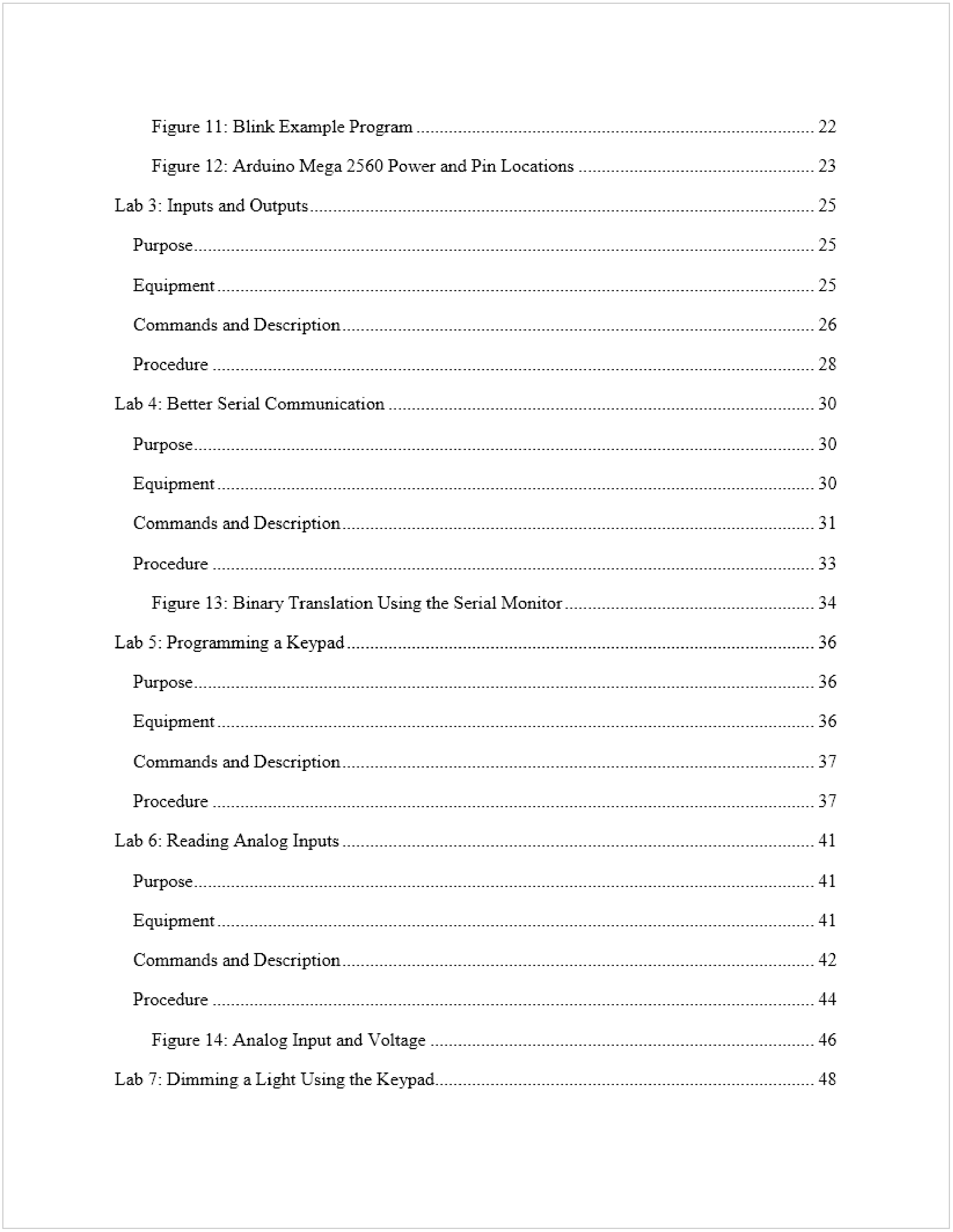


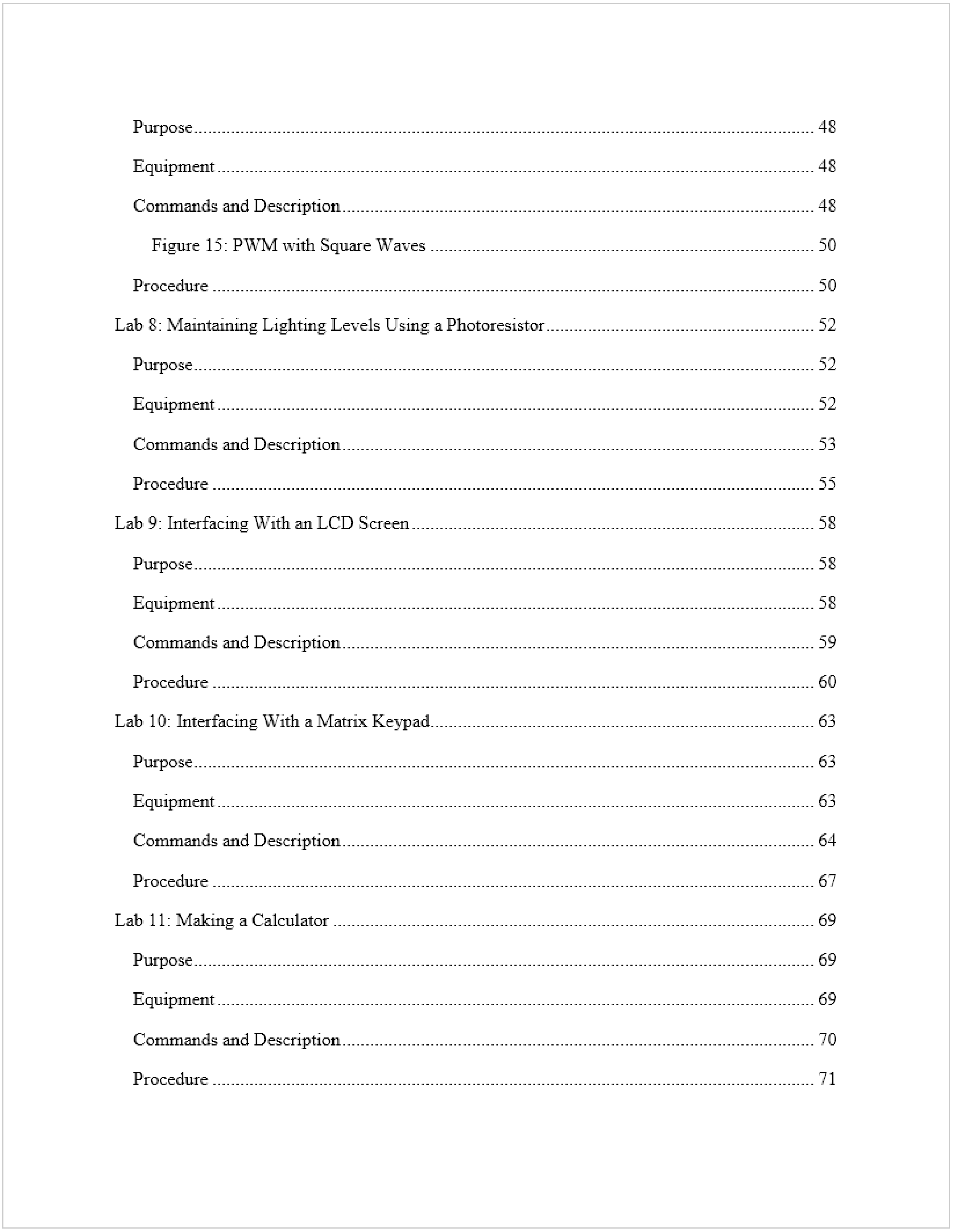
LAB MANUAL FOR THE ARDUINO MICROPROCESSOR TRAINER

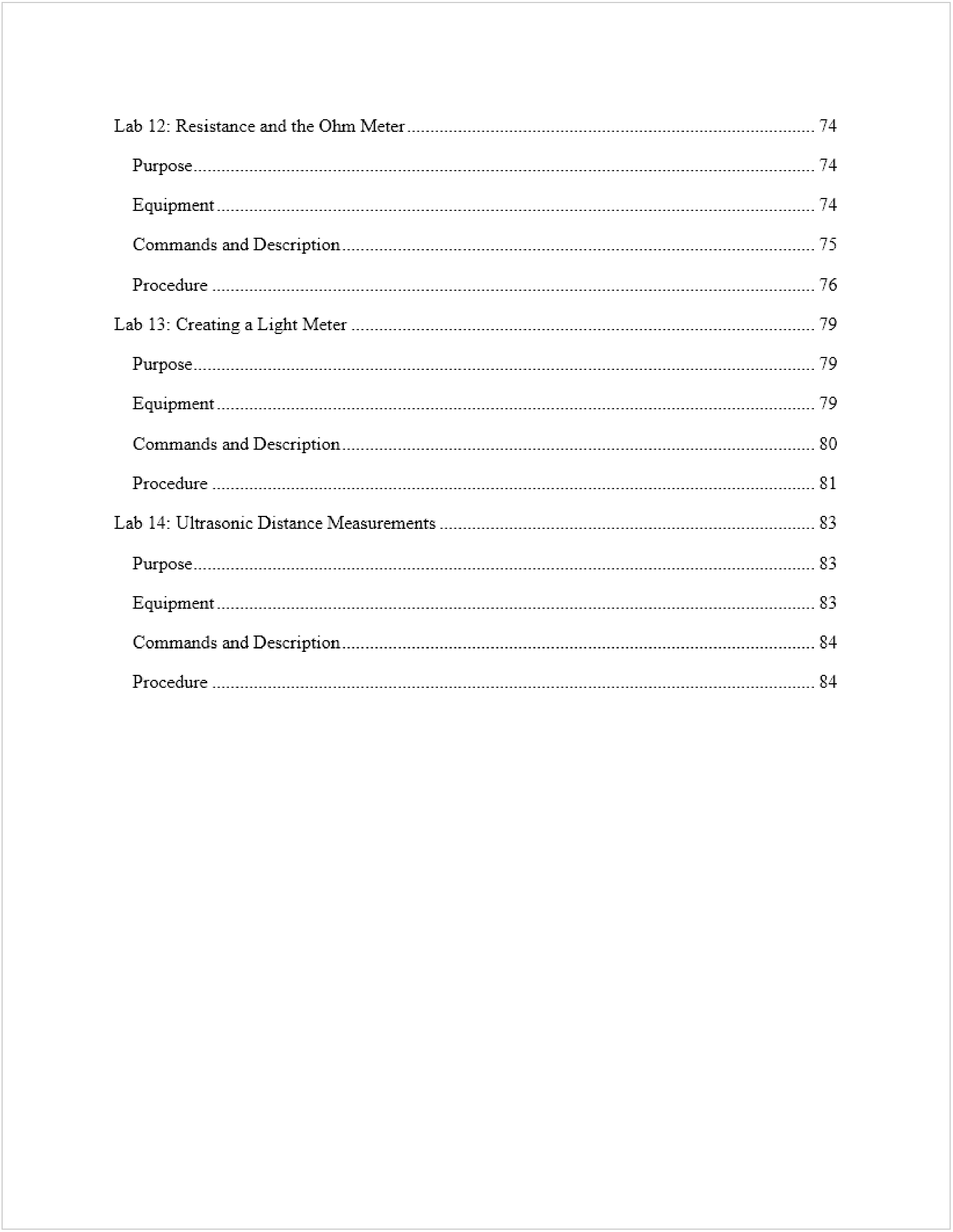
By: Brandyn M. Hoffer



Contents







First Time Setup

To set up your Arduino for the first time you will need to download the software, install the correct drivers to your computer, and select the correct USB port to ensure the signals are actually sent to the Arduino device rather than an external hard drive, or printer. This section will inform you on how to do all of this in a few easy steps… well, a few steps. Once the device has been set up it should work on that computer without the need for another setup unless a different Arduino device is being used. The steps described in this section are meant to guide you for a Windows 7 operating system. If you have an OS different from Windows 7 these steps may be different, but the same functions needs to be performed.

Step 1: Downloading the Software

First go to the Arduino web page at http://www.arduino.cc/ . Next you will see a menu bar located under the Arduino home page logo with options such as "Buy | Download | Getting Started…" From here, click on the "Download" button. Next click on the word "Windows" under the word "Download" on the page. This will prompt you to open or save the file. Make sure you save it to an easy to find location such as your desktop or documents folder. This file will download as "Arduino-1.0-windows.zip". You will then need to extract the contents of this .zip file to the same location. You can do this using a variety of programs such as *WinRAR* or *WinZip*. If you do not already have one of these programs or a similar program download and install them from www.win-rar.com or www.winzip.com .

To extract the files, right-click on the Arduino file and select "extract here" or "extract file" and select the appropriate destination. Next while the file is extracting grab yourself a cup of coffee or soda, this should take a minute or so. Once the file is done extracting you should see "arduino-1.0" as a file folder in the selected location. It should also be noted at this point that some antivirus software may warn you that this program is not trusted or unsafe. Being an open source software many firewalls and antivirus programs will try and prevent you from installing this software, however, if you have downloaded the program from the correct location I can assure you it has never negatively affected any computer I have ever put it on. Please ignore these warnings and add any exceptions necessary to install this on your computer.

Step 2: Installing the Drivers

Once the Arduino folder has been extracted to the proper location you must connect your Arduino Mega or other Arduino device. If Windows prompts you to install the drivers locate the option that lets you install them manually, and continue reading from the next paragraph. If Windows does not prompt you to install the drivers, select "Devices and Printers" in the start menu, or locate this option in the control panel. This can also be found by typing "Devices and Printers" in the search bar of the start menu. Locate the "Arduino Mega 2560" in this window. It should be listed under "Unspecified" The figure below shows the Arduino device at the bottom of the screen. Double click on the device and a new window will appear with the menu tabs "General" and "Hardware". Click on the "Hardware" option, then the "Properties" button located towards the bottom right hand corner of the window. This can be seen in Figure 2. A new window will open up with another button that says "Change settings". Click that button. And finally, yet another window with multiple menu tabs will pop up. Click the "Driver" tab in this window and the button that says either "Update Driver…" or "Install Driver…". This can be seen in Figure 3.

Figure 1: Locating The Arduino

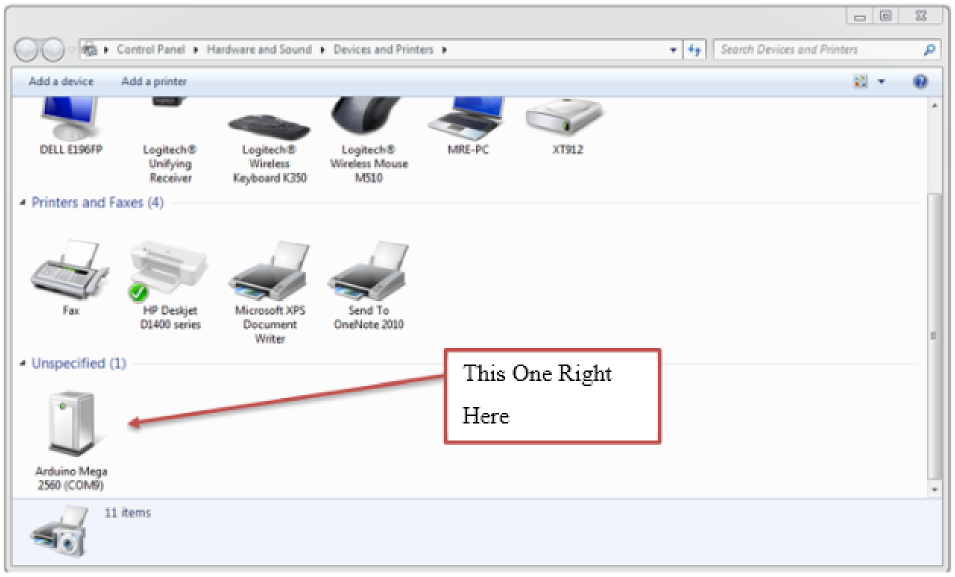
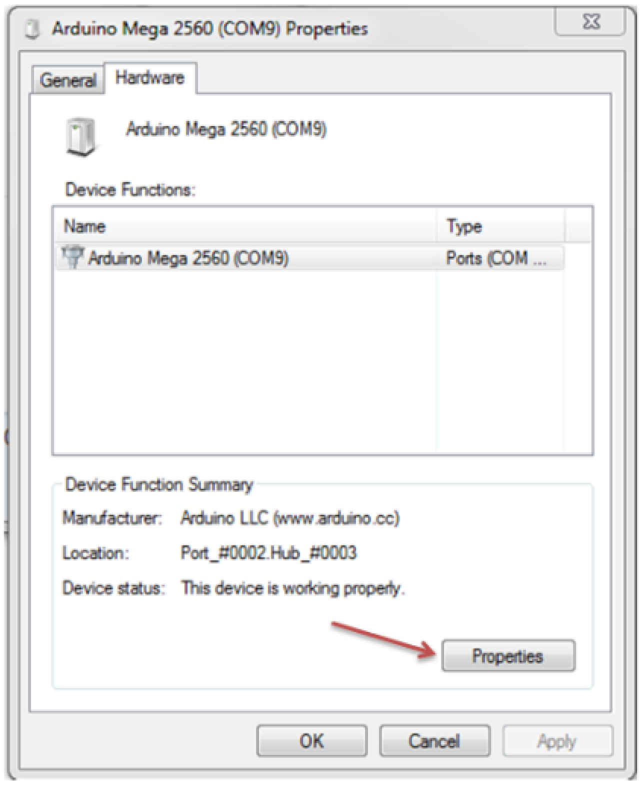

Control Panel ▸ Hardware and Sound ▸ Devices and Printers ▸
Search Devices and Printers
Add a device
Add a printer
DELL E196FP
Logitech® Unifying Receiver
Logitech® Wireless Keyboard K350
Logitech® Wireless Mouse M510
MRE-PC
XT912
Printers and Faxes (4)
Fax
HP Deskjet D1400 series
Microsoft XPS Document Writer
Send To OneNote 2010
Unspecified (1)
Arduino Mega 2560 (COM9)
This One Right Here
11 items

Figure 2: Hardware Properties

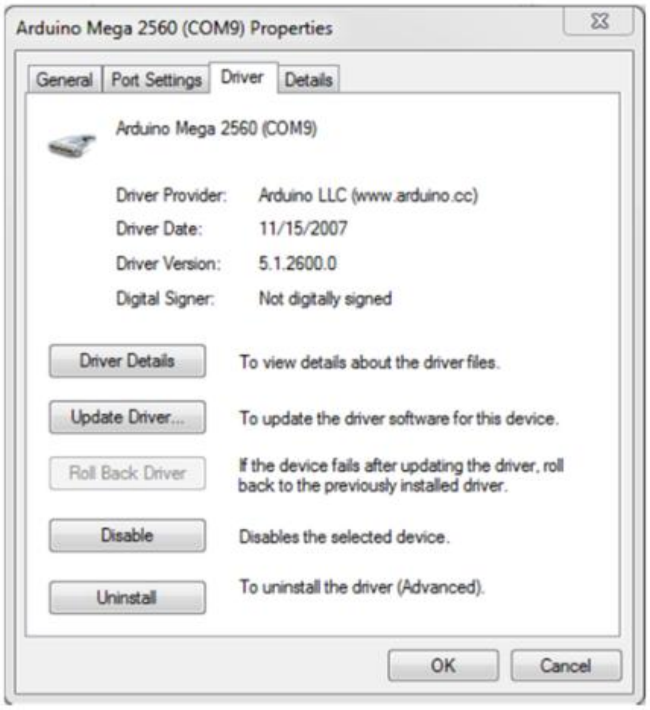


Manufacturer: Arduino LLC (www.arduino.cc)

Location: Port_#0002.Hub_#0003

Device status: This device is working properly.

Figure 3: Finally the Driver!


Arduino Mega 2560 (COM9) Properties

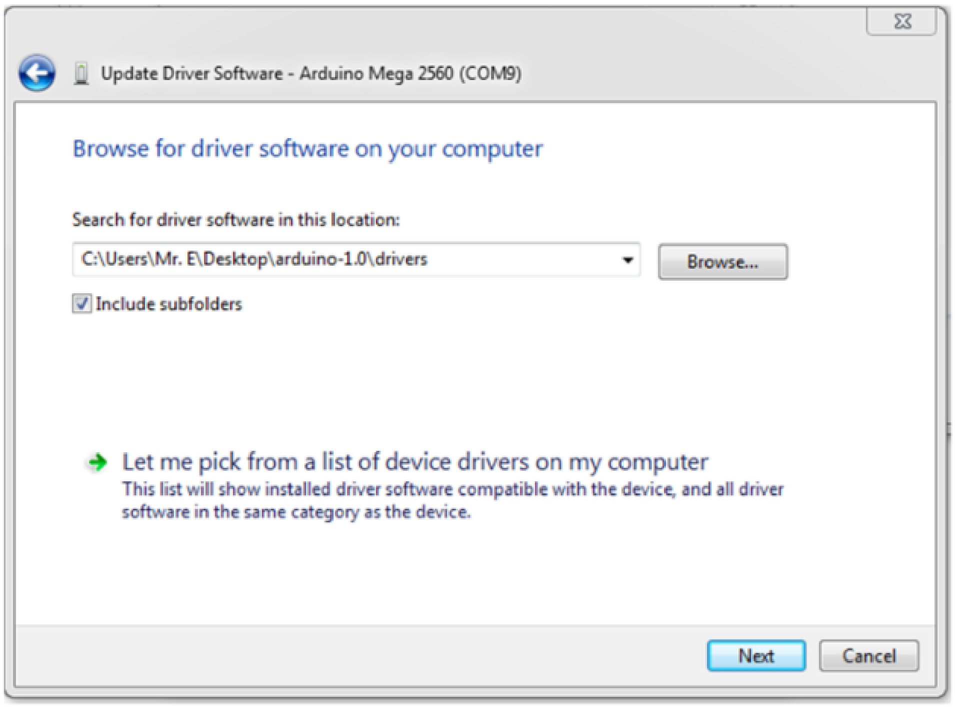
Arduino Mega 2560 (COM9)

Driver Provider: Arduino LLC (www.arduino.cc)
Driver Date: 11/15/2007
Driver Version: 5.1.2600.0
Digital Signer: Not digitally signed

Driver Details — To view details about the driver files.
Update Driver... — To update the driver software for this device.
Roll Back Driver — If the device fails after updating the driver, roll back to the previously installed driver.
Disable — Disables the selected device.
Uninstall — To uninstall the driver (Advanced).

Once you have located the window that asks if you would like to search automatically for driver software or browse your computer for it select the option that allows you to manually browse your computer for this software. Click on the "Browse" button and locate your arduino-1.0 folder that was previously installed. In this folder, click on the "drivers" subfolder, and hit "OK". Figure 4 shows this window and the location on the computer. Also make sure the "Include subfolders" box is checked. Now hit the "Next" button and the mission has been accomplished.

Figure 4: You Found the Drivers



Update Driver Software - Arduino Mega 2560 (COM9)

Browse for driver software on your computer

Search for driver software in this location:

C:\Users\Mr. E\Desktop\arduino-1.0\drivers

☑ Include subfolders

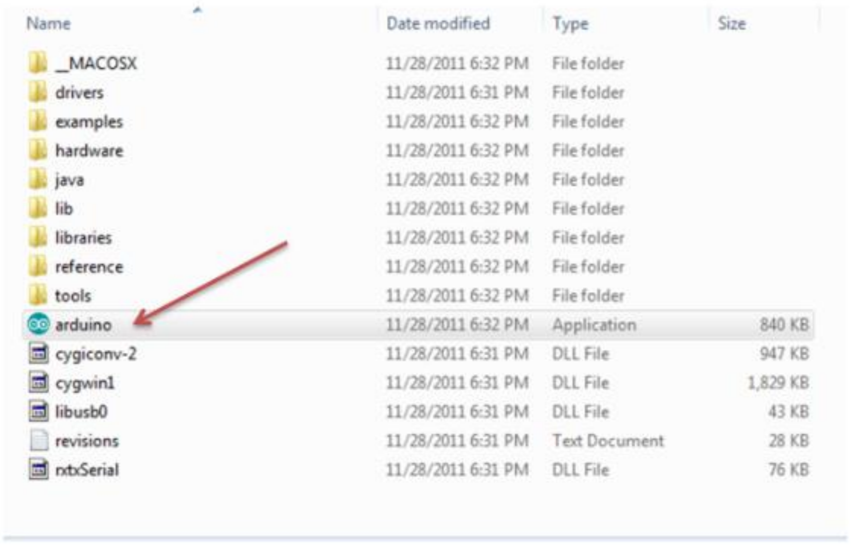
Let me pick from a list of device drivers on my computer

This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

Step 3: Insuring Everything Worked

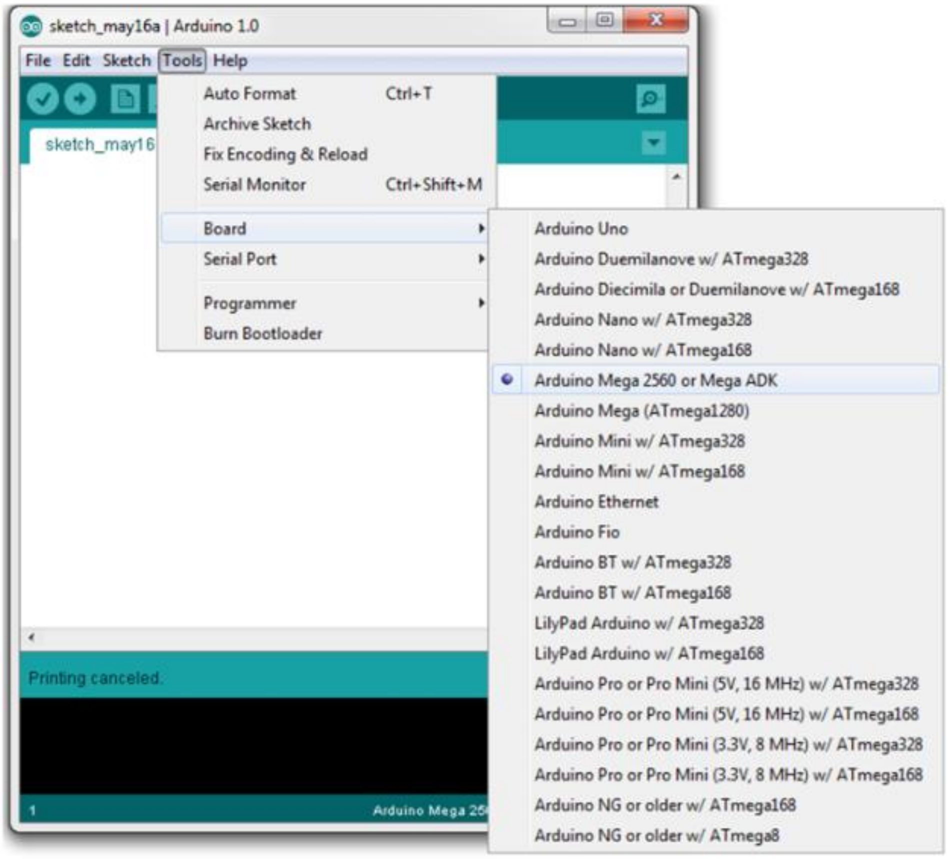
Now you will make sure everything was performed correctly by running an example lab in the Arduino program. To start open up the Arduino folder downloaded in the previous step and click on the Arduino program shown in the figure below. Next, click on "Tools> Board> Arduino Mega 2560 or Mega ADK". This can be seen in Figure 6 to make sure you locate the proper path. To select the proper USB port click on "Tools> Serial Port> COM#" where the # symbol represents the USB port the Arduino is connected to. If you do not know which USB port to select you must go back to the "Devices and Printers" window in Step 2. Looking at Figure 1 you will notice that the "Arduino Mega 2560" has "(COM9)" written next to it. This is the proper USB port. The number on your computer may be different from the one listed in the figure, however, this just needs to match the same port in the "Serial Port" menu.

Figure 5: Arduino Program



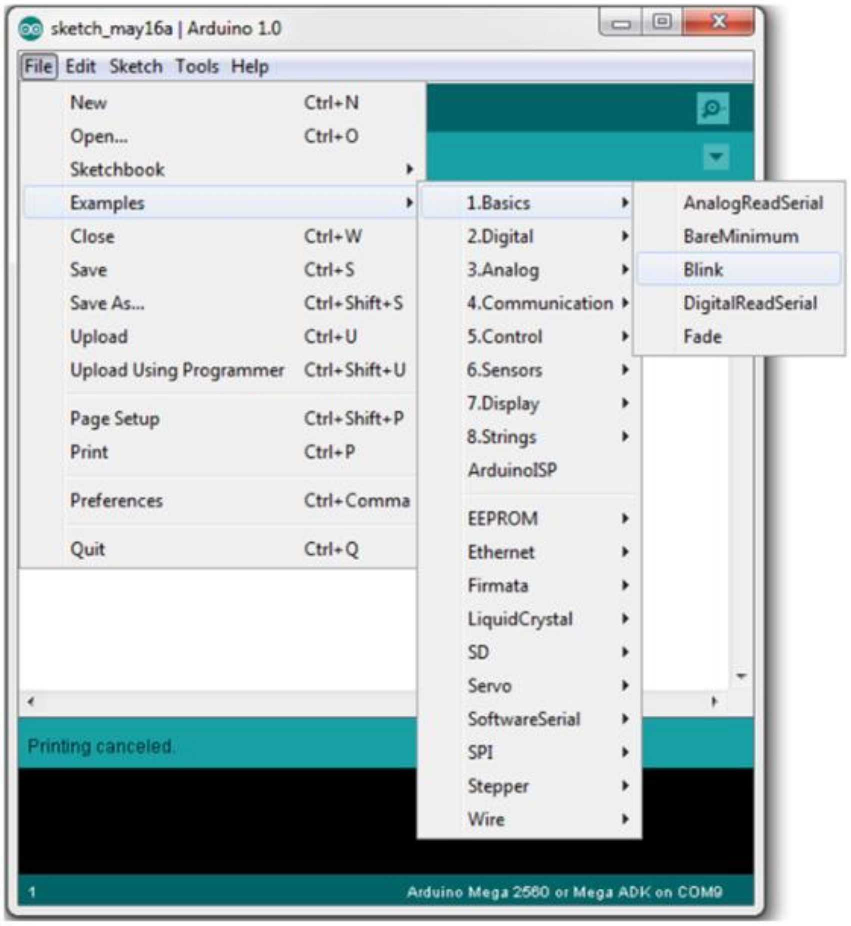
Name	Date modified	Type	Size
_MACOSX	11/28/2011 6:32 PM	File folder	
drivers	11/28/2011 6:31 PM	File folder	
examples	11/28/2011 6:32 PM	File folder	
hardware	11/28/2011 6:32 PM	File folder	
java	11/28/2011 6:32 PM	File folder	
lib	11/28/2011 6:32 PM	File folder	
libraries	11/28/2011 6:32 PM	File folder	
reference	11/28/2011 6:32 PM	File folder	
tools	11/28/2011 6:32 PM	File folder	
arduino	11/28/2011 6:32 PM	Application	840 KB
cygiconv-2	11/28/2011 6:31 PM	DLL File	947 KB
cygwin1	11/28/2011 6:31 PM	DLL File	1,829 KB
libusb0	11/28/2011 6:31 PM	DLL File	43 KB
revisions	11/28/2011 6:31 PM	Text Document	28 KB
rxtxSerial	11/28/2011 6:31 PM	DLL File	76 KB

Figure 6: Selecting the Proper Board


sketch_may16a | Arduino 1.0
File Edit Sketch Tools Help
sketch_may16
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board
Serial Port
Programmer
Burn Bootloader
Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
● Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
Printing canceled.
1
Arduino Mega 25

Once the proper board and serial port have been specified, go to the menu bar and select "File> Examples> 1.Basics> Blink". This will open up another Arduino window with a preloaded example program that can be used to test the connection and make sure everything is working properly. Figure 7 illustrates the location of this pre-installed program.

Figure 7: Blink Example Program



```
Printing canceled.
```

Located directly below the menu bar you will see a few buttons. One of these looks like a circle with a right arrow in it. This is the upload button. Hovering the mouse over these buttons will allow you to read what they are. Click this button and if everything worked correctly you will see the yellow led located next to pin 13 blinking on and off at a rate of 1 second after the program displays the words "Done uploading." If you get an error that says "avrdude:" followed by a bunch of nonsensical letters and numbers, this means something is wrong with the Arduino's communication. Check and make sure the Arduino is connected properly and recognized by the computer. Also check and make sure the correct “COM#” port has been selected and the proper board has been specified, than try again. If this does not work, try restarting the computer. Open the example again then uploading it a final time. If you still receive an error you will have to Google search the exact error you are getting to find a solution. There are many other errors that can occur and being an open source device, Google will become one of your best friends for solving these errors.

Lab 1: "Hi Guys!"

Purpose

This lab will introduce the basic structure of Arduino sketches. The two basic text commands that must be present in every Arduino sketch program will be learned. These commands are used to reference the set up and running portion of the sketch. You will also learn the serial response commands allowing for communication between the Arduino module and a computer. Serial communication with a computer is useful both for interfacing with the commuter and debugging programs through displaying the value of either digital or analog pins, or the value represented by a variable. To learn these commands you will write your first program using only the commands listed in the *Commands and Description* section of this lab. All commands must be written exactly as they appear in this section. That includes capitals and spaces. Successful completion of this lab will result in a program that can be loaded into the Arduino Mega, and cause a response on the computers serial monitor.

Equipment

Arduino Mega

Arduino USB cable

Computer

Commands and Description

Command	**Description**

void setup() {
}
	This command runs once during the start of a program and is used to initialize variables, pin modes, start using libraries, and other things you will be introduced to in later labs. Information is placed on the preceding line between the brackets.

void loop() {
}
	This command runs continuously in a loop as its name implies. The portion of the program that solicits responses, performs calculations, or has to do with any input or output is placed on the preceding line between the brackets.

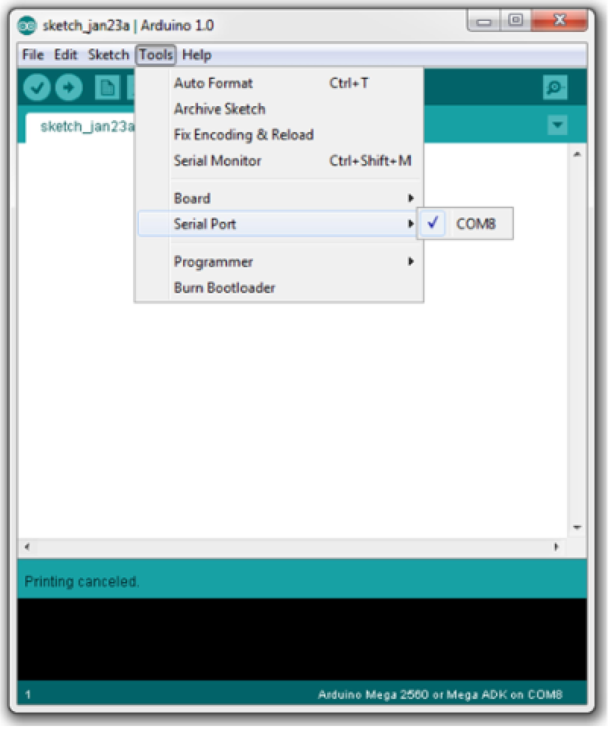
Serial.begin(9600)	This command is used to begin serial communication at the beginning of a sketch. It is used in the "void setup()" and will allow for communication between the computer and the Arduino via the attached USB cable. The value 9600 is the baud rate and should be set to match the baud rate of the computer's serial monitor.
Serial.print()	This command "prints" anything written between the colons. If the sketch is set up properly and serial communication is running, Serial.print("Hi Guys") will return "Hi Guys" on the computer's serial monitor.

Procedure

1. First plug in the USB interface cable to the computer. Once the computer has recognized the device open up the Arduino interface program to create your first sketch.

2. Next ensure the Arduino is communicating with the computer. To do this, select "Tools" from the drop down menu, then "Serial Port". You should see a port, or list of ports labeled "COM#" where the # signifies some number. If there is only one, select that device. If there are several, simply go to the "Control Panel" on the computer. Open up "Devices and Printers", and look for the Arduino under "Unspecified". Then go back to the sketch program and select the corresponding port. This can be seen in the figure below. You may also have to select which Arduino you are using from "Tools">"Board">, then select the Arduino you have. This step will be omitted in future labs, so remember how to do this.

Figure 8: Selecting the Serial Port

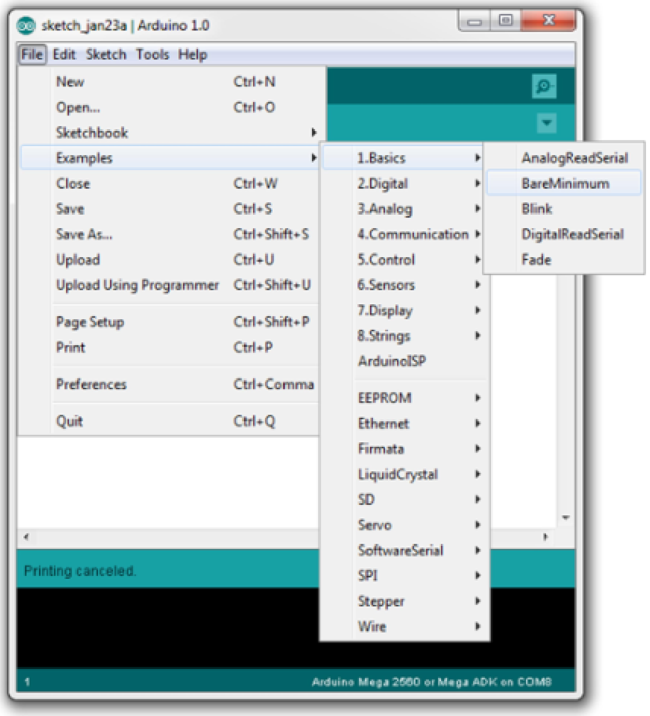


Printing canceled.

1 Arduino Mega 2560 or Mega ADK on COM8

3. Once the serial port has been specified, it is time to begin writing the sketch. Often in programming it is easier to start by opening another sketch that has the same programming structure, or commands, that you will be using. Many of the labs in this manual build on each other and others can be completed by modifying existing programs. To select a sketch you can build this lab from, select "File" >"Examples">"Basics">"BareMinium". This can be seen below, and if done properly you will see a sketch containing the "setup" and "loop" commands.

Figure 9: Opening BareMinium



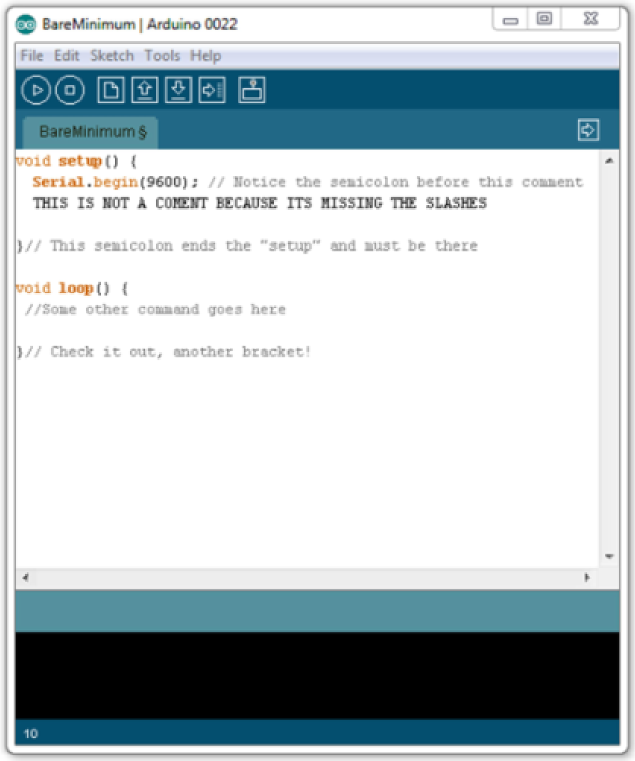
```
Printing canceled.
```

```
1                    Arduino Mega 2560 or Mega ADK on COM8
```

4. If you choose to start by writing from scratch remember to pay close attention to syntax. The "setup" and "loop" commands must start with the brackets seen in the commands, and end with the appropriate closing bracket. All other commands introduced must be followed by a semicolon ";" to signify the end of that line of data. If you would like to leave a comment in your sketch without it affecting the program, simply type "//". Anything written after the slash marks, on that same line of code, will appear gray in the sketch and the program will be unaffected by it.

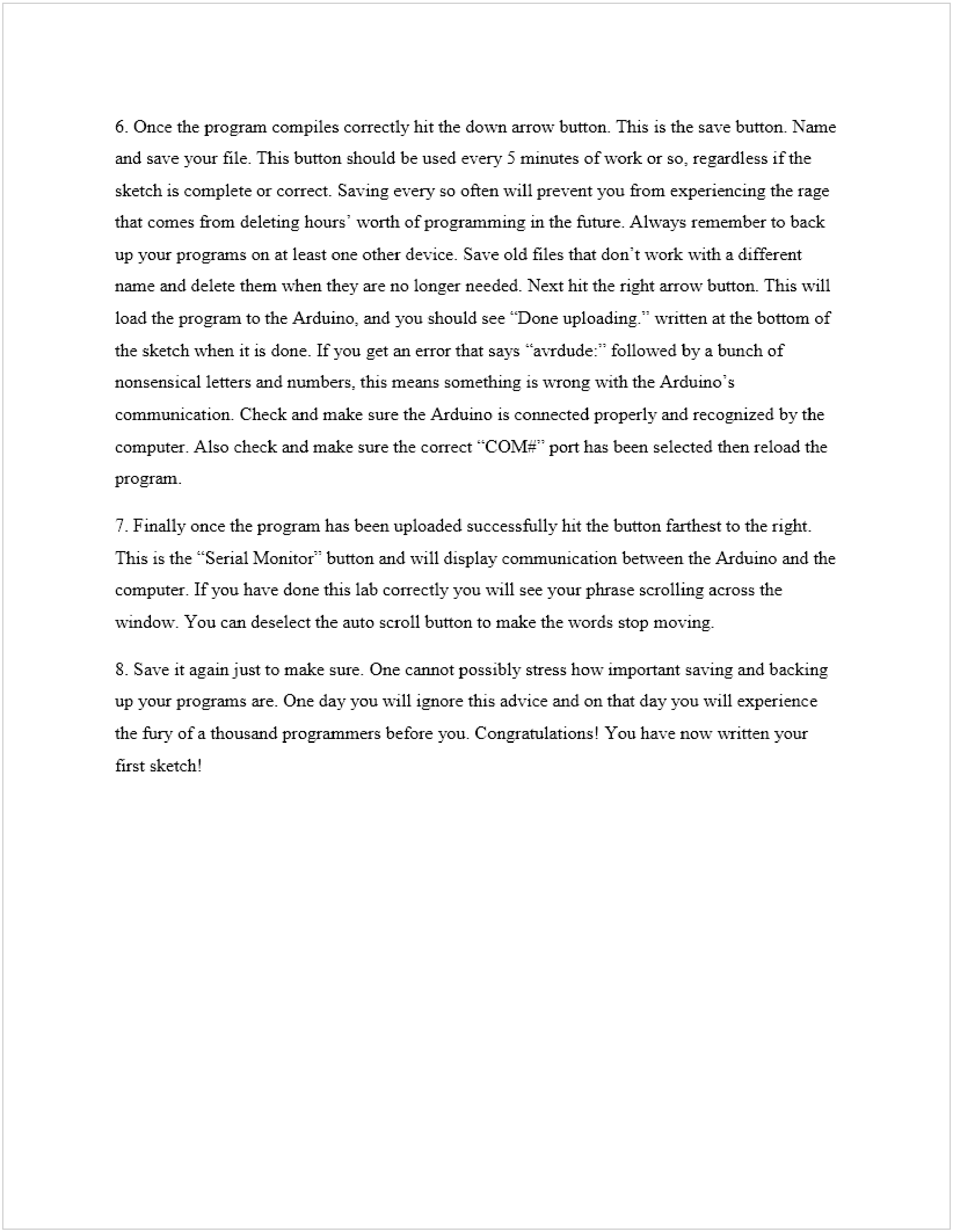
The "BareMinimum" program already has comments, and you should easily be able to pick them out. Use the commands introduced to write a sketch that returns "Hi Guys!" (or any other phrase) using the commands given. All commands entered correctly will appear orange to signify correct syntax. Figure 4 should clear up any confusion.

Figure 10: I've Almost Done it For You



```
void setup() {
  Serial.begin(9600); // Notice the semicolon before this comment
  THIS IS NOT A COMENT BECAUSE ITS MISSING THE SLASHES

}// This semicolon ends the "setup" and must be there

void loop() {
 //Some other command goes here

}// Check it out, another bracket!
```

5. Easy right? Once you have written the sketch check one last time to make sure you are not missing any syntax. Below the drop down menu are a few buttons. One of them looks like a circle check mark, and is the button on the farthest left. Hovering over these buttons will tell you what they do. This button is the "Verify" button and will compile your program and make sure you do not have any errors. If you get any errors, check the syntax again. Commands must be written EXACTLY as they appear in the *Commands and Description* section.



6. Once the program compiles correctly hit the down arrow button. This is the save button. Name and save your file. This button should be used every 5 minutes of work or so, regardless if the sketch is complete or correct. Saving every so often will prevent you from experiencing the rage that comes from deleting hours' worth of programming in the future. Always remember to back up your programs on at least one other device. Save old files that don't work with a different name and delete them when they are no longer needed. Next hit the right arrow button. This will load the program to the Arduino, and you should see "Done uploading." written at the bottom of the sketch when it is done. If you get an error that says "avrdude:" followed by a bunch of nonsensical letters and numbers, this means something is wrong with the Arduino's communication. Check and make sure the Arduino is connected properly and recognized by the computer. Also check and make sure the correct "COM#" port has been selected then reload the program.

7. Finally once the program has been uploaded successfully hit the button farthest to the right. This is the "Serial Monitor" button and will display communication between the Arduino and the computer. If you have done this lab correctly you will see your phrase scrolling across the window. You can deselect the auto scroll button to make the words stop moving.

8. Save it again just to make sure. One cannot possibly stress how important saving and backing up your programs are. One day you will ignore this advice and on that day you will experience the fury of a thousand programmers before you. Congratulations! You have now written your first sketch!

Lab 2: Simulating a Stop Light

Purpose

Now that you are familiar with the basic structure of writing the Arduino sketch programs you will learn to interface the Arduino with external electronic components. The Arduino must be programed to control six LED lights, turning them off and on in a "stop light" simulation. There are two set of green, yellow, and red lights, that must turn off and on for specific amounts of time, and they must switch simultaneously. For example, as one row goes to green the red on that same row must go off, while the red for the other row comes on and the yellow goes off. If this does not occur at the same time, imaginary people will certainly crash, possibly sustaining serious imaginary injury or even imaginary death. To avoid the horror of such a thing happening, the time delay functions and digital output functions will be learned to control the LED light on the Arduino trainer. Once again, you must only use the commands in the *Commands and Description* section of this lab, and any from the previous lab necessary for operation.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Power Cable (Optional)

Arduino Trainer

Commands and Description

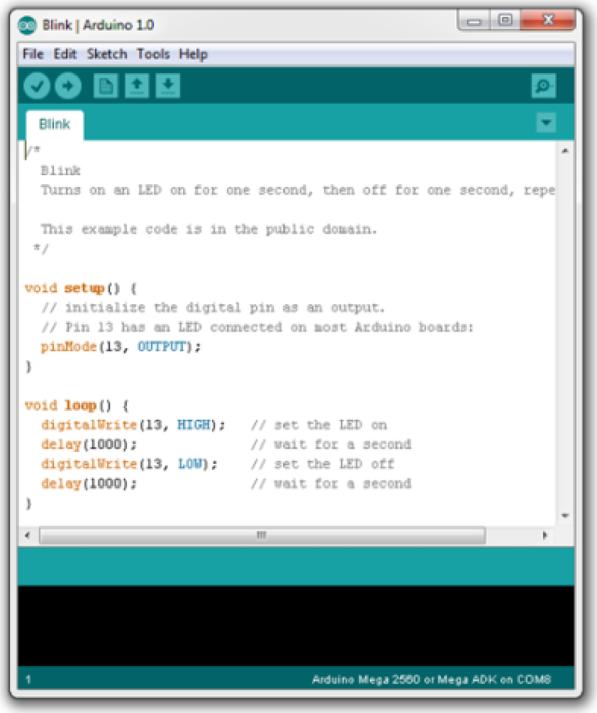
Command	**Description**
pinMode(#,*STATE)* *STATE*- INPUT/OUTPUT	This command is used to declare the state of a digital or analog pin of the Arduino (either INPUT or OUTPUT). The "#" in the command must be the pin number you wish to set, while the "*STATE*" must be either INPUT or OUTPUT. This command is mostly used in the "setup" portion of the program.
digitalWrite(#, *STATE*) *STATE*- HIGH/LOW	This command controls the state of a digital pin. It can be set HIGH to send a 5 V output signal to an external component or LOW to go to 0 V, sending no output. The "#" is the number of the pin you wish to change output of.
delay(#)	This command causes a delay in the program, or pauses it for the specified number ("#") of milliseconds. For example "delay(1000)" would delay the program for 1000 milliseconds or 1 second.

Procedure

1. Begin by opening up the Arduino program and ensuring the computer is communicating with the Arduino via the Arduino USB cable.

2. Open up the "Blink" example in the drop down menu by clicking on "File">"Examples">"Basics">"Blink". You should see the following program in figure 5.

Figure 11: Blink Example Program


Blink | Arduino 1.0
File Edit Sketch Tools Help
Blink
Arduino Mega 2560 or Mega ADK on COM8

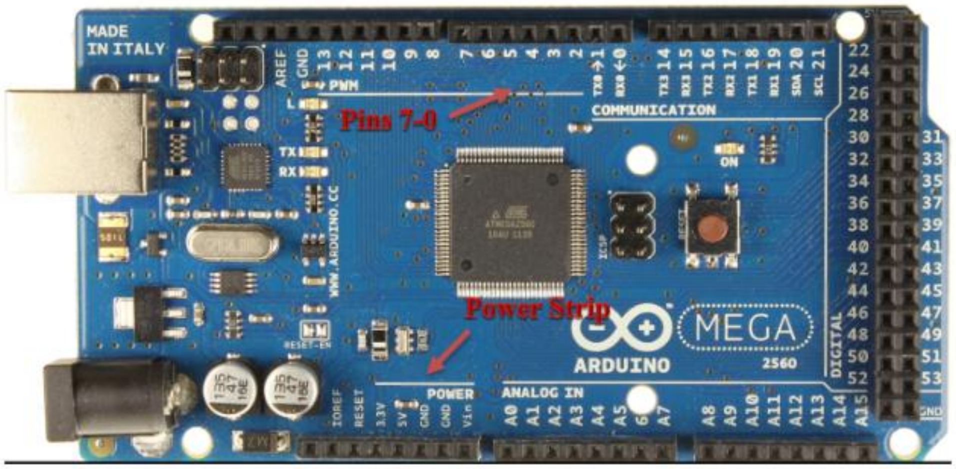
```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

3. You will notice in the "Blink" program that pin "13" is set as an "OUTPUT" in this program. This means that the Arduino can be used to set the voltage on this pin. In the "loop" section of the program you will notice the output goes "HIGH" first. This will turn on the LED with a 5 V signal. Next you will see the "delay(1000)" command followed by pin 13 being set "LOW". This causes the program to pause for 1 second after turning on the light, and then turn it off, sending 0V to the LED. The Program than pauses a second time for 1 second and turns the light back on in a loop causing it to blink on and off for 1 second each time.

4. Change the number of the pin from 13 to 7 in the program for all commands. Verify the program and upload it to the Arduino once you have made these changes. Once it is successfully uploaded get the Arduino trainer either unplug the Arduino USB cable and connect the Arduino power cable or leave it plugged in to the USB cable to continuously supply power. Next, hook up

the trainer's "POWER" plug to the "POWER" strip of the Arduino. Also hook up the module 1 plug labeled "1" to pins 7-0. You will notice two pins missing from the "POWER" plug. Looking at the "POWER" label and the outside edge of the Arduino, the pin to the farthest right should connect to the "Vin" hole of the Arduino. Again with the "1" facing you and the other outside edge of the Arduino facing you, the pin to the farthest right should go into the "0" pinhole of the Arduino. If done correctly the green LED in the first column should begin blinking on and off.

Figure 12: Arduino Mega 2560 Power and Pin Locations



5. If the bottom left, green LED was blinking then it is time to write the stop light simulation. Disconnect all plugs from the Arduino except the USB cable if you were using it to supply power. If not, re-connect the USB cable and ensure the Arduino is communicating. WARINING: If you do not disconnect the "POWER" and "1" cables from the training module your program may not load correctly. If anything is connected to the pin inputs of the Arduino while uploading a sketch program it can cause an error to occur that does not allow it to upload.

6. In the setup of the sketch program set digital pins 7-0 as OUTPUTS using the "pinMode" command. The LED lights correspond to these pins as follows:

Arduino Pin	**LED Light**
7	Column 1 Green
6	Column 2 Green
5	Column 1 Yellow
4	Column 2 Yellow
3	Column 1 Red
2	Column 2 Red

7. To create a stop light simulation you must alter the Blink sketch to change the delays and pins which go high or low. For the purpose of more accurately simulating a stop light, the green of one light should be set to be on for 12 seconds, yellow for 3 second, and red for 15 seconds. While the first stop light is going from green to yellow the other stop light must stay red until the first light becomes red. At this time the second stop light should become green and this process should repeat with no delays where all lights are off. Both lights should function with the same delays. In other words if the program would cause people to crash in real life then your imaginary people will surely sustain injury.

8. Once the program has been completed, verify it to check it for any syntax errors, and upload it to the Arduino. Reconnect the Arduino to the trainer making the same connections as in step 4. Watch the LEDs to verify that you have made a successful sketch that will allow your imaginary people to commute safely and efficiently. Once the sketch has been done correctly save and turn in your work.

**** Challenge: This lab can be written using only 4 "delay" functions and 12 "digitalWrite" functions where each pin only changes state two times. Can you figure it out?**

Lab 3: Inputs and Outputs

Purpose

This lab will teach you how to turn on and off digital outputs using external electronic components such as buttons, switches, or any other digital device. This is done by reading the state of a digital pin specified as an input and through conditional "if" statements. "if" statements allow for the Arduino to respond to specific situations (conditions) by continuously checking to see if the specified conditions are satisfied if they are, the portion of the program contained within the if stamen is run. Integers will also be introduced. These allow for a particular number or value to be assigned to a single character or string of characters. This can be useful in making programs more clearly understood, or to program with a value that changes based on conditions. This object of this lab is to make the 4 red LEDs turn off and on with the corresponding buttons on the trainer. This will be done using the commands introduced in this section and any necessary from previous sections.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Power Cable (Optional)

Arduino Trainer

Commands and Description

Command	**Description**
int *NAME = VALUE* *NAME*- Any letter or letters not already used for a command *VALUE*-Any number between -32,768 and 32,767 or const int *NAME = VALUE*	This command is used **before** the setup portion of the program and all integers should be declared using this command before anything else is written in the program. Their *"NAME"* can be any character or string of characters not already used as a sketch command. Their value can be any whole number between -32,768 and 32,767. For example "int LED = 5" will put the value "5" anywhere "LED" is written in the program. Adding "const" keeps the specified integer constant throughout the program and it will not change. This is useful in declaring pin numbers.

if (*VARIABLE OPERATOR VARIABLE) {*
}
else {
}
VARIABLE- Any previously defined integer, pin, or number
OPERATOR- Any symbol used for mathematical comparison: <, >, ==, !=
	Used to establish a conditional expression. If the condition is satisfied, the operation or program written between "{" and "}" will run. The "else" command can be used if only two actions are to be performed in any case. Examples of the usage of this statement are presented after this table to provide better understanding. This is one of the most important functions in programming

digitalRead(#)	This is similar to the "digitalWrite" command in that the value can either be "LOW" or "HIGH". The "#" is the number of the pin you wish to read from.

More on "if" statements:

The variables can either be numbers, states (such as HIGH or LOW), or integers and the operators can be any standard mathematical comparison as long as the syntax is correct. Because the "=" (equals sign) is already used to assign values, such as with the "int" function "==" is used as the "equals to" comparator. Similarly "!=" is used for "not equal". The "!" symbol is used as a Boolean operator to mean not. Other Boolean operators are "||" for "or", and "&&" for "and". Examples of the proper syntax for the comparison and Boolean operators are listed below.

Command Expression	**Result**

if (x == y) {
digitalWrite (7, HIGH);
}
	If "x" is equal to "y" then digital output 7 will be set high (sent a 5 V signal).

if (x > y) {
digitalWrite (7, HIGH);
}
	If "x" is greater than "y" then digital output 7 will be set high (sent a 5 V signal).

if (x < y) {
digitalWrite (7, HIGH);
}
	If "x" is less than "y" then digital output 7 will be high (sent a 5 V signal).

if (x >= y) {
digitalWrite (7, HIGH);
}
	If "x" is greater than or equal to "y" then digital output 7 will be set high (sent a 5 V signal). This also works for "<=" (less than or equal to).

if (x != y) {
digitalWrite (7, HIGH);
}
	If "x" is not equal to "y" then digital output 7 will be set high (sent a 5 V signal).

if (x > y && y < 3) {
digitalWrite (7, HIGH);
}
	If "x" greater than "y" and "y" is less than "3" then digital output 7 will be set high (sent a 5 V signal).

if (x == y || x == z) {
digitalWrite (7, HIGH);
}
	If "x" is equal to "y" or "x" is equal to "z" then digital output 7 will be set high (sent a 5 V signal).

if (x == HIGH) {
digitalWrite (7, HIGH);
}
else {
digitalWrite(7, LOW);
	If "x" is a digital input set "HIGH" (has 5 V on it) then digital output 7 will be set high (sent a 5 V signal) or else (any other time) digital output 7 will be set low (sent a 0 V signal).

Procedure

1. Open up the Arduino sketch program and connect the Arduino to the computer. You can begin by opening the "BareMinimum" example introduced in lab one, or any other sketch you find helpful to build off of. Start by defining your variables. You will be using the trainer module 2 plug in this lab which is connected to 4 buttons and 4 LEDs with the labels "B0, B1, B2, and B3". You may wish to name your integers "buttonB0" and "ledB0" or something else that makes these easily identified. These should be set equal to the pin they will be plugged into. When looking at the label of the plug and the outside edge of the Arduino Mega, the plug will go into pins 22-36, and their corresponding buttons and LEDs are:

Purpose	**Arduino Pin**
buttonB3	24
buttonB2	22
buttonB1	26
buttonB0	28
ledB3	30
ledB2	32
ledB1	34
ledB0	36

2. Next make all the buttons inputs and all the LEDs outputs using the "pinMode" command from lab 2. If you have used the same integers as the lab suggests, you can write "pinMode(buttonB3, INPUT)" rather than writing the pin number.

3. Using the "if" statements presented in this lab write a sketch that digitalReads the button pins, and digitalWrites the corresponding LED to come on. In other words, once you load your program and connect it to the proper plugs on the trainer, button B0 should turn on LED B0, button B1 should turn on LED B1, ect. The LEDs must also turn off after the button is released.

4. Once you have written your program, save it, verify it and check for any errors. When none are found upload it to the Arduino. If you are leaving the Arduino connected to the USB cable, simply plug in the "2" plug into the pins mentioned in step 1. Otherwise, connect the Arduino power cable once it is disconnected from the computer to supply power to the circuit. In this lab you may wish to use the "Serial.print" command to monitor the state of any buttons you press. A code similar to the following example can be used to monitor the state of the buttons. This code will be inserted into the loop section of the program, and will have to be written for every button. Also remember to initialize serial communication in the setup of your sketch.

Example:

```
if (digitalRead(buttonB0) == HIGH) {

        Serial.print("B0 is HIGH");

}

else {

Serial.print("B0 is LOW");

}
```

5. Once you have a working program save your sketch, and turn in your work. This sketch will be built upon in future labs, so make sure it is commented well and you can clearly understand all integers and functions used in the program.

Lab 4: Better Serial Communication

Purpose

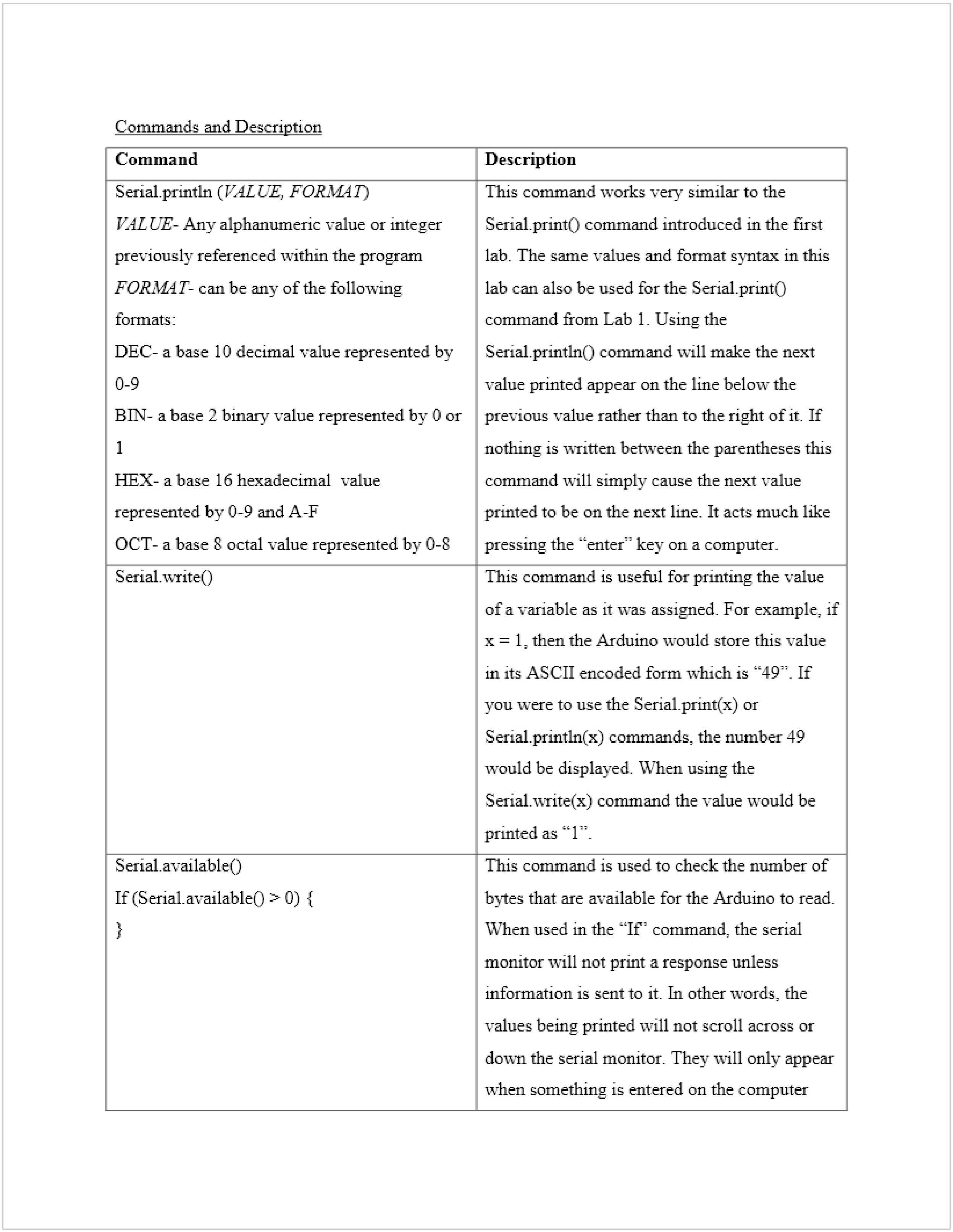
In this lab you will learn to better your skills at interfacing a computer with the Arduino using the serial commands introduced in the first lab, and some new serial commands that will allow for better communication between the two. You may remember that in Lab 1 the phrase entered would scroll continuously in the serial window. This lab will show you how to print serial commands only when prompted to. You will also learn how to do some basic math with the Arduino to translate between standard ten digit decimal format that you count in and the two digit binary format that computers use. As you may know all digital electronics, such as computers, communicate using a binary system; that is each individual bit may be either a "1" or a "0". You may be surprised to know that when your press the "!" key on a keyboard it is translated to a binary number through a language known as ASCII. In the ASCII language "!" corresponds with the binary number "100001" which is the number "33" in a base ten system that you count in. Because "!" is represented by the lowest value in the ASCII table (100001 or 33) all numbers 0-9 are represented by a value greater than the number being typed in. For example the number "1" is represented in the computer by the value "110001" which is the decimal value "49" in a base 10 system. This may sound quite confusing, and memorizing all of these values would take quite a bit of time, so today you will learn how to make the Arduino do this for you.

Equipment

Arduino Mega

Arduino USB cable

Computer



Commands and Description

Command	**Description**
Serial.println (*VALUE, FORMAT*) *VALUE*- Any alphanumeric value or integer previously referenced within the program *FORMAT*- can be any of the following formats: DEC- a base 10 decimal value represented by 0-9 BIN- a base 2 binary value represented by 0 or 1 HEX- a base 16 hexadecimal value represented by 0-9 and A-F OCT- a base 8 octal value represented by 0-8	This command works very similar to the Serial.print() command introduced in the first lab. The same values and format syntax in this lab can also be used for the Serial.print() command from Lab 1. Using the Serial.println() command will make the next value printed appear on the line below the previous value rather than to the right of it. If nothing is written between the parentheses this command will simply cause the next value printed to be on the next line. It acts much like pressing the "enter" key on a computer.
Serial.write()	This command is useful for printing the value of a variable as it was assigned. For example, if x = 1, then the Arduino would store this value in its ASCII encoded form which is "49". If you were to use the Serial.print(x) or Serial.println(x) commands, the number 49 would be displayed. When using the Serial.write(x) command the value would be printed as "1".

Serial.available()
If (Serial.available() > 0) {
}
	This command is used to check the number of bytes that are available for the Arduino to read. When used in the "If" command, the serial monitor will not print a response unless information is sent to it. In other words, the values being printed will not scroll across or down the serial monitor. They will only appear when something is entered on the computer

Serial.read()	This command is used to read the serial data available to the Arduino. It can be used in the form "x = Serial.read()" which will assign any number or key entered into the serial monitor to the variable "x". If you were to use this command and press "1" on the computer and send that information to the Arduino, "x" would equal "1"

You will need to do math within the sketch and while the operators are relatively intuitive, they are listed below for easy reference. Brackets such as "( )" and "[ ]" may also be used to better define the order of operations.

Mathematical Operators:

Operator	**Description**
+ Example: z = x + y	Addition z "equals" x "plus" y
- Example: z = x - y	Subtraction z "equals" x "minus" y
* Example: z = x * y	Multiplication z "equals" x "times" y
/ Example: z = x / y	Division z "equals" x "divided by" y
( ) Example: z = (2 + 3) * y	Brackets z "equals" 2 "plus" 3 (or the value 5) "times" y

A Little about Binary:

As mentioned in the Purpose section you have been taught to count in a base 10 decimal system. That means there are 10 digits used to express a given value. These digits are 0-9. Binary on the other hand uses only two digits to express a given value. These digits are 0 and 1. Computer and other electronic devices use this base 2 system so values can be expressed by a state of on or off, high or low, 1 or 0. When a specific input or output is on or high this registers as a 1, while off is 0. In order to translate between binary or any other language and decimal a simple equation can be used. Each digit in a binary number holds the value of 2^n where n represents the place of the digit. The first digit in binary is counted as 0, the second is 1, the third is 2, and so on. The digit farthest to the right is the LSB or the first digit. Look at the number 0001. There is a 1 in the first digit so $2^0 = 1$ so this is the number one. Now look at the number 1011. To solve this all you need to do is add up the value of each digit that has a 1 in it. To add them up count the place of digits starting with 0 and you get the equation:

$$2^0 + 2^1 + 2^3 = 1 + 2 + 8 = 11$$

Therefore 1011 is the number eleven. Can you figure out what 1100 is?

Procedure

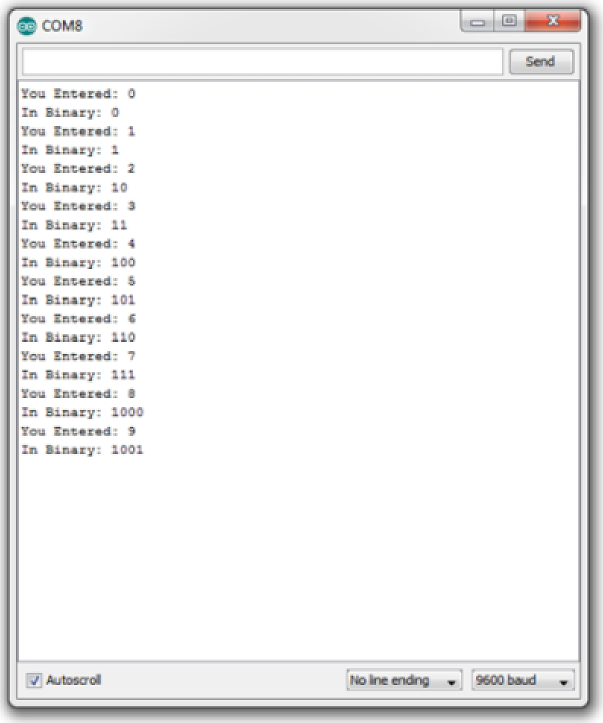
1. Begin by plugging in the Arduino to the computer, and opening up the Arduino sketch program. Also declare a variable to store the value you wish to enter. This can be entered before the setup and should hold the initial value of 0. For example "int x = 0" could be used.

2. Next enter the command to begin serial communication in the setup portion of the program. In order to prevent the constant scrolling that occurred in lab one, enter in the "if" command presented in the "*Commands and Description*" section of this lab.

3. Enter in a command that will assign any value entered into the serial monitor to the variable you chose. This command should be entered after the "if" command but before any other data. This will make the Arduino assign the entered value to the variable first when going through the loop portion of the program.

4. Use the "Serial.print()", "Serial.write()", and "Serial.println()" commands to produce the following table in the serial monitor:

Figure 13: Binary Translation Using the Serial Monitor



```
You Entered: 0
In Binary: 0
You Entered: 1
In Binary: 1
You Entered: 2
In Binary: 10
You Entered: 3
In Binary: 11
You Entered: 4
In Binary: 100
You Entered: 5
In Binary: 101
You Entered: 6
In Binary: 110
You Entered: 7
In Binary: 111
You Entered: 8
In Binary: 1000
You Entered: 9
In Binary: 1001
```

Remember that the Arduino will store the entered value in its ASCII encoded value. In other words you will need to do some math to get the entered value of 1 to actually display as the value of 1 in binary. The easiest way to do this is begin by entering in the number "0" and see what number is displayed in the serial monitor. Once you know this value, subtract it from your variable before printing the number in binary. Without subtracting the appropriate value the number 1 will be displayed as "110001" in binary rather than simply displaying "1".

5. You will most likely need to try several different ways of using the three “print”, “println”, and “write” commands to generate the above table correctly. Use the phrase “You Entered: ” as a label for the value entered when displaying it as a base 10 number. Use the phrase “In Binary: ” to display the translated binary value of the number entered.

6. Once you can enter in the numbers 0-9 and get their corresponding binary values generated in the serial monitor exactly as they appear in figure 7 you have done the lab correctly. You may need to load several programs and try this several times to get your table to match figure 7.

****Challenge: Once you have gotten the binary translation to display correctly add the commands to your sketch to display the number in hexadecimal and in octal. These must be displayed in the same organized way as in figure 5.**

Lab 5: Programming a Keypad

Purpose

Now that you have translated the numbers 0-9 into binary numbers you will learn how to use this knowledge to interface the Arduino with a keypad. The keypad on the Arduino trainer is a very simple ten digit key pad with an enter button. Using the "if" statements already learned and the binary table produced in the last lab you will program the Arduino to interpret the keypad as an input causing the "B0-B3" LEDs to illuminate. These four LED will represent the binary numbers with a "1" being represented by an illuminated LED and a "0" being represented by an LED which is off. LED "B0" is the least significant bit and LED "B3" is the most significant. The keypad numbers correspond with the binary numbers they represent for the most part. These keypad numbers are connected to four pins on plug number 3 forming a four bit input for the Arduino. While the goal of this lab is to interpret the signals from the keypad and translate them into usable data the purpose of this exorcise is to learn how to use multiple and somewhat complex if statements. This lab will focus on the program created rather than the end result achieved.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Power Cable (Optional)

Arduino Trainer

Commands and Description

The commands presented in the previous labs are used, and no new commands are necessary to perform the actions required by lab 5.

Procedure

1. This lab contains both inputs and outputs, and will use the "POWER" plug from the trainer as well as the module 2 and module 3 plugs. The "POWER" plug is to be connected to the "POWER" strip of the Arduino the same way it was connected in step 4 of Lab 2. That is the two empty pins will hang off of the left side of the "POWER" strip with the label facing the outside edge of the Arduino. The module 2 plug will be inserted into the digital pins 22-36 of the Arduino with the "2" label facing the outside edge of the Arduino. The module 3 plug will be inserted into pins 0-7, again with the "3" label facing the outside edge of the Arduino. When inserted correctly the following pins will be used in the program as either inputs or outputs:

pinMode	**Arduino Pin**	**Purpose**
INPUT	7	Keypad Bit0
INPUT	6	Keypad Bit1
INPUT	5	Keypad Bit2
INPUT	4	Keypad Bit3
OUTPUT	30	LED B0
OUTPUT	32	LED B1
OUTPUT	34	LED B2
OUTPUT	36	LED B3

2. To begin, open up the Sketch program for the Arduino and declare the integers you will use for this lab. While declaring integers is not necessary, your program will be much easier to understand and debug if your inputs are labeled with a recognizable phrase such as "ledB0" rather than using the number 30 to declare this pin. Use the "int *NAME = VALUE*" command introduced in Lab 3 to create easily recognizable names. Example: "int ledB0 = 30;"

3. Once you have names for all of your led pins and bits, and have set them equal to the pin values in the above table, declare them as and input or output in the setup portion of the program. Because you have already named the pins you can do this by writing "pinMode (ledB0, OUTPUT)" rather than specifying the pin number.

4. To get the keypad to display the number entered in binary a series of "if" statements must be formed to recognize which button is being pressed. The numbers 1-9 will correspond with their binary values determined in Lab 4. For example the number "1" is represented as "0001" in binary. Therefore, when "1" is pressed on the keypad, "Keypad Bit 0" will be high, and all other keypad bits will be low. When this is read by the Arduino, LED B0 should illuminate and all other LEDs should remain off.

5. You may need to investigate this more before you continue programing. To better illustrate how the keypad corresponds with the lights enter in the following command in the loop section of the sketch:

```
if (digitalRead (Bit0) == HIGH) {

            digitalWrite(ledB0, HIGH);

}
```

The code provided only works if you have labeled your Keypad Bit0 as "Bit0" and your LED B0 as "ledB0". You may have to change these integers to something different if you did not name them the same. Duplicate and enter this code for all bits "Bit0" to "Bit3" and all LEDs "ledB0" to"ledB3" forming a series of four "if" statements. Finally end the loop portion of the program with the following code:

```
if (digitalRead(Bit0) == LOW && digitalRead(Bit1) == LOW  && digitalRead(Bit2) == LOW && digitalRead(Bit3) ==
LOW) {   // if nothing is pressed

  digitalWrite(ledB0, LOW);

  digitalWrite(ledB1, LOW);

  digitalWrite(ledB2, LOW);

  digitalWrite(ledB3, LOW);
}
```

6. The last bit of code entered will turn off the LEDs once a keypad number is released. Once again you may need to change the integer names. Verify and upload this sketch to the Arduino Mega and plug in the connections from step one if you have not already done so. If this is done correctly you will notice pressing a number on the keypad (1-9) will illuminate the LEDs that represent that number in binary.

7. But wait! What about "0" and "ENT"? You will notice that when you press the "0" button on the keypad LEDs B3 and B1 will illuminate giving the binary value of "1010" which is the number "10" in a base ten counting system. This is necessary because the LEDs must turn off when no keys are being pressed. The keypad was wired up with "0" as the binary value "1010" so that the Arduino could register this key as an input, otherwise the default state of the keypad (when no buttons are being pressed) would constantly register as the number "0" if given the binary value "0000". The "ENT" button (short for Enter) was given the binary value of "1100" or "12" in a base ten system.

8. In order to display the value of "0" all of the LEDs should be off. However, for the purpose of distinguishing this from simply not hitting a button your finished sketch should display the value of "0" by illuminating all four LEDs giving the binary value of "1111". Enter may remain the default binary value.

9. The keypad in the Arduino trainer was designed for simplicity and easy interface with the Arduino. Most keypads purchased from an electronics supplier are not as simple, and therefore require much more complicated code. Currently the code you have in your sketch only turns on an output LED when the corresponding input keypad button is pressed. If the keypad was wired directly to the LEDs it would perform the same way without the Arduino. Since you are trying to learn programming rather than circuitry, completion of this lab will not be defined by observation of the outputs. Instead you must create a program that accounts for all possible inputs using "if" statements similar to the last one given in step five. In other words you need to write 11 "if" statement (one for each button). There will be a total of 12 "if" statements if the program is done correctly. One for each button that can be pressed and the one given in step five that will turn off the LEDs when nothing is being pressed.

8. Once you have a total of 12 "if" statements that account for the state of each of the four keypad bits (either HIGH or LOW) compile and load your sketch into the Arduino Mega and retest your circuit. The LEDs should illuminate the same as before for the when the keypad numbers "1-9" are pressed. When "0" is pressed all of the LEDs should illuminate, and when "ENT" is pressed the binary value "1100" should be displayed by the LEDs. This program will be useful for interfacing with more complex keypads in later lab assignments.

Lab 6: Reading Analog Inputs

Purpose

Now that you understand the basics digital inputs and outputs it's time to learn how to use analog inputs. Digital inputs on the Arduino Mega consist of either a 0 V or 5 V signal representing either on or off. Analog inputs on the other hand can take any value between 0 V and 5V. The Arduino does this by converting a voltage level to a digital number that is determined by a 10 bit analog to digital (A/D) converter. In the previous labs dealing with binary you programed a keypad based on a four bit system with a maximum number being represented by "1111" or the value of 15. The A/D converter on the Arduino Mega can hold any value from 0-1023 (0000000000-1111111111 in binary). This means that the voltage level on an analog pin can be represented by a total of 1024 different values when the value of 0 is included. While the voltage value is not exact this provides a very good approximation of the actual analog value. In this lab you will use the potentiometer on the Arduino trainer to vary the voltage on one of the analog inputs and display the measured value in the serial monitor. Interpreting analog signals is highly important to interfacing the Arduino with sensors and other devices.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Trainer

Commands and Description

Command	**Description**
analogRead(#)	This command reads the voltage level on an analog pin and converts it to a number from 0-1023. The "#" is the number of the analog pin to be read. This command is the analog version of digitalRead(#) and can be used similarly. For example "analogRead(7)" will read the value from analog pin 7.
float *NAME = VALUE*	This command is the same thing as the "int *NAME = VALUE*" command except it allows the values to be fractional numbers. In other words, this command is used when a decimal is needed. This is useful when the math done within a sketch will not yield whole numbers. A float has 6-7 digits of precision and can account for much larger or smaller numbers than the integer command.

What is Resolution?

Different A/D converters have something called resolution. The resolution is what determines the accuracy of the A/D converter. An analog signal is a signal that can take any value, but the Arduino uses binary for calculations so it cannot give a truly exact value. Think of 12 in spool of string. You can unroll the spool of string and cut of any amount between 0-12in. When you measure the string with a ruler you will not actually know the exact length of string, however. You can come close, but you are limited by the number of line divisions on the ruler. If the string does not fall exactly on one of the lines you simply round up or down to approximate the length. If there are 8 divisions in an inch on the ruler, the resolution of the ruler is 1/8 in or 0.125 in.

The resolution of an A/D works in the same way. The Arduino A/D converter can measure 0-5V and can represent this value with 1024 numbers. This can be calculated by dividing the maximum voltage that can be measured by the number of possible binary values. In this case you divide 5 V by 1024 gives the value 0.0048828 V. The resolution is therefore about 4.9 mV per bit. The exact value will not be determined by the Arduino will tell you a value with an accuracy of 4.9 mV

A New Trick with "if" Statements:

By now "if" statements have been used a wide variety of ways to monitor the current state of variables and digital pins on the Arduino within the previous sketches. All of the comparison and Boolean commands shown in Lab 3 work for analog inputs excluding the ones where "HIGH" or "LOW" are used, as these values are constants used specifically for digital inputs. To add to the already massive list of possible "if" statements there is one more that needs to be introduced for successful completion of this sketch. The "if" statement is used to monitor a change in state on an input, output, or a variable. In Lab 5 pressing a keypad button illuminated the corresponding LEDs to display this number in binary. If the same statements were to be used to "Serial.print()" these values to the serial monitor the values would continue to scroll until the button is released. It would be much more practical to monitor a change in state on these inputs to print the value only once rather than continuously have them scroll down the screen. You will encounter a similar problem in this lab when you print the values from the analog pin. The following code can be used to monitor a change in state rather than simply responding to the state of an input. For the purpose of this example digital pin 7 will be considered the input and has a push button connected to it. The code presented is a simplified version of the "StateChangeDetection" example in the digital examples provided with the Arduino software.

```
int State = 0;          // used to monitor the current state of the button

int LastState = 0;       // used to monitor the previous state of the button


void setup() {
  pinMode(7, INPUT);        // digital pin 7 is the input
  Serial.begin(9600);
}
void loop() {
  State = digitalRead(7);    // this will remain "LOW" or "0" until the button is pressed
  if (State != LastState) {  // once the button is pressed "State" will be "HIGH" but "LastState" will be "LOW"
showing a change
    if (State == HIGH) {     // if the state went from "LOW" to "HIGH" the button has been pressed
      Serial.println("Button State Changed High");  // "Button State Changed High" will be printed once
      State = LastState;     // sets the new state of pin 7 to the old state to continue to monitor change
    }
      else {               // otherwise if the state goes from high to low (button was released)
        Serial.println("Button State Changed Low"); // "Button State Changed Low" will be printed once
        State = LastState;     // sets the new state of pin 7 to the old state to continue to monitor change
    }
  }
}
```

Often times, there is no response programmed for when the button is released and that portion of the code can be omitted.

Procedure

1. Begin by opening the Arduino sketch program and starting a new sketch, or example you prefer to begin from. Choose and name three variables. You will need two of these to monitor the analog input. One will be used for the current state of the analog input and one will be used

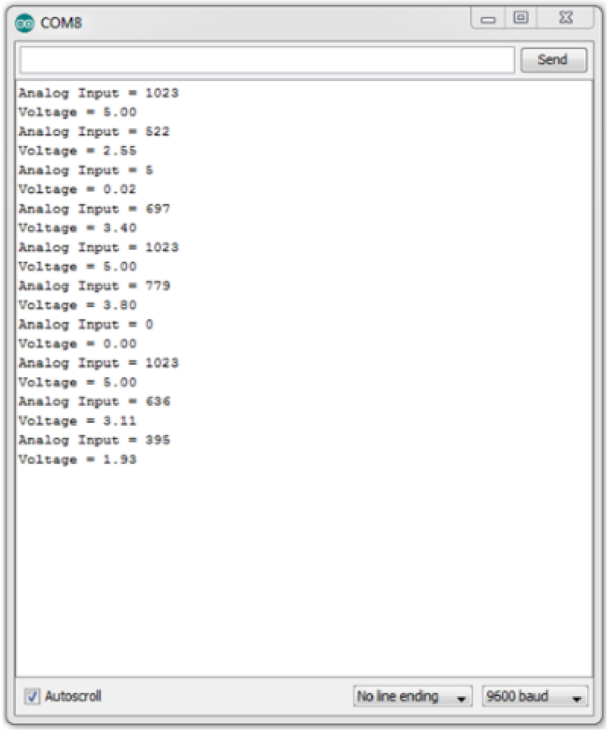
for the previous state. You will also need one of the variables to be a "float" that will allow for the voltage calculation. For the purpose of providing a better explanation in future steps the current state of the analog pin will be the variable "AnalogInput", the previous state will be the variable "AnalogChange" and the variable used for voltage calculation will simply be "Voltage". You may choose any name you prefer. You may also wish to name the potentiometer pin as a variable, rather than referencing the pin by its board number. "PotPin" is a pretty good name for that.

2. You will be using the "POWER" plug as well as the module 5 plug labeled "5". When you are ready to test your sketch the module 5 plug will connect to pins "A0-A7" with the label facing the outer edge of the Arduino Mega. This connects pin "A7" to the potentiometer "PotPin" of the Arduino trainer (the shiny metal knob). In the setup of your program begin serial communication and set pin "A7" as an input.

3. It may be easier to write this sketch in two steps. First by programming the appropriate serial response and math, then by adjusting the program to use state change detection. The first thing that should occur in the loop portion of the program is the "AnalogInput" should be read from the PotPin. Next this value should be multiplied by "0.0048828" (the resolution) converting the value to "Voltage" giving the voltage measurement on this pin.

4. Next, both of these values need to be sent to the serial monitor in and easy to read and understandable way. Rather than just displaying the raw values of these variables, use labels to properly demonstrate what is being measured. The following figure is presented to show what should be read in the serial monitor. The labels may be called anything you prefer as long as it makes the two variables easy to distinguish from each other.

Figure 14: Analog Input and Voltage



```
Analog Input = 1023
Voltage = 5.00
Analog Input = 522
Voltage = 2.55
Analog Input = 5
Voltage = 0.02
Analog Input = 697
Voltage = 3.40
Analog Input = 1023
Voltage = 5.00
Analog Input = 779
Voltage = 3.80
Analog Input = 0
Voltage = 0.00
Analog Input = 1023
Voltage = 5.00
Analog Input = 636
Voltage = 3.11
Analog Input = 395
Voltage = 1.93
```

5. You may wish to upload your sketch at this point and test to make sure the values are being printed correctly, and the "Voltage" value is being calculated correctly. Use a calculator to double check. The values should print continuously at this point because state change detection has not yet been added.

6. Next you will prevent the constant scrolling of values and just print a new value when the knob has been twisted and the voltage is changed. This is done by producing an "if" statement that compares the current "AnalogInput" value to the previous "AnalogChange" value. If you did test the sketch in step 5 you should have noticed slight fluctuations in the "AnalogInput" value even when the knob was not touched. This is caused by very slight voltage instability, and can be caused by many different factors. Your program will need to compensate for this change, and should not constantly print out values because of this instability. For example the statement

“if(AnalogInput > AnalogChange) {“ would not solve this issue. You must add a second qualifier to this statement that executes the code between the brackets only when the knob has been turned and the value actually changes. A small bit of math will solve this issue. Try using a qualifying statement that only prints the vale if the change is *greater than or less than* the previous value by an amount not affected by the voltage instability.

7. Once you have the code to only print a change in value, there is one last thing that needs to be corrected. A “delay()” must be inserted in the code to pause the program for a second or two so the serial monitor only displays the new changed value of the “AnalogInput”. This delay should be activated by a significant change in the analog voltage. When the delay is in the correct place the serial monitor will only display one value every second or two. If this portion of the code is not inserted correctly and the voltage is increased from 2 V to 3V, the serial monitor will print every value of voltage between these two values, or it will print the value of the voltage before the knob is done being twisted. Preventing this from happening makes the values much easier to read.

8. Once the sketch has been completed upload it to the Arduino and test to make sure all requirements function as they should. Once you have observed the proper output in your serial monitor you have completed the lab correctly.

Lab 7: Dimming a Light Using the Keypad

Purpose

In the last lab you discovered how to interpret analog signals on the Arduino. In this lab you will learn how to use analog outputs to control the lighting level of a LED. The Arduino does this using pulse width modulation (PWM). The analog inputs cannot be used as analog outputs. Instead, the Arduino has several digital pins labeled with "PWM". Any of these pins can be used to generate a variable voltage. To do this you will modify Lab 5 to vary the voltage to the ultra-bright LEDs on the Arduino trainer. Each number pressed will correspond with a different voltage and therefore a different observed brightness.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Power Cable (Optional)

Arduino Trainer

Commands and Description

Command	**Description**
analogWrite(#, Value)	This sets the voltage level on one of the digital PWM pins. The "#" is the number of the digital PWM pin to be written to. The "Value" must be set to a number between 0 and 255 where 0 is off all the time and 255 is full on. A number between these two values will set a voltage between 0 V and 5 V.

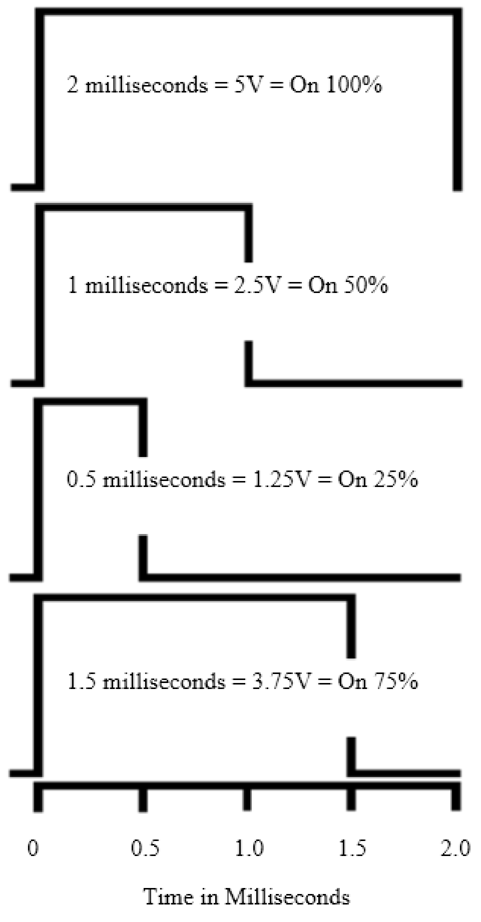
Pulse Width Modulation (PWM):

PWM is a way of regulating voltage by rapidly switching an output on and off. For the Arduino the modulation occurs at about 490 Hz. Hz stands for Hertz which is a measurement of on and off cycles per second. In other words 1 Hz is 1 cycle per second, or the signal is off for 0.5 seconds and on for 0.5 seconds totaling 1 second. When a signal is turned on and off very rapidly the voltage "felt" by the circuit or electrical device is the average voltage produced over a period of time. While this may sound complicated it can be easily understood by using percentages. If a 5 V signal is turned on 100% of the time the voltage "felt" by the device will be 5 V. Similarly if the voltage is turned on 50% of the time and off the other 50%, the voltage will be 2.5 V. In this lab the voltage will be observed through the brightness of the LEDs. The on and off switching happens so fast that it cannot be seen, and appears to be dimmed when the on pulse is reduced.

A good example of how this occurs is the operation of a television. The image viewed is not a constant stream of light, but rather a function of lights changing very rapidly to appear as a continuous signal. Most US televisions have a refresh rate of about 60 Hz. When watching the TV you do not notice it flickering 60 times a second, but rather the image produced by a change over a short period of time.

The modulation uses a square wave and the "width" referred to is the width of the on and off pulses in milliseconds. For a frequency of 490 Hz the signal refreshes about every 2 milliseconds, which is known as the period. The period of a frequency is the amount of time it takes to complete one cycle from on to off again. The period of a 60 Hz signal is given by the equation Frequency = 1/Period. The figure below illustrates this idea of Pulse Width Modulation.

Figure 15: PWM with Square Waves


2 milliseconds = 5V = On 100%
1 milliseconds = 2.5V = On 50%
0.5 milliseconds = 1.25V = On 25%
1.5 milliseconds = 3.75V = On 75%
0
0.5
1.0
1.5
2.0
Time in Milliseconds

<u>Procedure</u>

1. Begin by connecting the Arduino to the computer via the USB cable and open the sketch for Lab 5. You will need to add two more integers to the sketch to control the ultra-bright LEDs. These LEDs will be connected to pins 6 and 5. You will also need to change the pin values for the keypad buttons. If you used the same naming convention suggested in Lab 5 these will be

called “Bit0, Bit1, Bit2 and Bit3”. These will be connected to digital pins 14, 15, 16, 17 in order from least to greatest. This lab you will use the module 4 plug and the module 3 plug. The module 2 plug is optional. If you choose to use the module 2 plug it will be connected the same way as it was connected in Lab 5. That is the plug will be connected to digital pins 22-36 with the “2” label facing outward.

2. You will need to modify lab 5 by adding two “analogWrite” commands to each if statement. That is two commands for each button pressed. You can leave the “digitalWrite” commands for the LEDs if you would like to also connect the module 2 plug to better observe the circuit. Otherwise all of the “digitalWrite” commands can be deleted if you choose not to use the module 2 plug.

3. The keypad has 10 numbered buttons and an enter button which will be used to turn off the Led’s. The maximum value for “analogWrite” is 255 for a full 5 V signal. Each number pressed should increase the voltage on the LEDs with “1” being the least bright and “0” being the brightest (0 will be used as 10 in this lab). Each number should increase the voltage an equal amount in 10 total steps. Therefore pressing 1 will write the value of 25.5, and 0 will write the value of 255. Pressing “ENT” will write the value of 0 to turn off the LEDs.

4. Once you have the appropriate commands in your sketch and it has been modified to vary the voltage from the PWM digital pins, you can upload your sketch to the Arduino and connect the module 3 and module 4 plugs. Module 4 plug will be connected from digital pins 0-7 with the label “4” facing the outside edge of the Arduino. Module 3 plug will be connected to digital pins 14-21 with the label “3” also facing outward.

5. To test your sketch press the buttons in order from 1-9 and observe to see if the LEDs become brighter each number pressed. This may become less noticeable with the higher numbers. The enter button should turn the LEDs off and the 0 button should turn the LEDs full on. If this works as expected you have completed the lab correctly.

Lab 8: Maintaining Lighting Levels Using a Photoresistor

Purpose

Now that you can use the Arduino with both analog inputs and outputs you will be using both of them in a sketch. In the previous lab you dimmed a light using the keypad buttons; however, most dimmers you see in houses and lighting devices use a knob (potentiometer) to control the amount of light rather than a keypad. Another common use for dimming lights is to maintain a certain brightness level in an area by supplementing artificial light when natural light begins to disappear. Windows and skylights allow for sunlight to illuminate a room without the need for electricity, but the sun eventually sets, and it would be much easier if the lights just came on without you having to move to the nearest light controls to gradually make them brighter as this happens. In this lab you will be programming the Arduino to control the same LEDs used in the last lab. The "0" and "ENT" buttons will be used to select one of two different lighting modes. The first lighting mode will make the LEDs become brighter as the room becomes darker through the use of a Photo Resistor. A photo resistor is a device that has a changing resistance that varies in the presence of light. The second lighting mode will allow for manual adjustment of the LEDs through use of a potentiometer. This setup allows for both manual and automated lights in the home or work environment. You will also need to program a way to make your light sensor more or less sensitive based on the desired lighting levels for the environment.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Power Cable (Optional)

Arduino Trainer

Light Source

Commands and Description

Command	**Description**
map (*Intiger, fromLow, fromHigh, toLow, toHigh*) *fromLow, fromHigh, toLow, toHigh* = any number used for calculation	The map( ) function can be used to change the value of a variable or integer to another more appropriate value. Because the Arduino reads analog value from 0 to 1023 these are often used as the *fromLow* (0) and *fromHigh* (1023) numbers. Since analog outputs are 0 to 255 these are often used as the *toLow* (0) and *toHigh* (255). Examples are below.
constrain(*Value, Min, Max*) *Value* = any integer, variable, or other data type. *Min* = the minimum value *Max* = the maximum value	This function "constrains" a value to a particular set of numbers. In other words the *Min* value specified is the lowest number the value can take. Even if the sketch tells it to go lower the number will stay the value specified by *Min*. Similarly *Max* provides an upper limit.

More on the Introduced Commands:

In the last sketch the math to "analogWrite( )" the value of the LEDs was performed manually. The "map( )" function allows the Arduino to calculate these values for you. For example, if the integer "x" had a value set by an analog input (0-1,023), and the expression was "x = map(x, 0, 1,023, 0, 255);" the integer x would take the value of 255 when the analog input tells it to be 1023. In other words:

$$x_{final} = x_{initial} \times \frac{225}{1{,}023}$$

Similarly, if x was to take on the value 560 by an analog input, it would be set to ≈139. It is always set to the nearest whole number. If this still does not make sense look at the expression “x = map(x, 0, 100, 0, 1,000);” assuming x is an integer. In this case if x = 70, then x is set to the value of 700. If x = 65 then x is set to the value of 650 by the equation $x_{final} = x_{initial} \times \frac{10}{100}$.

It will be necessary to understand how the photo resistor in the trainer works in order to perform this lab. This resistor responds to a change in light by varying its resistance. The brighter it is the less resistance the photo resistor has, and conversely the darker it is, the more resistance the photo resistor has. This particular photo resistor has an 8K Ω resistance in the light and a 1M Ω resistance in the dark. The particular photo sensor in the trainer may go beyond these ratings in extreme darkness or extreme light, however these are the ratings for general operation. This photo resistor is in a voltage divider configuration with a 100k Ω resistor that is connected to the output that goes into the Arduino. The equation for the output voltage is:

$$V_{out} = V_{in}\left(\frac{100{,}000}{(R + 100{,}000)}\right)$$

Where “R” is the resistance of the photo resistor and "V_{in}" is the 5V input from the Arduino. If you use the 8K Ω rated value of the photo resistor (exposed to light) and solve the equation you get the output voltage of ≈4.63 V. Similarly, if you solve the equation for the 1M Ω resistance (in the dark) you get an output voltage of ≈0.45V. In other words, more light = lower resistance = higher voltage. This is all a function of Ohm’s Law.

The “constrain( )” command is useful when limiting sensors inputs or integers to a particular set of values. In this lab you will need to figure out how to set various sensitivity levels for the automated lighting. This means you will need to constrain an integer to specific values to make it more or less sensitive based on preference. Say you want to make the LEDs begin to come on when only a little amount of sunlight is lost in the room. In this case you would want to use the “map ( )” command introduced in this section to respond to a somewhat small set of values read from the analog input. Assume you discover that the analog input reads 700 when the room is only somewhat dark, and this is when you want the lights to come full on. Also assume the variable x is assigned a value by the photo sensor through the “analogRead( )” command. In this case you would need to use the command “x = map(x, 700, 1,023, 255, 0);”. This would make the device more sensitive to lighting levels in the specified range (700 to 1,025). In order to prevent an error where the analog input goes “out of bounds” so to speak, you would need to constrain the analog input from the photo sensor. In this case you would need to insert "*PhotoResistor* = constrain(*PhotoResistor*, 700, 1,023);". This prevents a number less than 700 from being read. More information will be in the procedure of this lab.

****Pro-Tip: The sketches are only going to get longer and longer. Use copy and paste as much as you can to make these things take less time. Do this without using the right-click on the mouse or by pressing the keys "ctrl+c" to copy and "ctrl+v" to paste; "ctrl+x" is the cut command. If you have the code in a previous lab, copy and paste to save time.**

Procedure

1. Begin by opening up a new sketch in the Arduino program or by opening up a previous sketch you wish to modify. The lighting levels in every room differ so you will need to discover what analog input values you will read for the particular room you are in. Lab 6 can allow you to do this without writing any code. In a separate sketch window, open up Lab 6 and modify the program to change the Analog input from "A7" for the potentiometer to "A6" for the photo resistor. Make sure you change all values of "A7" to "A6" throughout the whole sketch or your readings will not be correct.

2. Connect the Arduino to the computer and upload your modified Lab 6. Next connect the power plug from the trainer labeled "POWER" to the appropriate pins on the Arduino with the label facing outward. Also connect the module 5 plug labeled "5" to pins A0-A7 with the label facing outward. Open up the serial monitor and observe the input. If lighting is not sufficient in the room you are in, you may wish to use a flashlight or other direct lighting source to shine directly on the top of the Arduino trainer. Sufficient light should produce a voltage value of greater than 4.25V. If the maximum value in direct light is not 4.25V or greater, you will need a better lighting source. Record the highest voltage produced in direct light and the analog input value displayed in the serial monitor.

3. Next turn off the lights in the room for a moment and observe the voltage value and analog input value of the sensor in a dark room. If the lights cannot be turned off simply cup your hand around the photo resistor located above the potentiometer knob and to the left of LEDs. It's the squiggly looking thing. Once the values in the serial monitor have become somewhat constant,

record these values and turn the lights back on or remove your hand. If multiple people are sharing the same trainer feel free to share your measurements with others, as the values should not differ so long as the trainer stays in the same lighting source.

4. Now that you have your light and dark measurements close lab 6. Begin in the new sketch by creating two integers to monitor the values of the photo sensor and one for the potentiometer. Two will be used to directly record analog measurements, and the other one will be used to map the values for both modes of operation. You may wish to name these integers "PhotoResistor", "PotValue", and "x". Add the "Mode" integer to select between manual and automated lights with a default value of 0. You can also create integers to monitor the digital and analog inputs and outputs used in this lab to make your program easier to read. This is not necessary if you feel confident that you can understand the program simply by referencing the pin numbers within the sketch. If you do wish to name these pins you can use the following for easy reference and naming convention:

Pin Number	**Integer Name**	**pinMode**
14	Bit0	INPUT
15	Bit1	INPUT
16	Bit2	INPUT
17	Bit3	INPUT
A6	PhotoPin	INPUT
A7	PotPin	INPUT
6	LED1	OUTPUT
5	LED2	OUTPUT

5. In the setup of your program be sure to declare all of the appropriate inputs and outputs listed in step 4. If you want a little extra information you can also begin serial monitoring to keep track of the analog values within your sketch. Begin the loop part of your program by inserting a control statement to set the mode based on weather "0" or "ENT" has been pressed. Remember the binary value of 0 is 1010 and ENT is 1100. You can copy these from any previous keypad

sketch so long as the input pins match this one. Make the 0 button set manual controls and the ENT button set automatic controls using a conditional statement.

6. Write another statement for manual controls that reads the analog value of the potentiometer and set this to the variable you chose to store the value. This value must also be assigned to the main control variable (suggested to be "x"). Remember that "x" will be used to set the brightness of the LEDs, and that the "analogWrite" function uses the values 0-255. This is where the map command comes in handy.

7. You may wish to test your code by this point to verify you have used the map function correctly. The power and module 5 plugs will need to be connected the same as in step 2. The module 3 plug will need to be connected from digital pins 14-21 with the label facing the outside edge of the Arduino, and the module 4 plug will be connected to digital pins 0-7 with the same orientation. If it does not work correctly read the "More Helpful Information" section under the commands and pay attention to the example code and the way it is written.

8. Next create another condition statement for mode 2 or automatic controls. Create a similar section of code to assign values read from the analog photoresistor pin to the variable chosen for these values. When this mode is activated the brightness of the LEDs ("x") should be set from the photo resistor. In order for the photo resistor to function properly for the room you are in you must constrain them to the values discovered earlier for a light room and a dark room. Map these constrained values to the analog output for the LEDs. Make sure you invert the analog output values so that the LEDs come on when it is dark, rather than when it is light.

9. Once you believe the code is correct, upload it to the Arduino device and plug in the appropriate connections from the Arduino trainer the same as you did before; listed in steps six and two. If everything works correctly the LEDs should be off in the light and on in the dark. You should be able to switch between manual and automatic controls using the 0 and ENT buttons, and the potentiometer should vary the LEDs brightness.

Lab 9: Interfacing With an LCD Screen

Purpose

Up until now you have been observing any readout on the Arduino serial monitor on the computer. Often times it is much more practical to have a portable electronic device that allows you to observe various values without having to lug around a laptop or other device. In this lab you will use library files (".h" files) for the first time to interface with an LCD screen thus making the Arduino a portable device. Library files are files that you can include in sketches that allow you to program more complicated devices and tasks with ease. Rather than writing the sketch to display every number and letter with an LCD you simply load the library file and use commands specific to that library to "print" words to the screen much in the same way the "Serial.print" command works. The Arduino loads several libraries to simplify the sketches you are writing with Arduino specific commands such as "digitalWrite" and "delay()". These would otherwise take a significant amount of somewhat complicated C code. You will learn the various LCD library specific commands, and build a program that can be added onto for several useful pieces of technology.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Trainer II

Arduino 9V Power Plug (optional)

Commands and Description

Command	**Description**
#include < *NAME* >	This command includes library files in the programming of a sketch. In this lab you will include the "LiquidCrystal.h" library. This command is declared before the setup of the program. Begin this command with "#".
LiquidCrystal lcd(#, #, #, #, #, #)	This defines how the pins for the LCD screen are connected. Using the recommended connections for this lab the command will be "LiquidCrystal lcd(51, 49, 47, 45, 43, 41);". This command is declared before the setup of the program.
lcd.begin(#, #)	This command works the same way as "Serial.begin()". It initializes communication between the Arduino and the LCD screen. The first # defines the number of characters in a row, and the second # defines the number of rows on the LCD. The LCD for the trainer is 24 x 2 (24, 2). This command is declared in the setup of the program.
lcd.print()	This command works the same as the "Serial.print()" command only it prints to the LCD screen and it is not ASCII encoded. This means that "lcd.print(1)" will actually show 1 on the LCD screen rather than 49. Be sure to use quotation marks to print words otherwise the Arduino will assume the word is a variable.
lcd.clear()	This command clears the LCD screen. Be sure to include the brackets.

lcd.setCursor(#, #)
lcd.home()
	This command specifies where to set the cursor on the LCD screen. The coordinates (0,0) will print characters to the default top left of the screen. The command “lcd.home()” can also be used for this location. The coordinates (23,1) prints characters to the bottom right of the screen.

lcd.autoscroll()
lcd.noAutoscroll()
	These two commands turn on and off the auto scroll function, which prints each new command one position to the left of the cursor, and scrolls all other previously printed commands to the left.

Procedure

1. There are quite a few commands in this lab, and it is important to understand all of them, and how they affect the display of the LCD to accurately use it in future labs. Begin by opening up a sketch and putting in the first two commands introduced in this lab. The module 6 plug will be inserted from the “GND” or ground pin to digital pin 41 of the Arduino Mega with the plastic portion of the plug hanging over the outside edge of the Arduino.

2. In the setup portion of the sketch declare digital pin 53 as an output and write it HIGH to supply power to the LCD. Also include a 50 millisecond delay and “lcd.begin(24,2)” in that order. If pin 53 is not high there will be no display. Without a proper delay after writing it HIGH the LCD will not display the proper characters. You will have to do this for all future labs that use the LCD screen.

3. In the loop portion of your sketch you will simulate a conversation between the top and bottom lines of the display. This will use all of the commands introduced and show you how they work with the display. After each line “speaks” be sure to add a delay of 2 seconds so there is a

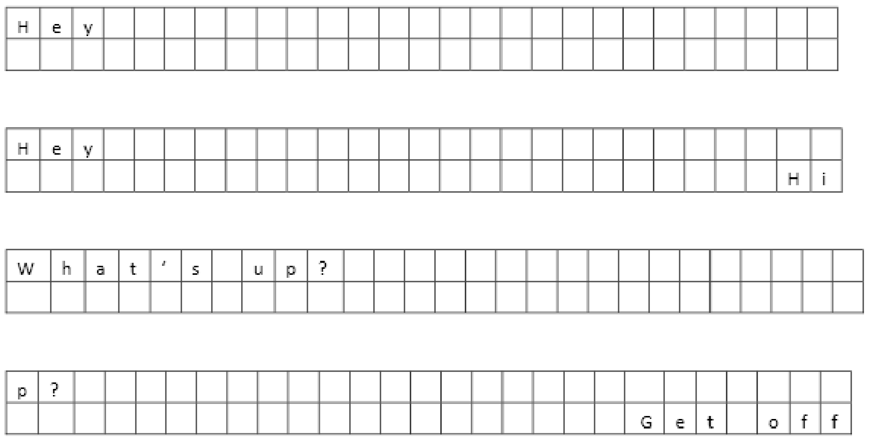
brief pause between words giving you and your instructor a chance to read what it says. First print the word "Hey" in the sketch. You will see this appear in the top left of the screen.

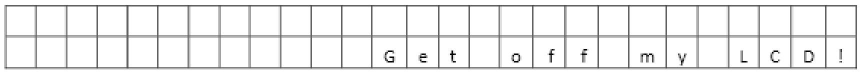
4. Next set the cursor to the bottom right hand corner of the screen and print the word "Hi". Remember to set the cursor so the letter "i" is in the corner and not the "H". After a small delay clear the screen and set the cursor in the top left hand corner again to say "What's up?"

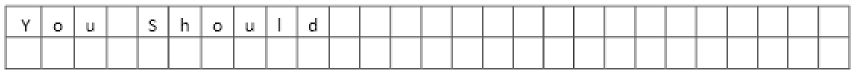
5. Turn on the auto scroll function and set the cursor to the bottom right again. This function prints 1 segment to the left of where you set the cursor, so remember to compensate for that. If done correctly the sentence will end in the bottom right. To demonstrate the scroll function, print "Get off my LCD!" with each word separated by a 0.3 second delay.

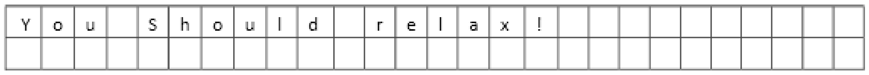
6. Clear the screen again, turn off auto scroll and set the cursor to the top left again. Print the sentence "You should relax!" with a 0.3 second delay between each word. Clear the screen one more time and you should be done. Connect the module 6 plug to the Arduino in the second trainer from the pin GND or ground to pin 41 with the wire hanging over the outside edge. Next connect the USB cable through the battery door to provide power, or connect the 9V battery plug to the power socket. If everything was performed correctly the conversation should look like this as it plays out:

Figure 16: The Conversation




Get off my LCD!


You Should


You Should relax!

Lab 10: Interfacing With a Matrix Keypad

Purpose

In this lab you will learn how to interface with the matrix keypad on the second trainer. A matrix keypad is a type of keypad that closes the circuit between two pins when a button is pressed rather than writing a pin HIGH as with the previous keypad. The keypad library will be included in this sketch, which sends pulses through the input pins and reads which output pins go HIGH to determine which key has been pressed. You will also learn about the data type "char" or character. This data type allows a number, letter, or symbol to be used in conditional statement and commands. You will also learn the byte data type. A byte is 8 binary bits that are used to store data. ASCII uses one byte of data for each character. Using the new data types and a few other library specific keypad commands you will be able to begin more complicated programming using the keypad, rather than a serial monitor, to control what prints.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Trainer II

Commands and Description

Command	**Description**

char *NAME* = # or ‘*CHARICTER*’
char *NAME*[#] ={‘*CHARICTER*’}
char *NAME*[#][#] = {
{‘*CHARICTER*’}
{‘*CHARICTER*’}
}
	The first command allows for one character to be specified by a particular name, much like integer. The ASCII number must be used to specify the character, or a single character can be placed between two apostrophes such as ‘A’. The second command is used for a single line of characters where the number specifies the number of characters. The third specifies an array of characters where the first is the number of rows, and the second is the number of columns

byte	8 bits of data used to store a single ASCII character in its binary form.
key	A library specific command used in the program to refer to a specific key
NAME.getKey()	A library specific command that is a variation of the C command “getcar()”. The name is whatever you choose to name the keypad, and signifies that the command will get the key from that device. The “.getKey()” command reads the character specified for that key and “gets” the one you press to use in your program.
Keypad	This command is used to declare the name of the keypad for use with the “.getKey()” command.

makeKeymap(*Characters), RowPins, ColumnPins, CharicterRows, CharicterColumns*)	The *Charicters* are the characters specified for the keypad array. The *RowPins* in this lab will be 30, 32, 34, 36, and the *ColumnPins* will be 38, 40, 42, 44, 46, 48, 50, 52. The *CharicterRows* and *CharicterColums* are the number of rows and columns in the character array for the keypad.

More on Keypad.h and char

Keypad.h is not a preloaded library file. In order to get the file you must download it from http://arduino.cc/playground/Code/Keypad#Download. Once you download the file you must extract the file and move it to the proper location before opening the Arduino sketch program. Extract the file the same way you extracted the Arduino software. Then copy the file into the "libraries" folder within the "arduino-1.0" folder. Once this is done you can open the sketch program and use this library with the #include<> command, and you will have access to several new examples.

This library file uses characters in unique ways. For the purpose of this sketch you will be specifying a keypad array that uses a single character for each key. First, here are some examples of how to use char. Note the word "Character" is what I have chosen to use as a name.

Command Entered	**Output on the LCD**

char Character = 'A';
lcd.print(Character);
	
	A

char Character[3] = {'A', 'B', 'C'};
lcd.print(Character[1]);
lcd.print(Character[2]);
lcd.print(Character[0]);
	
	B
	C
	A

char Character[2][3] = {
  {'A', '1', 'B'},
  {'2', '3', 'C'},
  };
  lcd.print(Character[0][0]);
  lcd.print(Character[0][2]);
  lcd.print(Character[1][2]);
	
	A
	B
	C

The keypad on the Arduino trainer is a 4x8 keypad. In other words there are four columns and eight rows in the internal circuitry. You will need to create a character array similar to the final example for your keypad to display each number or mathematical function pressed. Later you will use these characters to perform various math functions. The character array for the keypad looks like this:

```
{'^','/','9','8','7',' ','=','Q'},

{'T','X','6','5','4',' ','0',' '},

{'C','-','3','2','1',' ','A','N'},

 {'S','+',' ',' ','.','c',' ',' '},
```

Each key can only hold one character, so functions such as “Sin” must be represented by a single letter “S”. Some of the characters have been omitted and will be added in later labs. The table below shows how each letter corresponds with the button pressed. Functions with longer names have parenthesis around the brackets used to display those characters.

Cal(c)				+	(S)in
(L)ight	1	2	3	-	(C)os
(O)hm	4	5	6	X	(T)an
(D)ist.	7	8	9	/	^
(A)lpha	Clear/(N)o	0	Enter/Yes		S(Q)RT

Procedure

1. Open up your previous sketch and copy all the code used to initiate the LCD screen. These are the "#include<>" function, the LCD pins, the digital output used to power the LCD and the "lcd.begin()" commands. Also include the new keypad library "Keypad.h".

2. Before the setup of your program include two constant bytes. Declare one for "Rows" and one for "Columns". These will be used to create the character array for the 4x8 keypad. Make the character array mentioned in the *Commands and Description* section. Name the character something easily recognizable like "keys".

3. Next you will need a number array to specify the pins used by the keypad. This will be an array of bytes and is entered "byte rowPins[Rows] = {30, 32, 34, 36};". These pins are also mentioned in the *Commands and Description* section. The name "Rows" in the brackets specifies that the number of rows in the keypad is 4 from your previously declared byte. You will also need to add one for "columnPins".

4. To map the appropriate Arduino pins to the character array you created name your keypad using the Keypad declaration and specify the row and column pins. If you used all suggested variable names the code should appear as "Keypad keypad = Keypad(makeKeymap(keys), rowPins, columnPins, Rows, Columns);". If any of this is confusing you may wish to consult one of the examples that came with the Keypad to see how your code should appear. One good example is "CustomKeypad" that is used to create a 4x4 keypad.

5. Now within the loop of your program you will need to assign one more character to the key pressed using the "getKey()" function introduced. To verify that all of your declarations and arrays have been created correctly, begin serial communication and print the values of the keys to the serial monitor with a short delay to prevent the monitor from scrolling too quickly. Connect the module 7 plug on the keypad to digital pins 52-30 with the wire strip hanging over the outside edge of the Arduino. Verify that each printed character matches the key pressed.

6. Once you have got the proper keys printing, you will need to print these to the LCD screen to properly display them. The character "c" for Cal(c)ulator will later be used to initialize a calculator function. Using the knowledge gained in the last lab, create a conditional statement that tests to see if the key received is "c". If this character is received, clear the LCD screen to get rid of any unwanted characters, print the word "Calculator" in the top left corner, and clear it again after 2 seconds. The character "N" is used to represent the "Clear/(N)o" button. Make this button simply clear the LCD.

7. Use conditional statements to print all numbers to the screen. Try using the Boolean operator "||" or to do this in a single statement. In separate statements you will need to print the mathematical operators "+, -, /, ^, =" and "X" used for multiplication, as these will be used for more complex functions in the future. Also, print the "." for using decimal numbers in a separate statement.

8. For the tragicomic functions "S" for sin, "C" for cosine and "T" for tangent, you will need to print "sin(, cos(, tan(". Rather than simply displaying the character.

9. Once you believe your code is correct, connect the module 6 plug in the second trainer to the GND-41 digital pins as you did in the previous lab. Also connect the keypad module 7 plug to digital pins 52-30 as you did in step 5. These two should overlap slightly. Connect the USB cable through the battery door of the second trainer and close the box. If everything has been performed correctly, you should be able to print any mathematical operation or number. The clear button should clear the screen, and pressing the Calc button should display the word Calculator.

Lab 11: Making a Calculator

Purpose

The purpose of this lab is to make a basic calculator. You will modify your last lab to perform calculations using arrays. This calculator will perform all the basic math functions including sin, tan, and cos. You will also learn the long data form which allows for much larger numbers to be used in calculations. This will require you to use many variable types for calculations and you will need to create several new conditional statements to successfully complete the lab. This calculator will have some limitations so you can program it in the allotted class time. Creating a fully functional calculator would take a lot more complicated code, and thus a lot more time.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Power Cable (Optional)

Arduino Trainer

Commands and Description

Command	**Description**
sin(*VARIABLE*)	The tragicomic function sin. The *VARIABLE* used for calculation must be written between parentheses. Results are expressed as radians.
cos(*VARIABLE*)	The tragicomic function cosine. The *VARIABLE* used for calculation must be written between parentheses. Results are expressed as radians.
tan(*VARIABLE*)	The tragicomic function tan. The *VARIABLE* used for calculation must be written between parentheses. Results are expressed as radians.
long *NAME*	Just like the "int" function long can be any integer value, named anything that is not already a command. Long numbers take up 4 bytes and can be between -2,147,483,648 and +2,147,483,647 allowing for calculation with much larger numbers.
goto *LABEL* *LABEL:*	You can use this command to specify a label by any name. When you use this command the program will search for the label and run any code after the colon.
++ and --	Adding to pluses to an integer adds 1 to that integer. Two minuses subtract 1. These are the increment and decrement commands.

Arrays:

In Lab 10 you created a character array to use the keypad with the "char" command. Number arrays can be created from any variable type (int, long, float) with the same syntax introduced for the “char” command. Simply replace “char” with “float” and use numbers to create an array of numbers with decimal places. Consider the following code:

```
float Number[5] = {0.1,0.2,0.3,0.4,0.5};

lcd.print(Number[2]);
```

The first float variable is assigned the address of 0 making “Number[0] = 0.1”. In the given code the number 0.3 would be printed to the LCD screen. You will use both number and character arrays to perform calculations in this lab. Now consider this code:

```
float Number[5] = {0.1,0.2,0.3,0.4,0.5};\

int i = 0;

lcd.print(Number[i]);

++i;

lcd.print(Number[i]);

--i;

lcd.print(Number[i]);
```

Using the variable “i” to specify the address in the array allows for the increment and decrement commands to be used. This will print “0.1 0.2 0.1”

Procedure

1. Begin by opening Lab 10 and creating a float array before the setup of your program. It is strongly recommended that you use the variable names mentioned in this procedure to make corrections easier, and allowing others to help if there is a problem. The float array should be named “Number”, and contain 5 variables all set to 0 initially. Create a character array named “math” with the same number of variables. Set these all to the character ‘0’. You will need two integers “i”, and “j”, used to increment and decrement to perform calculations. And finally you will need a long “Answer” variable and a float “fAnswer” variable. Set everything to 0.

2. It is a good idea to save this as Lab 11 at this point to avoid overwriting your previous lab 10 file. In the last lab pressing the clear button simply cleared the screen. In this lab the clear button should set all variables equal to 0, including both of the arrays. To do this Write " Number[0] = 0; Number[1] = 0;" … ect. This may be annoying but it is necessary to perform calculations properly.

3. You will need to set the key pressed to the float variable "Number[i]" to store this key and use it for calculation. Furthermore, pressing 10 should set "Number[i] = 10" rather than 1, then changing it to 0. A small bit of math can solve this problem. The key entered is a character in the ASCII encoded format. 0 is ASCII for 48 so you will need to subtract 48 from the key value entered. Use the equation "Number[i] = ((key-48) + (Number[i]*10))". This will multiply the first number entered by 10 to enter in larger numbers and subtract 48 from each new number to ensure it is stored in its decimal format.

4. Pressing +,-,X, or / should increment the "I" variable to signify the next key pressed will be a new number. You will also need to assign "math[j]" to the key pressed to store which mathematical function should be performed. Increment "j" after storing this value in the array. The sin, cosine, and tangent functions should simply store the character pressed and increment "j". These will only be used to solve for radian of integer numbers to verify that they work.

5. Pressing the enter key should first clear the screen and print the equals sign. This should also set "i", and "j" equal to 0 so the first number and math function entered can be checked. Set "Answer" and "fAnswer" equal to the first number in the float array. After all this is entered add the label "Again:".

6. For each math function you will need to add a conditional if statement that checks to see what character is stored in the "math[j]" array. Because you have already set Answer to the first number in the "Number[i]" array you will use this for the mathematical operations +,-, and X. For example if + is in the math array "Answer = Answer + Number[i+1]". This will add the current value to the next. Increment both the "i" and "j" variables after the math has been performed and go to the "Again:" label to check the next operation to be performed. Division and the tragicomic functions will use decimal numbers, so you will need to use the float "fAnswer" for calculation. At the end of each one of these conditional statements set the next

math character to the value '1'. Create a conditional statement for each of the basic math and tragicomic functions.

7. Finally you will need to print the answer to the screen of the lcd. By default, all characters in the math array are set to '0'. Create a conditional statement that checks for this, and prints the "Answer". If more complicated math has been performed math will equal '1'. Check for this and print the "fAnswer" variable for this case.

8. Now it is time to test if your code works. Connect the module 6 plug in the second trainer to the GND-41 digital pins. Also connect the keypad module 7 plug to digital pins 52-30 as you did in the previous lab. These two should overlap slightly. Connect the USB cable through the battery door of the second trainer and close the box. If your code works you should be able to add, subtract, and multiply up to 5 numbers. You should be able to divide any two integer numbers and display the answer with two decimal places of accuracy. You should also be able to use the sin, cosine, and tan function to find the radian value of any integer to two decimal places of accuracy. Try the equations below and compare your answers with others in the class or check against another calculator to verify the answer.

10,000-5,000+2,500-11+10 = _____. This checks to see if the Calculator can handle 5 numbers.

100X20X30X2X 5 = _____. This checks to see if the long Answer displays large numbers.

5/2 = _____. This checks to see if the calculator handles decimals well.

sin(1) = _____. This should display the answer in radians and checks the trig functions.

You may wish to improve your calculator in the future to use decimal numbers, and the other math functions. If you want more than two decimal places of accuracy try printing the values in scientific notation. Try to figure it out on your own or use some of the open source code available through the Arduino site. This thing can perform any calculation you code for.

Lab 12: Resistance and the Ohm Meter

Purpose

In this lab you will be creating an Ohm meter, which is a device used to measure resistance. You will write a sketch that measures the voltage on an analog input, and converts this to a measurement of resistance using the voltage divider equation. The equation introduced in Lab 8 can be rearranged using algebra to solve for the resistor values rather than voltage. You will use this device to measure the resistance of several resistors, and compare your measurements with the theoretical values of the resistors based on their color codes. You will also be introduced to the "while" loop. This is a type of command much like the "if" statement that only runs the portion of the code within the statement when the condition is satisfied. This will allow you to use your Ohm meter without the code from the calculator sketch interfering with you measurements.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Power Cable (Optional)

Arduino Trainer

Safety Leeds

Resistors

Commands and Description

Command	**Description**
while (*VARIABLE OPERATOR VARIABLE) {* *}*	This command is used exactly like an "if" statement. The only difference is that code included between the brackets will loop until the conditional statement is no longer true. No code outside of the brackets will be run.

Resistors:

Resistors are electrical devices used to vary voltage and current. These were used in previous labs with the photoresistor and the potentiometer, a type of variable resistor. There are several resistors of set values in the second trainer. Each one can be used with the voltage divider equation to measure a certain range of resistance. Instead of solving for voltage with the voltage divider equation you will use it to solve for resistance. The equation is:

$$V_{out} = V_{in}\left(\frac{R2}{(R1 + R2)}\right)$$

The R1 and R2 variables are the two resistors in the voltage divider. R1 is the first resistor connected to the voltage source. These are the resistors already connected to the module 8 plug. You will be solving for R2. These will be the external resistors you will be measuring. Rearranging the equation to solve for R2, you get:

$$R2 = \frac{R1}{({}^{V_{in}}/_{V_{out}} - 1)}$$

In this equation Vin is the 5 V input while Vout is the measured voltage. Resistors are labeled with 4 color coded bands. Each of the first three bands represents a number. The last band is usually gold or silver and is a measure of tolerance. The tolerance of a resistor determines how close it is to its theoretical value. If the resistor is labeled to be 500 ohms, it will be close to this number, but will not be exactly 500 ohms. This is why you will be comparing the measured and theoretical values. The colors each correspond with a particular number. These colors are listed below.

Number	**Color**	**Number**	**Color**
0	Black	5	Green
1	Brown	6	Blue
2	Red	7	Violet
3	Orange	8	Grey
4	Yellow	9	White

The first two colors are the bands used to specify the resistor value, while the third tells you how many zeros to place after the first two numbers. If the first three bands on a resistor where red, green, orange, then the theoretical value of the resistor would be 2 5 000 = 25,000. Similarly brown, black, brown would be 100.

Procedure

1. To begin open up Lab 11. You will continue building up this code until all devices on the second trainer can be run using a single sketch. Before the setup of the program create some new variables for your resistance calculations. You will need one integer "Resistor" to choose which resistor will get power for measurement. This should have a default value of 1. You will need one integer to activate the "OhmMeter" with a default value of 0. You will need an integer to read the analog pin used as an input to monitor voltage. Try naming this "Reading" and stet it to 0. You will also need a float to calculate "Voltage" also set to 0. Finally you will need a long variable to calculate "Resistance". Many of these were featured in Lab 6, and you may want to open this lab for reference.

2. In the character array for the keypad you will need to add the character 'O' for Ohm meter. This will take the previously unused space between the number '4' and the number '0'. This will be used to activate the Ohm meter. Below your code for the calculator create a conditional statement that tests to see if this new character has been pressed. If it has been pressed set

"OhmMeter" equal to 1. Next use the while command introduced to test if "OhmMeter is equal to 1. All of your code for this lab will go within this statement.

3. When the "while" loop is running, none of the code outside of the loop will be used by the Arduino, so you will need to copy and paste the code that sets the char key equal to the key pressed so the Arduino will still respond to buttons pressed on the keypad. Next create a conditional statement that tests if the Alpha key has been pressed. This is represented by the character 'A'. When this key has been pressed add 1 to the variable resistor. This will be used to select between different ranges of measurement. There will be 4 ranges, so you will need a separate conditional statement that sets "Resistor" equal to 1 when it is greater than 4. This will allow you to go back to the lowest range by pressing the Alpha key to cycle through.

4. Create another conditional statement that tests if the Clear/No button 'N' has been pressed. This should set "OhmMeter" equal to 0 to exit the while loop. These keys are the only keys that should be used in this section of code.

5. You will need to create four more conditional statements. One statement will be used for each range of measurement. The first will be when "Resistor" is equal to 1, and will be the lowest range. This lab will use the analog pins A0-A4. For the lowest range A0 will be set to OUTPUT with the "pinMode" command. All other pins will be set to INPUT. You can "digitalWrite (A0, HIGH)" to send voltage to this pin and power the first resistor. The variable "Reading" should read analog pin A4 which is used in every range to measure the voltage of the voltage divider created when measuring an external resistor.

6. Think back to Lab 6, and recall that analog measurements take the value of 0-1023. Convert the "Reading" measurement to a voltage the same way you did in Lab 6, and assign this to the "Voltage" variable. Next you will insert the equation to calculate "Resistance". In the equation given in the *Commands and Description* section, "Resistance" is R2. R1 is the resistor receiving power. Each resistor and their measured values are listed below. You will need to use these values in your calculations. The voltage listed is the actual measured voltage of each pin. While the signal is supposed to be 5V it varies slightly for each pin. The Voltage values listed will be used for the Vin value of the voltage divider equation.

Analog Pin	Resistor Value	Voltage Output When High
A0	99	4.75
A1	985	4.99
A2	9,860	4.95
A3	100,600	5

7. After calculating the value you will need to print this to the LCD along with the range you are in. To do this, clear the screen, and set the cursor to the top left corner and print the range. The first range is "Range 0-500". The second is 500-5000, the third 5000-50000, and the fourth 50000+. After printing the range, set the cursor to the bottom left hand side of the screen and print the "Resistance" value. Finally add a delay of 250 milliseconds to prevent the values from printing too fast, making them unreadable.

8. Now you should have code to calculate resistance for the first range. Do this for the other 3 ranges as well. Each range should "digitalWrite" the previous output "LOW" as to not power more than one pin at a time. For the first Range, you will need to write A3 low. This should be the very first thing written after the "if" statement. You will need to specify the "pinMode" for every range as well. Most of this can be copy and pasted excluding the resistance calculation and the range.
9. Once you are done with your code, connect the module 6 and 7 plugs the same way as you did in the last two labs. Connect the module 8 plugs to analog pins A0-A7. Connect the Module 9 plug from GND to digital pin 8. All plugs should hang over the outside edge of the Arduino. Close the trainer and connect the USB through the battery door. Connect safety leads to the Red and Black sockets of the trainer, and get some resistors.

10. To test if the code works, pressing the Ohm button should activate the Ohm meter. Clear should exit. When you press Alpha, the range should cycle through. Read the value of 4 different resistors color codes. Next select the range on your Ohm meter that best fits the theoretical value. The value displayed should be close to the predicted one, but it is very rarely the same. Compare your measured values with others to test for accuracy. When no measurement is taken, a random number will be displayed. This is normal and should be ignored.

Lab 13: Creating a Light Meter

Purpose

In this assignment you will be creating a light meter that measures the unit of Lux. This will use the photo resistor in the same configuration as Lab 6. The code for this will be added on to the previous code produced for the second trainer. You will also be learning about the "for" loop to average measurements over time. In the last lab you may have noticed the number displayed on the LCD would change somewhat rapidly making it difficult to read. Averaging the recorded values over time can produce more stable readouts. You may wish to go back after completing this lab to create more stable measurements for the Ohm meter. For loops can also be used to increment arrays and set all the values equal to a particular number rather than specifying every number in the series.

100300

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Power Cable (Optional)

Arduino Trainer

Light Source

Commands and Description

Command	**Description**

for (*INITIALIZATION; CONDITION; INCREMENT) {*
}
	INITIALIZATION occurs once when the code is run. For example "i = 0" will set that value once the for loop is encountered. The condition is tested every time the loop runs. As long as it is true the loop will run again. For example "i < 6" will test the variable "i". Once this is no longer true the loop terminates. The variable tested is incremented with the statement "i++" to raise it by one every time the loop is tested. This way the condition will eventually be broken. The decrement command, "i--" , may also be used. Any code between the brackets will be run each time through the loop.

Calculating Averages with for

This is useful code for dealing with arrays. This lab will use previous variables to calculate light, and some new ones. First the code from the previous lab was used to determine the resistance of the of the ohm meter. The photoresistor is R1 in this case, however, so the equation below was used for the resistance calculation:

$$R1 = \frac{R2 \times V_{in}}{V_{out}} - R2$$

This resistance value will be store in a new array "LightResistance[i]". A conditional statement checks to see if 5 values have been stored then executes the "for" loop:

```
if (i == 5) {
    for (i=0; i<6; i++) {
      Average = (Average + LightResistance[i]);
    }
i = 0;
Average = Average/5;
}
```

The “Average” variable is used for calculating the average of the resistance. This loop starts by setting “i = 0”, it loops 5 times incrementing “i” each time, and finally when the loop is done it sets “i” back to 0 for new values to be stored. The average is then divided by 5 to find the resistance. This will create a more stable and accurate reading.

Procedure

1. Open up the sketch you have been creating for the second trainer and create an integer “LightMeter” that will be used to activate the light meter the same way as the previous lab. Create two long integers “Average” and “Lux” for calculations. Create a long array with 5 variables called “LightResisistance”. Set all of these integers equal to zero. Finally add one more integer for “Calculator”. Now that you have three separate sections of code that can run, you will need to create a while loop that runs the code for the calculator when the ‘c’ button is pressed.

2. In the character array for the keypad add the character ‘L’ for Light after the number ‘1’ and before ‘A’. Below the code you have already made, create another conditional statement that will activate the light meter by receiving the ‘L’ character. Create another while loop for this section of your sketch. This while loop will need the same code to get the keys pressed on the keypad, and should be terminated by pressing the clear button. Pressing any of the other buttons for calculator or ohm meter should also select these modes in all of the while loops.

3. Analog pin A5 will be the input for reading the voltage on the photoresistor and A6 should be HIGH to power the device. Use the same code, and variables used to calculate voltage in Lab 12, to determine the voltage on A6.

4. To calculate the resistance of the photoresistor, use the equation in the *Commands and Description* section. Resistor R2 is 100,300 ohms. Set “LightResistance[i]” to this value, and increment i each time this calculation is performed. Using the “for” command introduced find the average of 5 resistance readings, and assign this value to the integer “Average”. 5. In order to

covert the resistance measurement of the photoresistor to "Lux" the resistance was calculated and plotted on a graph with resistance as the x axis and "Lux" (measured with a meter of known accuracy) was the y axis. The photo resistor on the second trainer was tested and its resistance values were plotted against lux measurements of known accuracy. To perform the conversion, constrain "Average" to 800 as the min, and 35000 as the max. Also constrain "Lux" from 0-800. Use the "map" command to map "Average" to these values. Remember that the photoresistor has more resistance in the dark, when there will be a lower "Lux" reading, so you will need to reverse the 800-0 measurement. Review Lab 8 if you have forgotten how to do this.

5. Next you will need to print the Lux measurement in the bottom left hand corner of the LCD and display "LUX" in the top right. You should be familiar enough with the LCD screen commands to do this. If you have trouble look at how you printed the resistance in the last lab. Also add a 50 millisecond delay at the end of the code to prevent these values from printing too fast.

6. Once you are done with your code, connect the module 6 and 7 plugs for the keypad and LCD. Connect the module 8 plugs to analog pins A0-A7 and the Module 9 plug from GND to digital pin 8 just like in the last lab. All plugs should hang over the outside edge of the Arduino. Close the trainer and connect the USB through the battery door.

7. If your code is correct pressing the Light button should begin light readings. You should expect between 200 and 800 for an average lighting source. The value should increase when exposed to more light, and decrease when turned away from the light, or covered slightly. The value should not exceed the numbers specified in the constrain command. Compare your measurements with others in the class to verify that your calculations are correct.

Lab 14: Ultrasonic Distance Measurements

Purpose

In this lab you will be using your programming knowledge to work in a group and interface with a new sensor. The distance sensor is located on top of the second trainer. This is an ultrasonic range finder, and uses sound above the human level of hearing to measure distance. You will work together to create code that will produce distance measurements in both inches and centimeters. You will need to use a ruler to determine how to make this conversion and compare your measurements to verify accuracy. To keep it simple you will only be introduced to two new commands very similar to ones you have already used. These commands enable you to use microseconds for calculations.

Equipment

Arduino Mega

Arduino USB Cable

Computer

Arduino Power Cable (Optional)

Arduino Trainer

Measurement device

Commands and Description

Command	**Description**
delayMicroseconds()	This is the same as the "delay()" command, except the number specifies the time in microseconds rather than milliseconds.
pulseIn(*PIN, STATE*)	This command measures the amount of time in microseconds it takes a pin to go from one state to another. The *PIN* can be any digital PWM pin, and the *STATE* is either high or low.

The Ultrasonic Sensor

This sensor can measure any distance from 3 centimeters to 4 meters with accuracy. Like most devices it has a power and ground pin, and a signal wire. When a short 5V pulse is sent to the signal wire the device sends out a sound wave that humans cannot hear. The device then listens for the sound wave to return and sends a high pulse on the same signal wire when the pulse is received. The longer away an object is, the longer time between the send and the receive pulse. Because sound travels very fast, this all happens in a matter of microseconds.

To control the device you will need to use the "pulseIn" command to measure the amount of time it takes for the device to go high. Using the command "pulseIn (12, HIGH)" will start timing after the signal pulse is sent. When the device receives the sound it will go high again. The time between these pulses will be returned in microseconds.

Procedure

1. Most of this will be up to you and your group and this section will only pride coding help on how to use the distance sensor. You will need to insert the character 'D' between '7' and '=' in the keypad character array. This will be used to activate the distance measuring mode of operation by pressing the "Dist" button.

2. To use the ultrasonic sensor you will need to send pulses through digital pin 12. First configure it as an OUTPUT and write the pin LOW to ensure it goes to 0. Next insert a very short delay of only 2 microseconds. Write the pin HIGH to start sending the sound wave pulse. The pulse should only be high for 15 microseconds. After writing the pin LOW again, insert a delay of 20 milliseconds and configure pin 12 as an INPUT to receive the pulse. You will need to create a variable to record the time it takes for the pulse to return. Set this variable equal to the "pulseIn(12, HIGH)" time received.

3. You will need to read these numbers on the LCD screen for this assignment. Your code should allow you to press the specified button to initiate this mode of operation. You will need to connect all of the plugs for the second trainer the same way as you did in the last lab. Once you have code that will print the received values to the screen upload your code and make all of the necessary connections.

4. To interpret this measurement, place the second trainer on a flat surface. Set a measurement device of known accuracy down to visually observe the distance of an object from the sensor. Place some object a few inches away from the sensor and record the number observed; also record the distance of the object from the sensor. Do this for at least 10 different measurements, in equal steps, recording each one.

5. Once you have your data plot the time received on the x axis and the distance on the y axis of a graph. Determine an equation that will allow you to convert your time measurements to inches, and centimeters.

6. Once you have the equation go back to your program and display the distance received in both inches and centimeters in an easy to read way.

8. Reload your modified program and make all necessary connections. Use the measurement device previously used to verify that your calculations are correct. If they are, congratulations! You have now mastered the basics of the Arduino!

Appendix C: Lab Descriptions

Lab 1: “Hi Guys!”

Lab 1 introduces the Serial function of the Arduino as well as the basic program format. Serial commands allow the Arduino to communicate with a computer while a program is running. This is a valuable tool in monitoring and debugging sketches. It allows users to use the “print” command to send data to the Arduino serial monitor on the computer to display variable values, input/output states, and any other information in real time. The setup command is used to setup initializers in a program that are meant to run once, while the loop command loops continuously once the program is started. Because much of the Arduino programming is meant to continuously monitor inputs/outputs and variables, the majority of programming statements are used within the loop. Initializers such as “Serial.begin” are used in the setup to start communication before the rest of the program is run. The purpose of this lab is to print the phrase “Hi guys!”, or any other phrase chosen by students, to the serial monitor on the computer. These are Arduino library specific commands, and not generic C commands. The “print” command works similarly as the library specific command “printf” in the stdio.h library file for the C language. “printf, print formatted, is perhaps the most commonly used output function (Berry & Meekings, 1984, p.11). The “print” command must also be preceded by “Serial.” to specify that the Arduino is printing to the serial monitor.

Lab 2: Simulating a Stop Light

In lab 2 other Arduino library specific commands are introduced to show the basics of using the Arduino I/O functions. The “pinMode” command specifies weather a specific Arduino pin is an input or an output. The “digitalWrite” command is used to write a specific digital pin either HIGH (1, 5V) or LOW (0, 0V). These two commands are used for interfacing with all digital peripheral devices. The “dealy” command which pauses the program for a given time in milliseconds is also introduced. This is included in the Arduino library, and is usually formed in C through the use of a “for” loop and the time.h library file. These commands are used along with the previously introduced commands to simulate a stoplight on the trainer that uses red, yellow, and green LEDs.

Lab 3: Inputs and Outputs

In this lab students learn the command “int” used to define a data type as an integer is introduced. These can be named anything so long as they start with a letter, and do not take the name of an existing command. Kernighan B.W. and Ritchie (1978, p. 9) state “an int is a 16-bit signed number, that is, one that lies between -32,768 and +32,768.” These are useful for both calculations, and to simplify code into a more readable structure. Rather than specifying a pin number every time the “int” function allows students to name pins more understandable expressions such as “GreenLED”. The “const” command is used with data types to keep them constant. This is also introduced and recommended for pin numbers as these should not change throughout the sketch.

This lab also introduces the “if…else” control expression to perform logical operations based on inputs or variables. The “if” statement is one that tests a conditional expression, and executes the action specified if that expression is true. The “if…else” expression performs the same task, however a second action is specified after the command “else” if the condition is not satisfied, the program executes the action listed under the “else” command. (Johnsonbaugh & Kalin, 1997, p. 49). This is the primary statement used in the labs to enact various functions. In order to properly form conditional expressions the syntax for comparison operators are also introduced, such as greater than, less than, and equal to. The Boolean operators for and, or, not are also introduced to form conditional expressions where multiple conditions should control the output. Finally the “digitalRead” command is introduced that reads the state of a digital input in the same way “digitalWrite” controls an output. This allows students to base conditions on inputs.

Lab 4: Better Serial Communication

Lab 4 teaches students about better serial communication between the Arduino and the computer to introduce various data types and the concept of binary. The “println” command is introduced that functions the same as the “print” command except it enters each new text on the line below the previous one, rather than scrolling to the right. This is useful in organizing information in the serial monitor. When using these commands any character is not printed in its ASCII encoded form.

In this lab students use the serial monitor to neatly display the numbers 0-9 in their decimal, binary, hexadecimal and octal values. The “Serial.write” command is also introduced that automatically prints characters in their ASCII encoded form. Students are taught how to convert between decimal, a base 10 counting system, and binary, a base 2 counting system. An Arduino specific command “Serial.avalable” is used to retrieve data entered into the serial monitor and prompt an output based on this data. While the conversion is not shown for octal, a base 8 system, and hexadecimal, base 16, students use the Arduino to discover what these are.

Lab 5: Programming a Keypad

This assignment does not introduce any new C commands but focuses on using those already taught in more complicated ways. The goal of this lab is to have students use the information discovered in the previous lab to prompt a keypad to display the number pressed in binary using 4 LEDs. This lab uses multiple inputs and outputs, and total of 12 “if” statements to produce the binary readout. Completion of this lab gives students the ability to interface with the keypad on the first trainer for future labs, thus saving time by editing this sketch to produce different outcomes. The keypad uses a 4 input binary circuit to display the numbers and allows for control using multiple buttons.

Lab 6: Reading Analog Inputs

Next students learn the “analogRead” function and a new data type known as a “float”. A “float” is a command used to specify a variable type where “float stands for floating point, i.e., numbers which may have a fractional part” (Kernighan & Ritchie, 1978, p. 9). The “analogRead” function reads the voltage (from 0-5V) on an analog input pin. The value is converted from a 10-bit binary value to a decimal number from 0-1023, and sent to the serial monitor for interpretation. Students convert this number to the actual voltage read by the analog pin that is connected to the wiper of a potentiometer used to vary the voltage. This provides the basis for interfacing with analog electrical devices, and introduces the concept of an analog to digital (A/D) converter. The concept of resolution is introduced to explain the A/D converter. Students also learn how to only print relevant data from inputs and outputs to the serial monitor. Previously data would print only when prompted thorough serial communication or otherwise print continuously.

Lab 7: Dimming a Light Using the Keypad

The Arduino also has an “analogWrite” function that uses pulse width modulation (PWM) to produce a varying DC value. Students control the lighting level of two ultra-bright LEDs in equal steps using the keypad introduced in previous labs. The “analogWrite” function is used for controlling analog electronic devices, and is useful in teaching the concept of PWM. The commands and description section explains that PWM is the act of pulsing a signal on and off very quickly to form an analog voltage when measured over time. The ultra-bright LEDs appear to dim by using this method. The PWM occurs as a 490 hertz (Hz) square wave. The width of the positive pulse of the 490 Hz square wave determines the amount of time the LEDs are on and thus the brightness of the LEDs. For example, if the positive pulse width is only active for 75% of the period the LEDs appear to be dimed by 25%.

Lab 8: Maintaining Lighting Levels Using a Photo Resistor

Students use the knowledge they gained from previous labs to automatically adjust the brightness of a light based on the lighting levels in a room in this assignment. A photo sensitive resistor that varies its resistive value based on the presence of light is used to control the ultra-bright LEDs. While these are not actually bright enough to illuminate the whole room, the concept taught is valuable in coordinating analog inputs and outputs and has real world application in designing an automatic dimmer circuit. Students also learn the Arduino library specific “map” command and the “constrain” command. The map command performs basic arithmetic to map one set of values to another. This can be used to reverse a set of input values, i.e. 0-100 to 100-0, or to multiply or divide them, i.e. 0-100 to 0-1000. In the second case the value of 90 would be mapped to 900. Constraining values causes them to stay within specific boundaries. In other words an integer, “int” cannot be less than or greater than the specified values. These commands allow students to better use variables.

Lab 9: Interfacing With an LCD Screen

This is the first lab where a header file is introduced using the “#include” command. As previously mentioned, the Arduino automatically loads several library files to allow for the use of Arduino specific commands that execute more complicated C code. In this lab students are shown how to manually use header files and their uses through the “LiquidCrystal” library. This library allows for interface with an LCD screen on the second trainer to produce readout on a portable device. Students are made familiar with many library specific commands used to control the LCD screen. Some of the notable LCD commands are “setCursor”, “home”, and “clear”. The commands are commonly set up as “#define” statements. “#define” statement replaces any character string with the specified code before the program is compiled into machine code. The commands introduced are commonly used as “#define” statements to set cursor positions without having to specify their specific coordinates.

Lab 10: Interfacing With a Matrix Keypad

In Lab 10 student are introduced to several new data types and C commands through the use of a matrix keypad library. Students will use the keypad on the second trainer and interface it with the LCD screen to produce a readable output on the portable trainer. This also provides the basics for the next lab in which students design a calculator. The “byte” data type is introduced that signifies that the data consists of 8 binary bits. The “char” command is also used to create a character array. The “char” command is short for character and “creates a 1-byte cell that can hold one character” (Johnsonbaugh & Kalin, 1997, p. 70). A character array is a series of characters that each occupies 1-byte arranged in columns and row. Students use the array to specify which button is pushed.

The command “getchar” is also used to retrieve a character from the character array. “The command “getchar” reads one character from the standard input” (Johnsonbaugh & Kalin, 1997, p. 70). In this case the input is the matrix keypad, and the term “Key” is used to denote that the character comes from this input. As a result this command takes the form “getKey” in the sketch. A matrix keypad is a type of keypad that closes the circuit between various rows and columns to signify that a key has been pressed. The Arduino checks to see which row is written HIGH and which column is read as HIGH and determines the key. This in turn will enable students to use the keypad to enact certain actions based on the character, or simply print the character.

Lab 11: Making a Calculator

This assignment expands upon the previous one, and introduces the data type “long” as well as the “++” increment and “--" decrement functions. A “long” is an integer number that occupies 32 bits and thus allows for calculations with much larger numbers. The increment and decrement commands are shorthand that adds or subtracts the value of 1 from an integer. The “array” function is also introduced for storing variables that can be called by the name of the array and the numerical pointer address for the specific number needed. This allows students to perform more complicated calculations with multiple numbers without having to specify a variable name for each one (Berry & Meekings, 1984, p.60-64). This will also allow students to store subsequent numbers entered into the array. Students will use all arithmetic operations (+,-,*,/) and the tragicomic operators “sin”, “cos”, and “tan” to perform mathematical functions on the Arduino.

Lab 12: Resistance and the Ohm Meter

Students have been programming primarily in the “loop” part of the sketch. Because this loops continuously all conditions up to this point could be satisfied with the “if” statement. This lab introduces the “while” loop and its uses. The “while” loop is accompanied by a conditional expression much like the “if” statement. If the condition is true the action in the loop is executed and the conditional expression is tested again. As long as the expression remains true, only the portion of the program contained within the while loop is run. This will allow students to add the Ohm Meter program they will create into the same sketch as the calculator while ignoring the commands used to operate the calculator (Johnsonbaugh & Kalin, 1997, p. 40).

This lab also introduces students to the concept of electrical resistance, and the Ohm. Students will first write a sketch that allows the Arduino to calculate resistance using the leads on the second trainer, through a voltage measurement on an analog pin. They will measure several resistors and compare these to the resistor values expressed by their color codes.

Lab 13: Creating a Light Meter

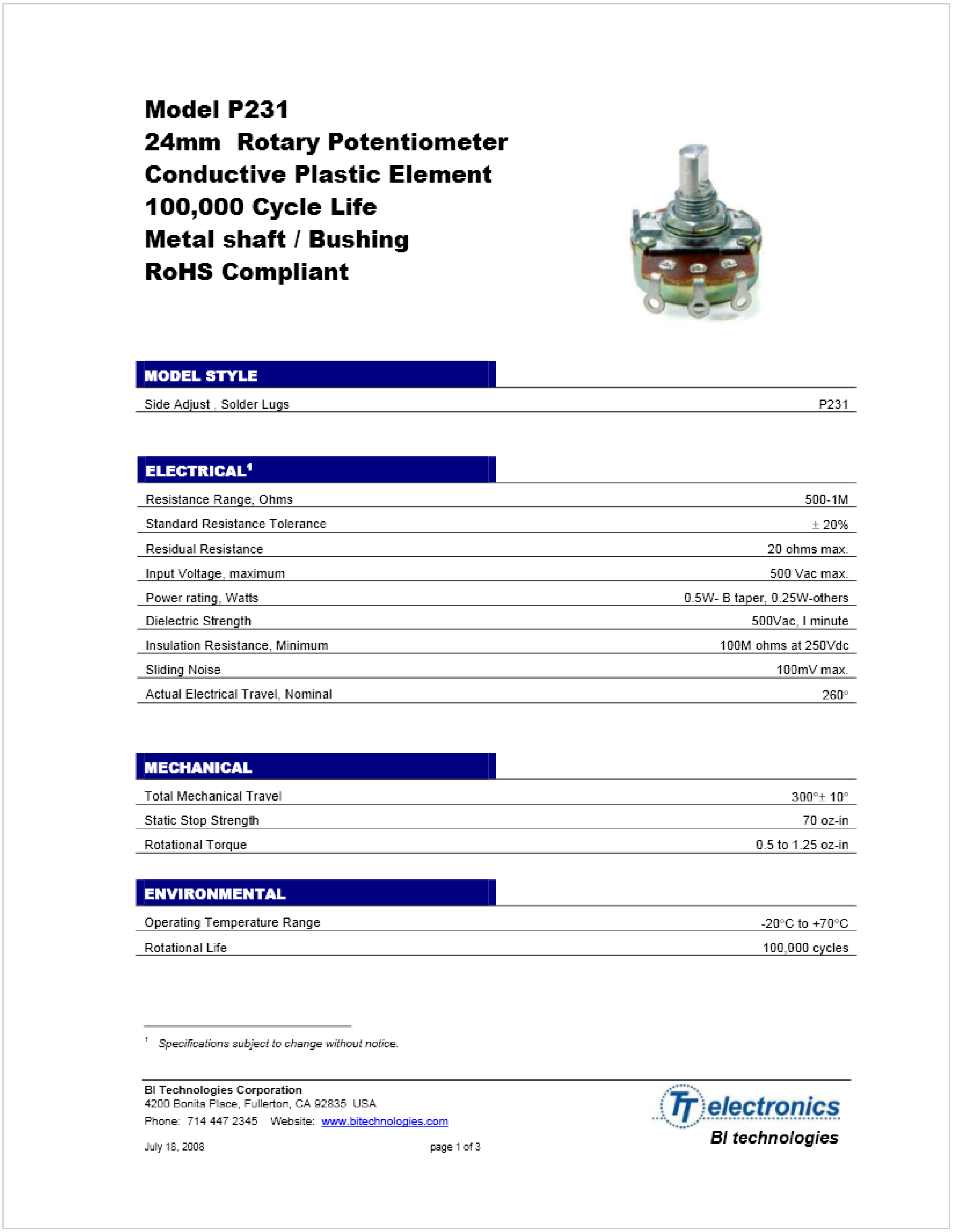
In this lab students will average light readings from the same photo resistor used to control lighting levels in Lab 8. They will develop the equation to convert the analog input value measured to the SI unit of Lux. The "for" loop will also be used to average the values over a short period of time to ensure accurate measurements, and to correct for small fluctuations. The "for" command is another looping command. It consists of 3 expressions where the first expression is tested, and initiates the loop if it is true. The second expression tell the loop when to terminate, and the third expression usually increments or decrements a variable used in the first two expressions. "The *for* statement proves convenient to use when it is necessary to execute a loop a given number of times" (Berry & Meekings, 1984, p.49). This statement can be used to average the measurements over a period of time. This program will also be compiled into the calculator and Ohm meter sketch to make the second portable trainer a multi-functional device.

Lab 14: Ultrasonic Distance Measurements

This is the final lab on the second trainer, and will allow the trainer to perform four different functions at the press of a button. Students will interface with an ultrasonic distance sensor that measures the distance of an object located in front of it using a sound wave above the human level of hearing. Students will use the "pulseIn" Arduino specific command that measures the amount of time it takes for an input to return to a HIGH state. The "delayMicroseconds" command works the same way as the "delay" command for shorter time intervals. This command will be used to rapidly switch the device from input to output in order to use the measured time to determine the distance of the object from the sensor. Students will work in a group to plot the digital measurements taken, and produce an equation that allows for the distance to be displayed in inches, and converted to centimeters so both units of measurement can be known. The sketch will be checked with a ruler of known accuracy to ensure the distance is measured appropriately and that the conversion is correct.

Appendix D: Datasheets

10 K Rotary Potentiometer



Model P231
24mm Rotary Potentiometer
Conductive Plastic Element
100,000 Cycle Life
Metal shaft / Bushing
RoHS Compliant

MODEL STYLE	
Side Adjust , Solder Lugs	P231

ELECTRICAL[1]	
Resistance Range, Ohms	500-1M
Standard Resistance Tolerance	± 20%
Residual Resistance	20 ohms max.
Input Voltage, maximum	500 Vac max.
Power rating, Watts	0.5W- B taper, 0.25W-others
Dielectric Strength	500Vac, I minute
Insulation Resistance, Minimum	100M ohms at 250Vdc
Sliding Noise	100mV max.
Actual Electrical Travel, Nominal	260°

MECHANICAL	
Total Mechanical Travel	300°± 10°
Static Stop Strength	70 oz-in
Rotational Torque	0.5 to 1.25 oz-in

ENVIRONMENTAL	
Operating Temperature Range	-20°C to +70°C
Rotational Life	100,000 cycles

[1] *Specifications subject to change without notice.*

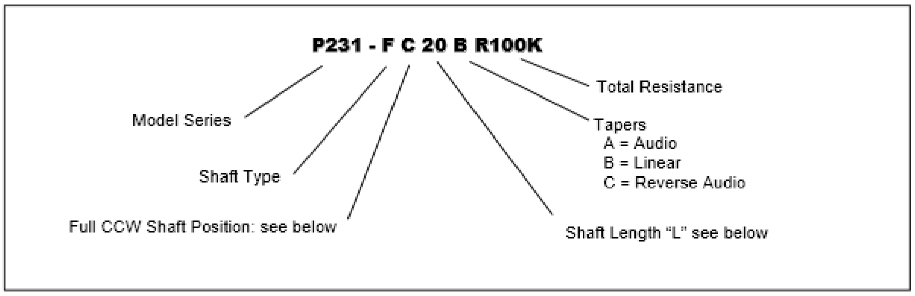
BI Technologies Corporation
4200 Bonita Place, Fullerton, CA 92835 USA
Phone: 714 447 2345 Website: www.bitechnologies.com

TT electronics
BI technologies

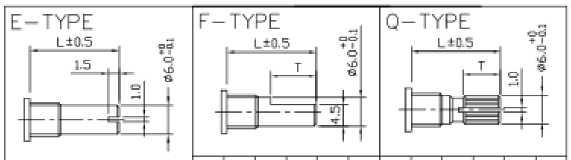
July 18, 2008 page 1 of 3

Model P231

ORDERING INFORMATION[2]



Shaft Types



E-TYPE

L	15	20	25	30	35

F-TYPE

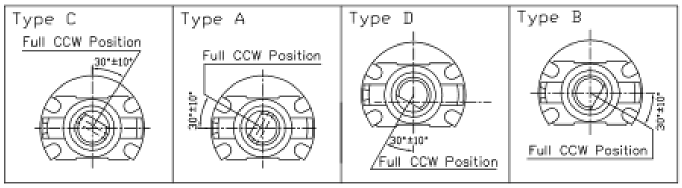
L	15	20	25	30	35
T	7	12	12	12	12

Q-TYPE

L	15	20	25	30	35
T	6	10	12	12	12

Shaft Position (F-Type Shaft)

Dashed lines on Type "C" and Type "A" shows position of adjustment slot for E-Type and Q-Type shafts



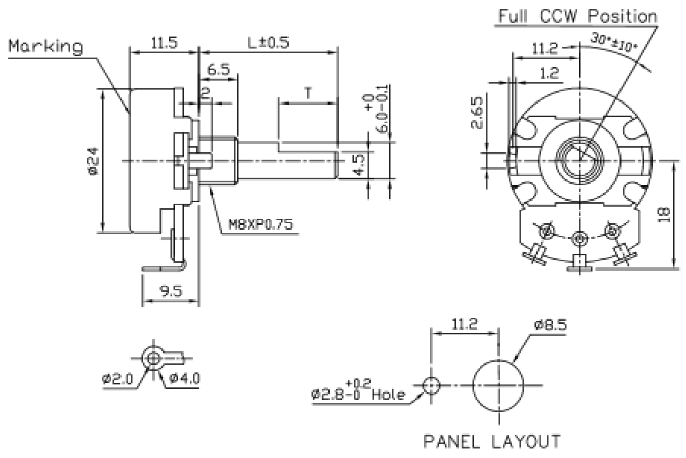
STANDARD RESISTANCE VALUES, OHMS

500	1K	2K	5K	10K	20K	50K	100K	200K	500K	1MEG

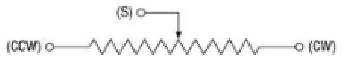
[2] *Contact our customer service for custom designs and features.*

Model P231

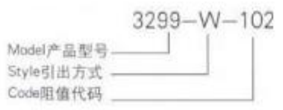
OUTLINE DRAWING

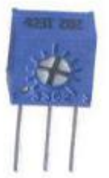


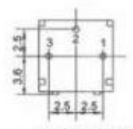
CIRCUIT DIAGRAM



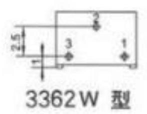
10K Tuner Potentiometer




3362P


3362P 型

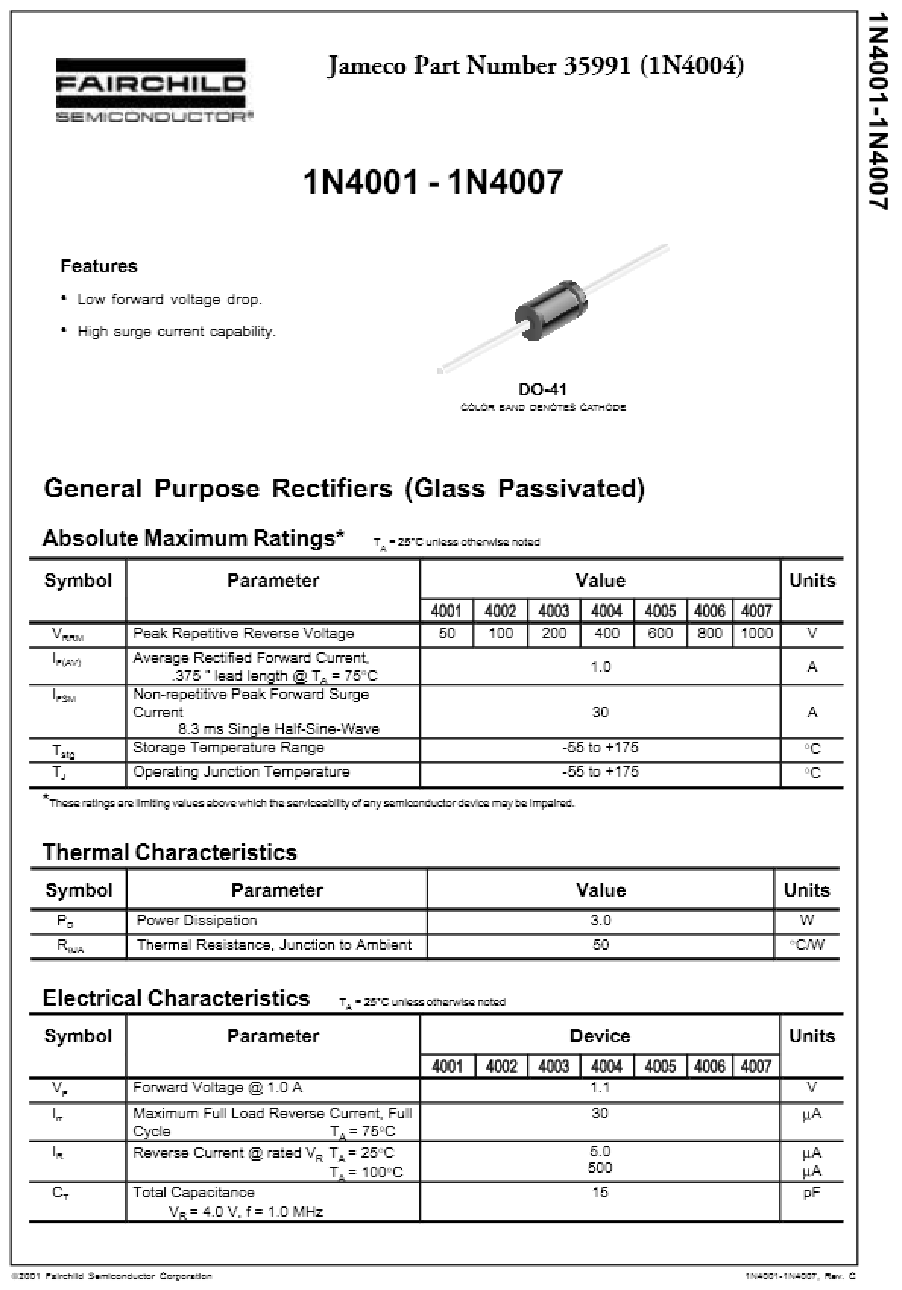
3362W


3362W 型

Series	P/N	MFG	Resistance (Ohms)
P	253982	3362P-1-102	1,000
	254036	3362P-1-104	100,000
	254028	3362P-1-503	50,000
	254044	3362P-1-504	500,000
W	254175	3362W-1-102	1,000
	254204	3362W-1-103	10,000
	254247	3362W-1-104	100,000

Characteristics			
Total Mechanical Travel	250°±10°	**Starting Torque**	≤20mN.m
Withstand Voltage	101.3KPa 500V 8.5KPa 315V	**Insulation Resistance**	R1≥1GΩ
Effective Electrical Travel	210°±10°	**Standard Packaging**	50pcs/tube
Rated Power	+70℃ 0.5W +125℃ 0W	**Mechanical Endurance**	200 cycles △R≤±10%R

Diode



Jameco Part Number 35991 (1N4004)

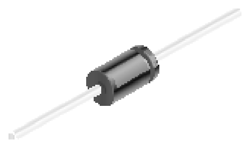
FAIRCHILD SEMICONDUCTOR®

1N4001-1N4007

1N4001 - 1N4007

Features

- Low forward voltage drop.
- High surge current capability.



DO-41
COLOR BAND DENOTES CATHODE

General Purpose Rectifiers (Glass Passivated)

Absolute Maximum Ratings*

T_A = 25°C unless otherwise noted

Symbol	Parameter	Value							Units
		4001	4002	4003	4004	4005	4006	4007	
V_{RRM}	Peak Repetitive Reverse Voltage	50	100	200	400	600	800	1000	V
$I_{F(AV)}$	Average Rectified Forward Current, .375 " lead length @ T_A = 75°C	1.0							A
I_{FSM}	Non-repetitive Peak Forward Surge Current 8.3 ms Single Half-Sine-Wave	30							A
T_{stg}	Storage Temperature Range	-55 to +175							°C
T_J	Operating Junction Temperature	-55 to +175							°C

*These ratings are limiting values above which the serviceability of any semiconductor device may be impaired.

Thermal Characteristics

Symbol	Parameter	Value	Units
P_D	Power Dissipation	3.0	W
$R_{\theta JA}$	Thermal Resistance, Junction to Ambient	50	°C/W

Electrical Characteristics

T_A = 25°C unless otherwise noted

Symbol	Parameter	Device							Units
		4001	4002	4003	4004	4005	4006	4007	
V_F	Forward Voltage @ 1.0 A	1.1							V
I_{rr}	Maximum Full Load Reverse Current, Full Cycle T_A = 75°C	30							µA
I_R	Reverse Current @ rated V_R T_A = 25°C / T_A = 100°C	5.0 / 500							µA / µA
C_T	Total Capacitance V_R = 4.0 V, f = 1.0 MHz	15							pF

©2001 Fairchild Semiconductor Corporation

1N4001-1N4007, Rev. C

Typical Characteristics

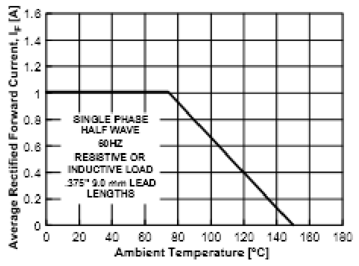


Figure 1. Forward Current Derating Curve

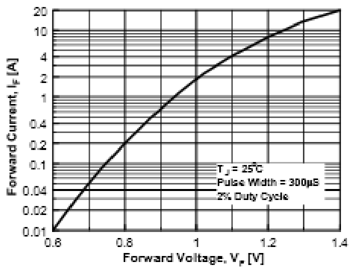


Figure 2. Forward Voltage Characteristics

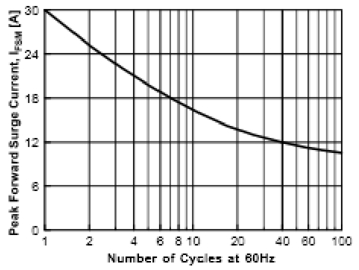


Figure 3. Non-Repetitive Surge Current

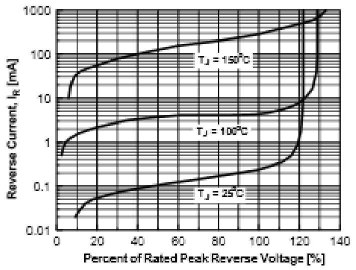
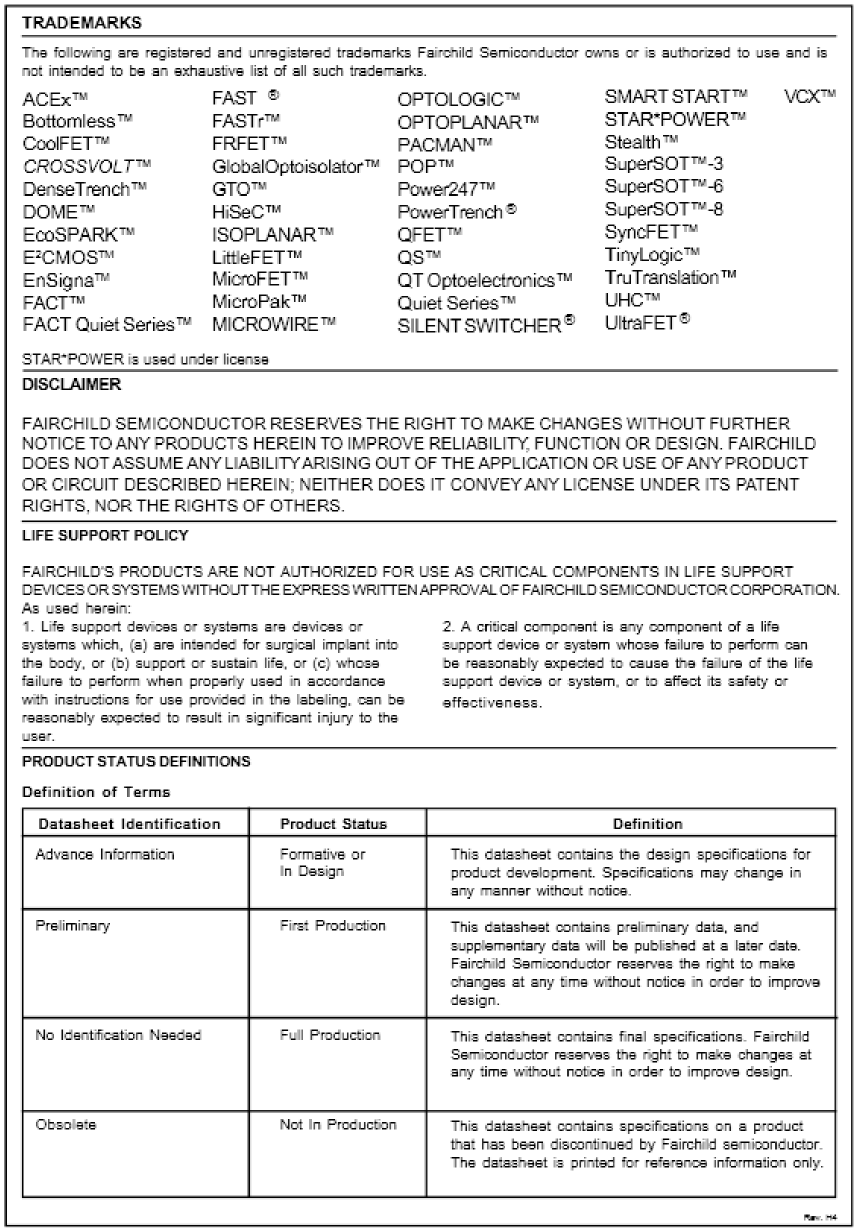


Figure 4. Reverse Current vs Reverse Voltage



Green LED

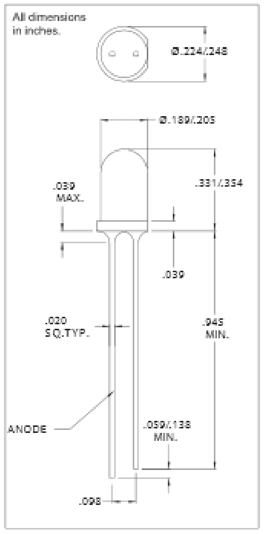
Chicago Miniature Lighting, LLC

Lighting the world since 1910

4304H Series Solid State LED Lamps-
Super Brite 30 mcd T-1 3/4 (5mm)

4304H Series Solid State LED Lamps-Super Brite 30 mcd T-1 3/4 (5mm)

DESCRIPTION AND FEATURES



T-1 3/4 Super Brite LEDs
- High-intensity light output.
- Wide viewing angle.

ELECTRO-OPTICAL CHARACTERISTICS AND RATINGS

PART NUMBER	4304H1	4304H3	4304H5	4304H7
Output Color	Red	Amber	Green	Yellow
Diffusion	Diffused	Diffused	Diffused	Diffused
Package Color	Red	Amber	Green	Yellow
Test Current (mA)	10	10	10	10
Forward Voltage Typ. (V)	2.0	2.2	2.1	2.1
Forward Voltage Max. (V)	3.0	3.0	3.0	3.0
Luminous Intensity Min. (mcd)	3.0	4.0	3.0	2.5
Luminous Intensity Typ. (mcd)	6.3	20	10	6.3
Capacitance @ V=0 (pF)	45	4	20	45
Peak Wavelength (nm)	650	608	563	585
Viewing Angle 2θ 1/2 (degrees)	65	60	75	65
Power Dissipation (mW)	60	135	75	60
Reverse Breakdown Voltage Min. (V)	5	5	5	5

147 Central Avenue
Hackensack, NJ 07601
Phone: (201) 489-8989
Fax: (201) 489-6911

18703 Dix Toledo Road
Brownstown, MI 48193

Saret Zone Franca, Edifico B
Alajuela Costa Rica

E-mail:sales@chml.com • Web Site: www.chml.com

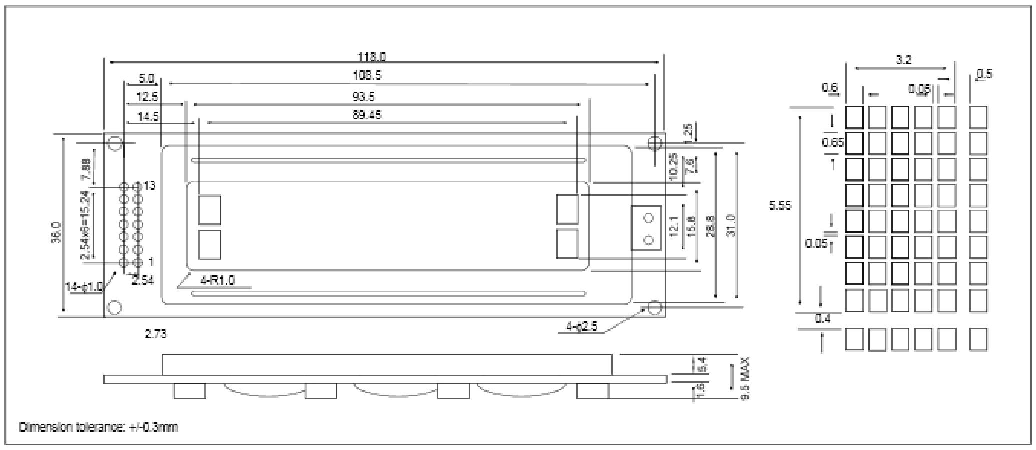
Chicago Miniature Lighting, LLC reserves the right to make specification revisions that enhance the design and/or performance of the product

LCD Screen

HDM24216H-2

Dimensional Drawing

24 Characters x 2 Lines



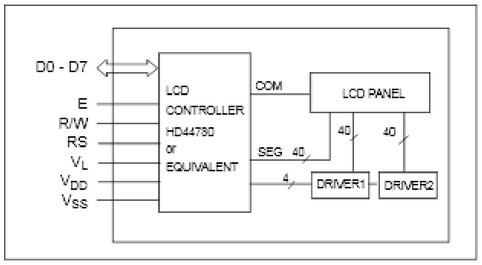
Features

Character Format ....5x7 Dots with Cursor
Backlight...EL Optional
Options....TN/Gray STN/Yellow STN, 12 o'Clock/6 o'Clock View
Normal/Extended Temperature
Normal/Negative Displays

Physical Data

Module Size....................................118.0W x 36.0H x 9.5T mm
Viewing Area Size.......................................93.5W x 15.8H mm
Weight...42g

Block Diagram



Absolute Maximum Ratings

PARAMETER	SYMBOL	MIN	MAX	UNIT
SUPPLY VOLTAGE	V_{DD}-V_{SS}	0	7.0	V
SUPPLY VOLTAGE FOR LCD	V_{DD}-V_L	0	13.5	V
INPUT VOLTAGE	V_{IN}	V_{SS}	V_{DD}	V
OPERATING TEMPERATURE	T_{OP}	0	50	°C
STORAGE TEMPERATURE	T_{STG}	-20	70	°C

Electrical Characteristics (VDD=5.0±0.25V 25°C)

PARAMETER	SYM	CONDITION	MIN	TYP	MAX	UNIT
INPUT HIGH VOLTAGE	V_{IH}	-	2.2	-	-	V
INPUT LOW VOLTAGE	V_{IL}	-	-	-	.6	V
OUTPUT HIGH VOLTAGE	V_{OH}	I_{OH}=0.2mA	2.4	-	-	V
OUTPUT LOW VOLTAGE	V_{OL}	I_{OL}=1.2mA	-	-	0.4	V
POWER SUPPLY CURRENT	I_{DD}	V_{DD}=5.0V	-	1.0	2.2	mA
POWER SUPPLY FOR LCD	V_{DD}-V_L	TA=25°C	4.3	-	4.7	V
DRIVE METHOD	1/16 Duty					

Pin Connections

PIN NO.	SYMBOL	LEVEL	FUNCTION	
1	V_{SS}	-	0V	Power supply
2	V_{DD}	-	5V	
3	V_L	-	-	
4	RS	H/L	H: Data Input L: Instruction data Input	
5	R/W	H/L	H: Data read L: Data write	
6	E	H,H→L	Enable signal	
7	D0	H/L	Data bus	
8	D1	H/L		
9	D2	H/L		
10	D3	H/L		
11	D4	H/L		
12	D5	H/L		
13	D6	H/L		
14	D7	H/L		

HANTRONIX Page 64

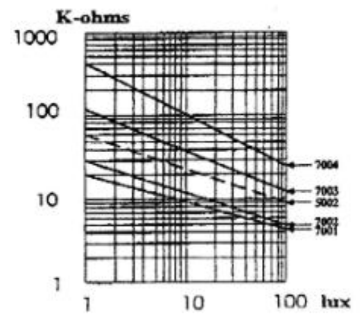
Photoresistor

Jameco Part Number 202454

Plastic-coated Types (7P 10p Types

Type No.	Out-line	Applied Voltage at 25℃ (Vdc)	Allowable Power Dissipation at 25℃ (mW)	Ambient Temperature Ta (℃)	Cell Resistance A 10 lx (at 2856K) Min. (KΩ)	Cell Resistance A 10 lx (at 2856K) Max. (KΩ)	Cell Resistance A 0 lx B Min. (MΩ)	C 100 ~10 lx Typ.	Response Time at 10 lx D Rise Time Type (ms)	Response Time at 10 lx D Decay Time Typ. (ms)
7001		200	150	-30~+75	3.6	14.4	0.3	0.6	50	20
7002		200	150	-30~+75	4	20	0.5	0.65	55	20
7003		200	150	-30~+75	8	24	0.5	0.7	55	20
7004		200	150	-30~+75	15	60	0.5	0.7	60	25
7005		200	150	-30~+75	50	150	20	0.85	60	25
5001		350	400	-30~+75	8	16	0.3	0.6	55	25
5002		350	400	-30~+75	12	30	0.5	0.75	55	25
5003		350	400	-30~+75	12	58	1	0.75	55	25

· Cell resistance vs. illuminance



A.Measured with the light source of a tungsten lamp operated at a color temperature of 2856K.

B.Measured 10 seconds after removal of incident illuminance of 10 lux.

C.Gamma characteristic between 10 lux and 100 lux and gievn by

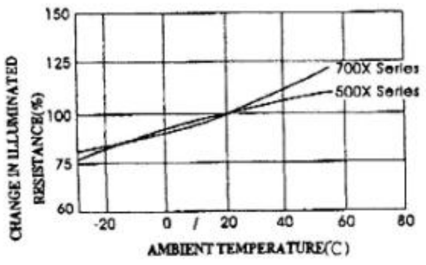
$$= \frac{\log(R100)-\log(R10)}{\log(E100)-\log(E10)}$$

Where R100, R10: cell resistances at 100 lux and 10 lux re spectively
E100, E10: illuminances of 100 lux and 10 lux respec tively

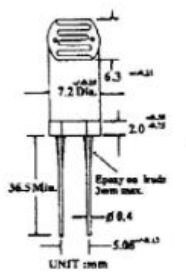
D.The rise time is the time required for the cell conductance to rise to 63% of the saturated level. The decay time is the time required for the cell conductance to decay from the saturated level to 37%.

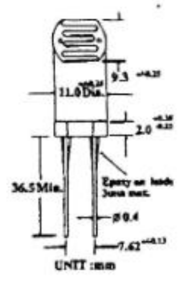
E. All characteristics are measured with the light history conditions: the CdS cell is exposed to light (100 to 500 lux) for one to two hours.

· Cell resistance vs. temperature

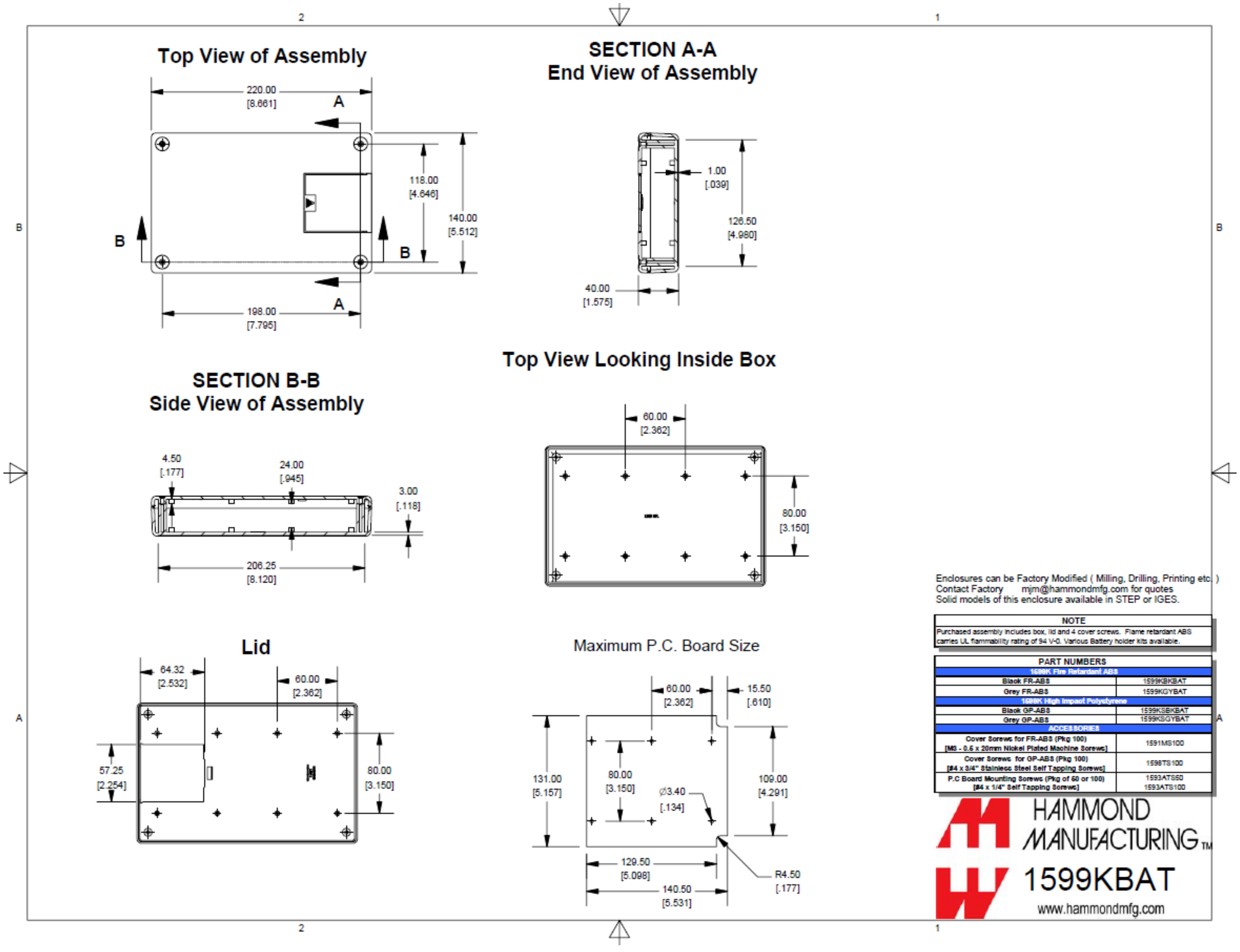


· Out-line Dimension.





Project Box



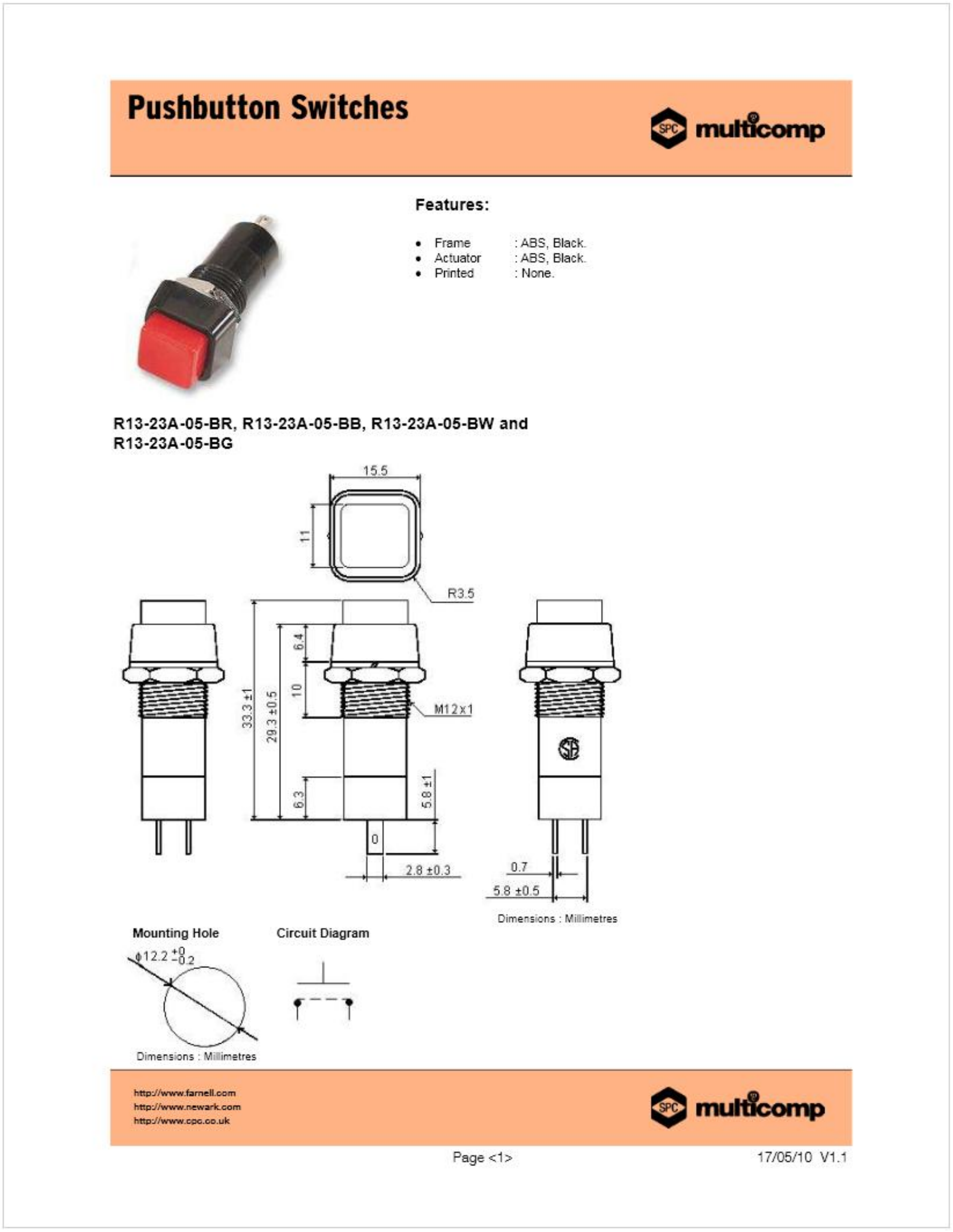
Enclosures can be Factory Modified (Milling, Drilling, Printing etc.)
Contact Factory mjm@hammondmfg.com for quotes
Solid models of this enclosure available in STEP or IGES.

NOTE
Purchased assembly includes box, lid and 4 cover screws. Flame retardant ABS carries UL flammability rating of 94 V-0. Various Battery holder kits available.

PART NUMBERS	
1599K Fire Retardant ABS	
Black FR-ABS	1599KBKBAT
Grey FR-ABS	1599KGYBAT
1599K High Impact Polystyrene	
Black GP-ABS	1599KSBKBAT
Grey GP-ABS	1599KSGYBAT
ACCESSORIES	
Cover Screws for FR-ABS (Pkg 100) [M3 - 0.5 x 20mm Nickel Plated Machine Screws]	1591MS100
Cover Screws for GP-ABS (Pkg 100) [#4 x 3/4" Stainless Steel Self Tapping Screws]	1598TS100
P.C Board Mounting Screws (Pkg of 50 or 100) [#4 x 1/4" Self Tapping Screws]	1593ATS50 1593ATS100

HAMMOND MANUFACTURING™
1599KBAT
www.hammondmfg.com

Push Button



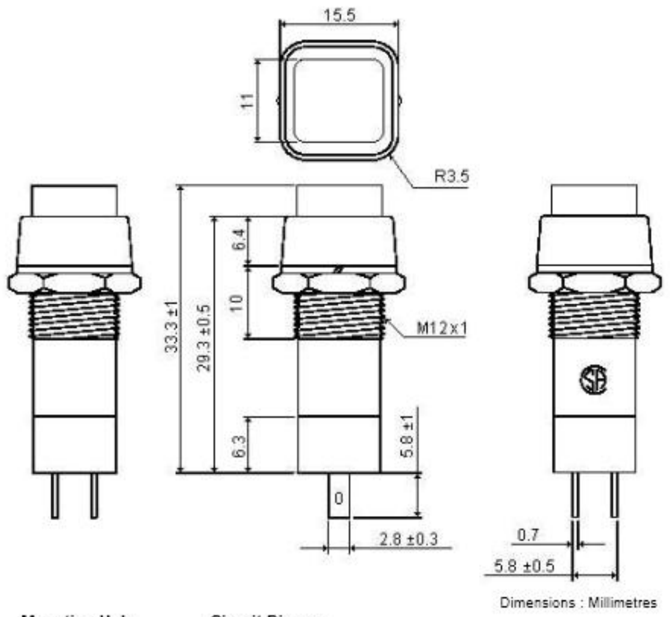
Pushbutton Switches

multicomp

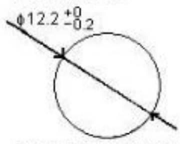
Features:

- Frame : ABS, Black.
- Actuator : ABS, Black.
- Printed : None.

R13-23A-05-BR, R13-23A-05-BB, R13-23A-05-BW and R13-23A-05-BG

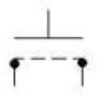


Mounting Hole



Dimensions : Millimetres

Circuit Diagram



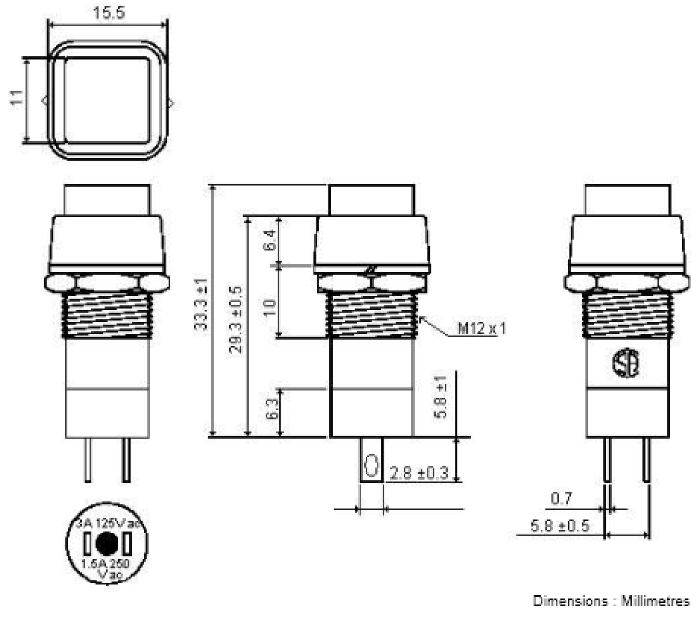
http://www.farnell.com
http://www.newark.com
http://www.cpc.co.uk

multicomp

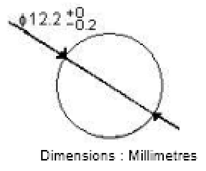
Page <1>

17/05/10 V1.1

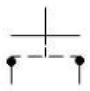
R13-23B-05-BR, R13-23B-05-BB, R13-23B-05-BW and R13-23B-05-BG



Mounting Hole



Circuit Diagram



Pushbutton Switches

Specifications:

Rating	: 1.5A 250V, 3A 12V ac. 10A 250V ac 1/2HP.
Maximum contact Resistance	: 50mΩ.
Minimum insulation Resistance	: 500V dc 100MΩ.
Dielectric strength	: 1000V ac 1 minute.
Circuit	: 2P SPST OFF-(ON)
Switch Function	: 2P SPST OFF-ON.

Part Number Table

Description	Part Number
Switch, SPST, MOM, RED	R13-23A-05-BR
Switch, SPST, MOM, Black	R13-23A-05-BB
Switch, SPST, MOM, White	R13-23A-05-BW
Switch, SPST, MOM, Green	R13-23A-05-BG
Switch, SPST, Latching, Red	R13-23B-05-BR
Switch, SPST, Latching, Black	R13-23B-05-BB
Switch, SPST, Latching, White	R13-23B-05-BW
Switch, SPST, Latching, Green	R13-23B-05-BG

Red LED

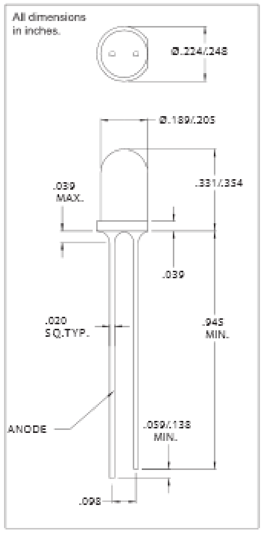
Chicago Miniature Lighting, LLC

Lighting the world since 1910

4304H Series Solid State LED Lamps-Super Brite 30 mcd T-1 3/4 (5mm)

4304H Series Solid State LED Lamps-Super Brite 30 mcd T-1 3/4 (5mm)

DESCRIPTION AND FEATURES



T-1 3/4 Super Brite LEDs
- High-intensity light output.
- Wide viewing angle.

ELECTRO-OPTICAL CHARACTERISTICS AND RATINGS

PART NUMBER	4304H1	4304H3	4304H5	4304H7
Output Color	Red	Amber	Green	Yellow
Diffusion	Diffused	Diffused	Diffused	Diffused
Package Color	Red	Amber	Green	Yellow
Test Current (mA)	10	10	10	10
Forward Voltage Typ. (V)	2.0	2.2	2.1	2.1
Forward Voltage Max. (V)	3.0	3.0	3.0	3.0
Luminous Intensity Min. (mcd)	3.0	4.0	3.0	2.5
Luminous Intensity Typ. (mcd)	6.3	20	10	6.3
Capacitance @ V=0 (pF)	45	4	20	45
Peak Wavelength (nm)	650	608	563	585
~~Viewing Angle 2θ 1/2 (degrees)~~	65	60	75	65
Power Dissipation (mW)	60	135	75	60
Reverse Breakdown Voltage Min. (V)	5	5	5	5

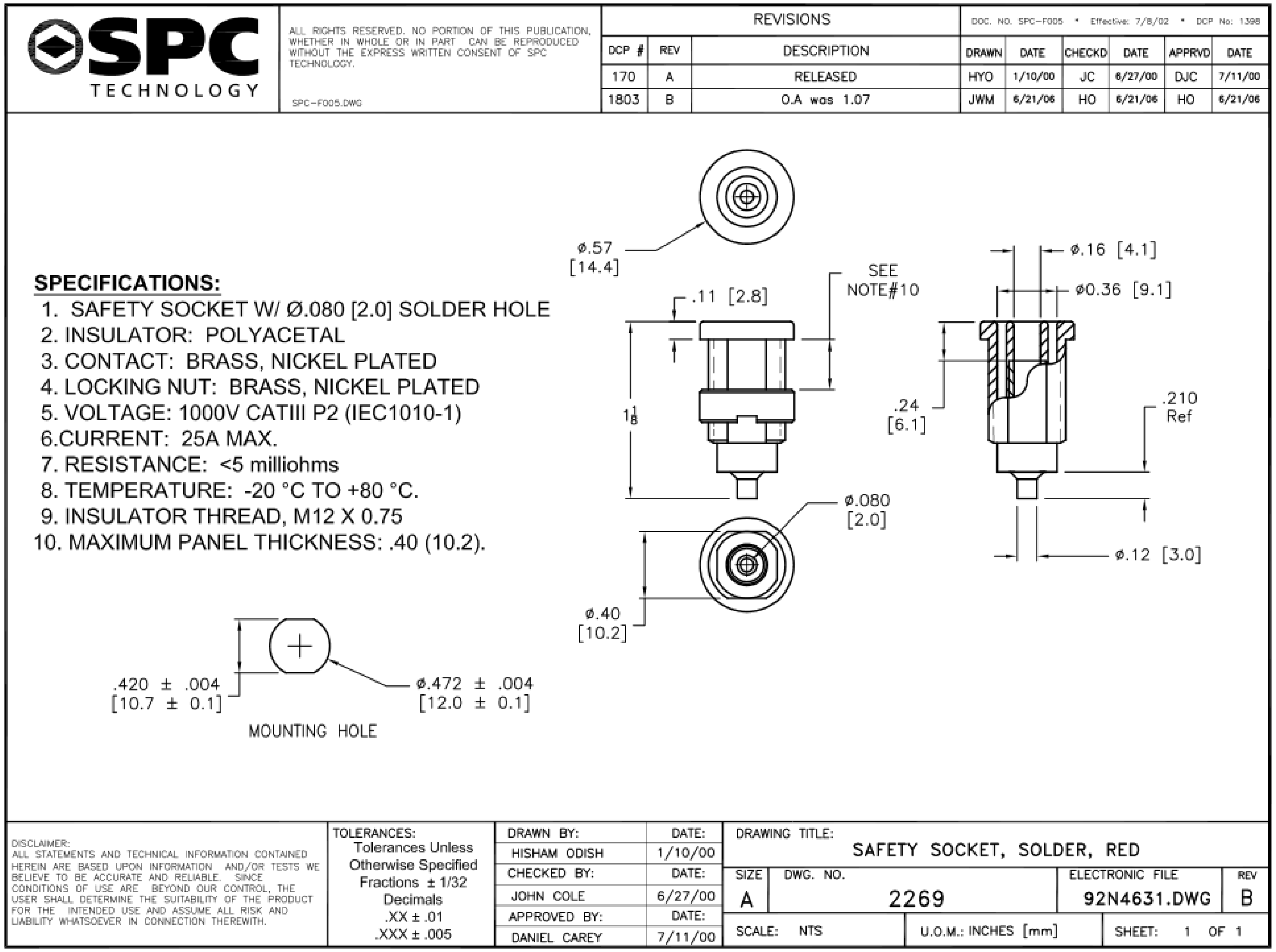
147 Central Avenue
Hackensack, NJ 07601
Phone: (201) 489-8989
Fax: (201) 489-6911

18703 Dix Toledo Road
Brownstown, MI 48193

Saret Zone Franca, Edifico B
Alajuela Costa Rica

E-mail:sales@chml.com • Web Site: www.chml.com

Safety Socket (Both)



SPC TECHNOLOGY

SPC-F005.DWG

DOC. NO. SPC-F005 • Effective: 7/8/02 • DCP No: 1398

REVISIONS

DCP #	REV	DESCRIPTION	DRAWN	DATE	CHECKD	DATE	APPRVD	DATE
170	A	RELEASED	HYO	1/10/00	JC	6/27/00	DJC	7/11/00
1803	B	O.A was 1.07	JWM	6/21/06	HO	6/21/06	HO	6/21/06

SPECIFICATIONS:

1. SAFETY SOCKET W/ Ø.080 [2.0] SOLDER HOLE
2. INSULATOR: POLYACETAL
3. CONTACT: BRASS, NICKEL PLATED
4. LOCKING NUT: BRASS, NICKEL PLATED
5. VOLTAGE: 1000V CATIII P2 (IEC1010-1)
6. CURRENT: 25A MAX.
7. RESISTANCE: <5 milliohms
8. TEMPERATURE: -20 °C TO +80 °C.
9. INSULATOR THREAD, M12 X 0.75
10. MAXIMUM PANEL THICKNESS: .40 (10.2).

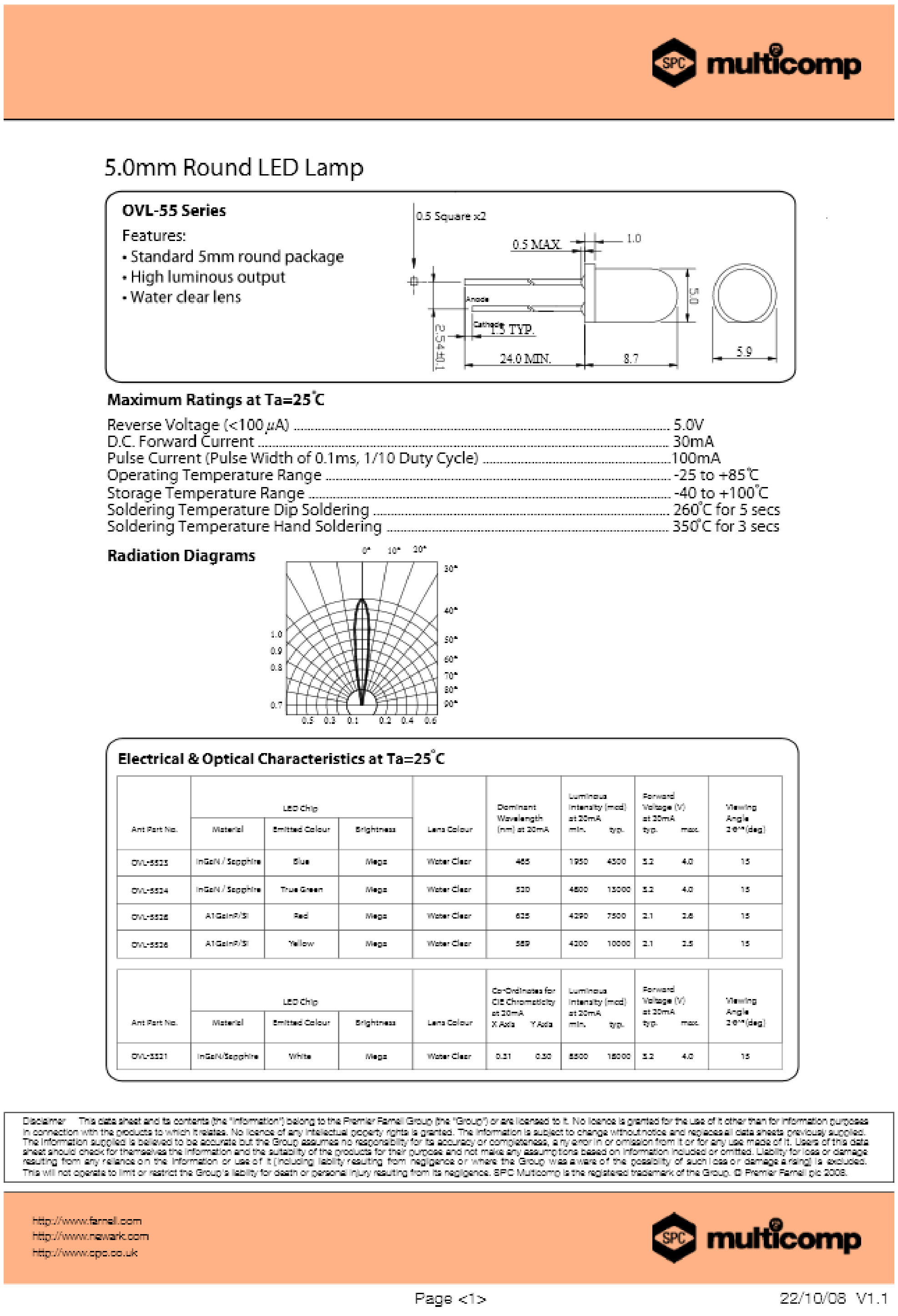
TOLERANCES:
Tolerances Unless Otherwise Specified
Fractions ± 1/32
Decimals
.XX ± .01
.XXX ± .005

DRAWN BY:	DATE:
HISHAM ODISH	1/10/00
CHECKED BY:	DATE:
JOHN COLE	6/27/00
APPROVED BY:	DATE:
DANIEL CAREY	7/11/00

DRAWING TITLE: SAFETY SOCKET, SOLDER, RED

SIZE	DWG. NO.	ELECTRONIC FILE	REV
A	2269	92N4631.DWG	B

SCALE: NTS | U.O.M.: INCHES [mm] | SHEET: 1 OF 1



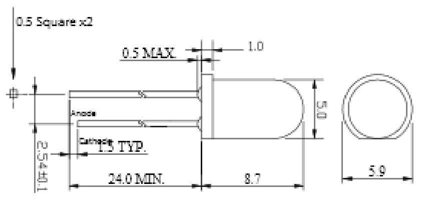
multicomp

5.0mm Round LED Lamp

OVL-55 Series

Features:

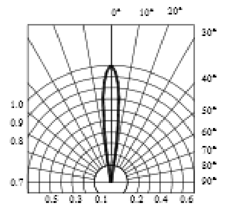
- Standard 5mm round package
- High luminous output
- Water clear lens



Maximum Ratings at Ta=25˚C

Reverse Voltage (<100 μA)	5.0V
D.C. Forward Current	30mA
Pulse Current (Pulse Width of 0.1ms, 1/10 Duty Cycle)	100mA
Operating Temperature Range	-25 to +85˚C
Storage Temperature Range	-40 to +100˚C
Soldering Temperature Dip Soldering	260˚C for 5 secs
Soldering Temperature Hand Soldering	350˚C for 3 secs

Radiation Diagrams



Electrical & Optical Characteristics at Ta=25˚C

Ant Part No.	LED Chip			Lens Colour	Dominant Wavelength (nm) at 20mA	Luminous Intensity (mcd) at 20mA		Forward Voltage (V) at 20mA		Viewing Angle 2θ½ (deg)
	Material	Emitted Colour	Brightness			min.	typ.	typ.	max.	
OVL-5523	InGaN / Sapphire	Blue	Mega	Water Clear	465	1950	4300	3.2	4.0	15
OVL-5524	InGaN / Sapphire	True Green	Mega	Water Clear	530	4800	13000	3.2	4.0	15
OVL-5528	A1GaInP/SI	Red	Mega	Water Clear	625	4290	7500	2.1	2.6	15
OVL-5526	A1GaInP/SI	Yellow	Mega	Water Clear	589	4200	10000	2.1	2.5	15

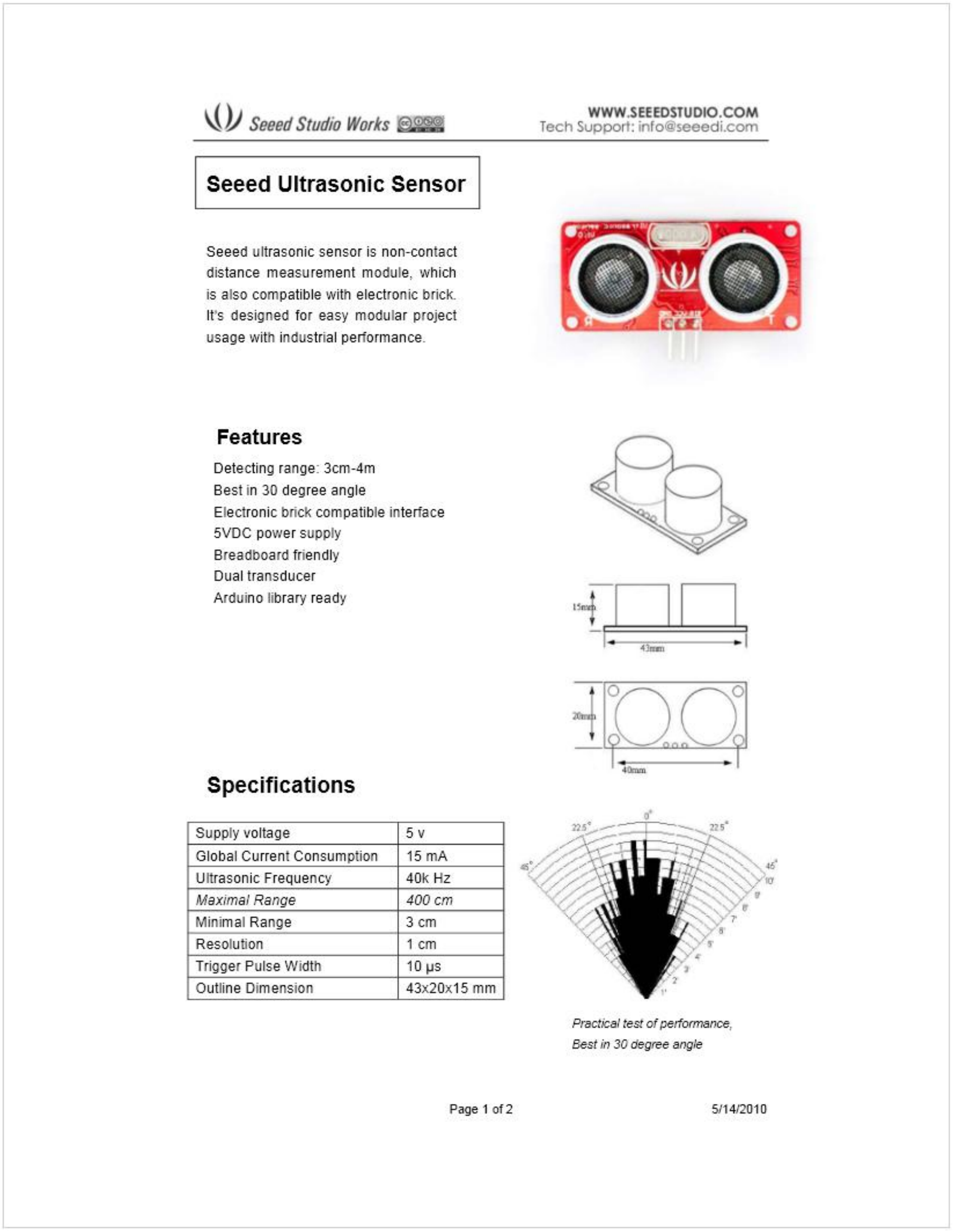
Ant Part No.	LED Chip			Lens Colour	Co-Ordinates for CIE Chromaticity at 20mA		Luminous Intensity (mcd) at 20mA		Forward Voltage (V) at 20mA		Viewing Angle 2θ½ (deg)
	Material	Emitted Colour	Brightness		X Axis	Y Axis	min.	typ.	typ.	max.	
OVL-5521	InGaN/Sapphire	White	Mega	Water Clear	0.31	0.30	8500	18000	3.2	4.0	15

http://www.farnell.com
http://www.newark.com
http://www.cpc.co.uk

multicomp

Page <1> 22/10/08 V1.1

Ultrasonic Distance Sensor



Seeed Studio Works

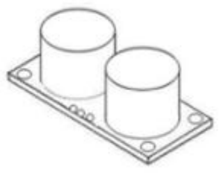
WWW.SEEEDSTUDIO.COM
Tech Support: info@seeedi.com

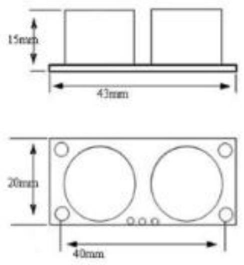
Seeed Ultrasonic Sensor

Seeed ultrasonic sensor is non-contact distance measurement module, which is also compatible with electronic brick. It's designed for easy modular project usage with industrial performance.

Features

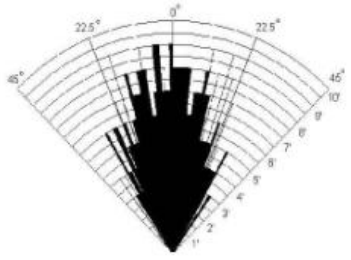
Detecting range: 3cm-4m
Best in 30 degree angle
Electronic brick compatible interface
5VDC power supply
Breadboard friendly
Dual transducer
Arduino library ready





Specifications

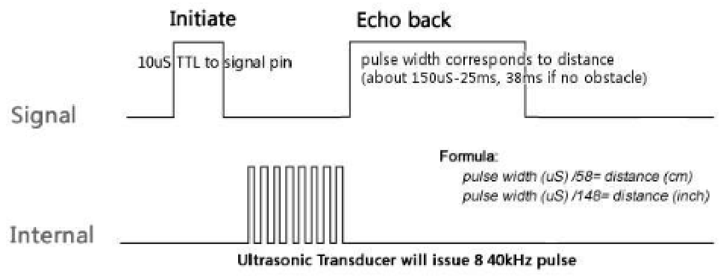
Supply voltage	5 v
Global Current Consumption	15 mA
Ultrasonic Frequency	40k Hz
Maximal Range	*400 cm*
Minimal Range	3 cm
Resolution	1 cm
Trigger Pulse Width	10 µs
Outline Dimension	43x20x15 mm



Practical test of performance,
Best in 30 degree angle

Page 1 of 2 5/14/2010

Sequence chart


Initiate
Echo back
10uS TTL to signal pin
pulse width corresponds to distance
(about 150uS-25ms, 38ms if no obstacle)
Signal
Formula:
pulse width (uS) /58= distance (cm)
pulse width (uS) /148= distance (inch)
Internal
Ultrasonic Transducer will issue 8 40kHz pulse

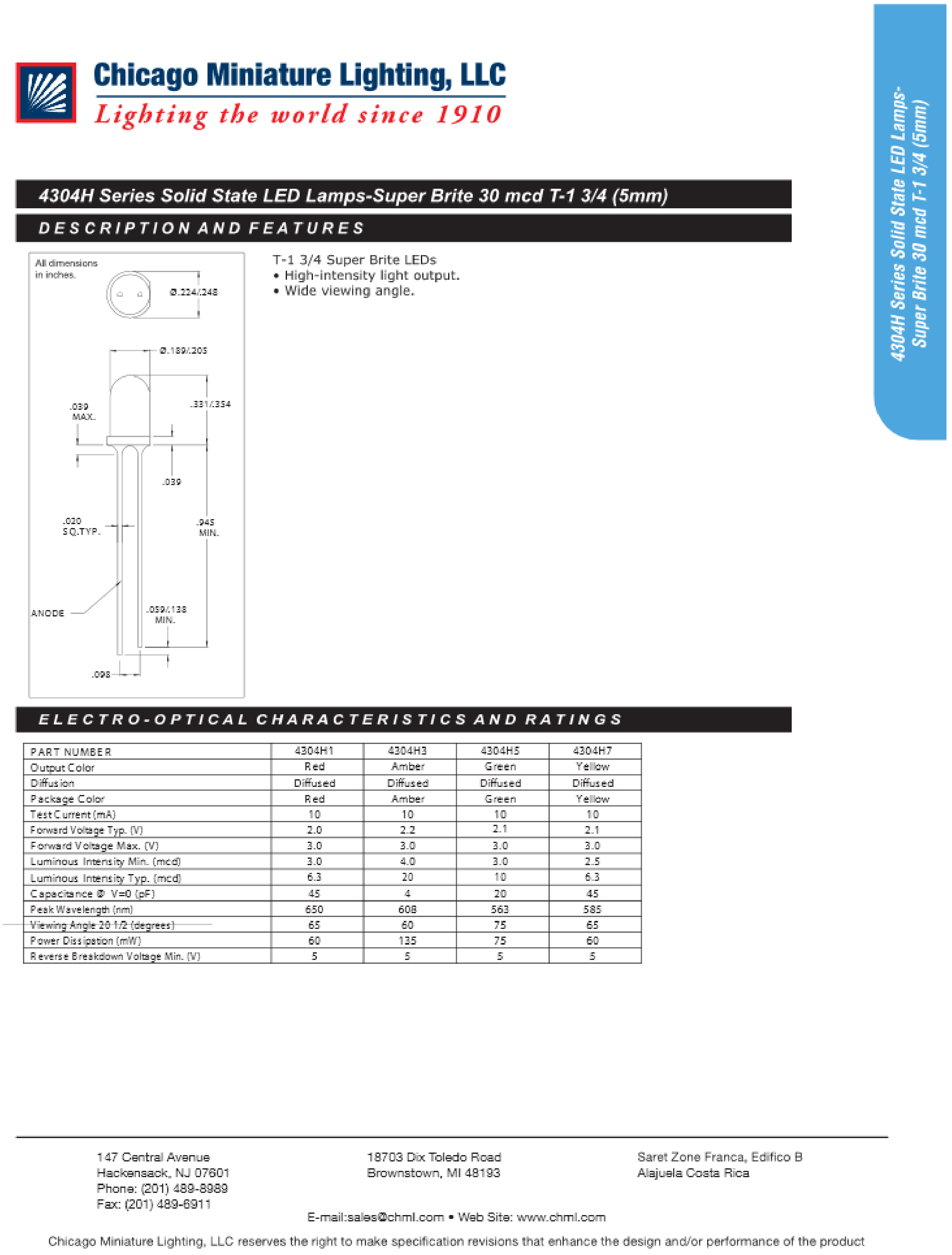
A short ultrasonic pulse is transmitted at the time 0, reflected by an object. The senor receives this signal and converts it to an electric signal. The next pulse can be transmitted when the echo is faded away. This time period is called cycle period. The recommend cycle period should be no less than 50ms.

If a 10μs width trigger pulse is sent to the signal pin, the Ultrasonic module will output eight 40kHz ultrasonic signal and detect the echo back. The measured distance is proportional to the echo pulse width and can be calculated by the formula above. If no obstacle is detected, the output pin will give a 38ms high level signal.

Revision History

Rev.	Descriptions	Release date
1.0	Seeed Ultrasonic Sensor	14.05.2010

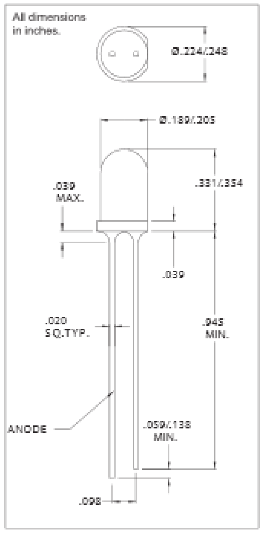
Yellow LED



Chicago Miniature Lighting, LLC
Lighting the world since 1910

4304H Series Solid State LED Lamps-Super Brite 30 mcd T-1 3/4 (5mm)

DESCRIPTION AND FEATURES



T-1 3/4 Super Brite LEDs
- High-intensity light output.
- Wide viewing angle.

ELECTRO-OPTICAL CHARACTERISTICS AND RATINGS

PART NUMBER	4304H1	4304H3	4304H5	4304H7
Output Color	Red	Amber	Green	Yellow
Diffusion	Diffused	Diffused	Diffused	Diffused
Package Color	Red	Amber	Green	Yellow
Test Current (mA)	10	10	10	10
Forward Voltage Typ. (V)	2.0	2.2	2.1	2.1
Forward Voltage Max. (V)	3.0	3.0	3.0	3.0
Luminous Intensity Min. (mcd)	3.0	4.0	3.0	2.5
Luminous Intensity Typ. (mcd)	6.3	20	10	6.3
Capacitance @ V=0 (pF)	45	4	20	45
Peak Wavelength (nm)	650	608	563	585
~~Viewing Angle 2θ 1/2 (degrees)~~	65	60	75	65
Power Dissipation (mW)	60	135	75	60
Reverse Breakdown Voltage Min. (V)	5	5	5	5

4304H Series Solid State LED Lamps-Super Brite 30 mcd T-1 3/4 (5mm)

147 Central Avenue
Hackensack, NJ 07601
Phone: (201) 489-8989
Fax: (201) 489-6911

18703 Dix Toledo Road
Brownstown, MI 48193

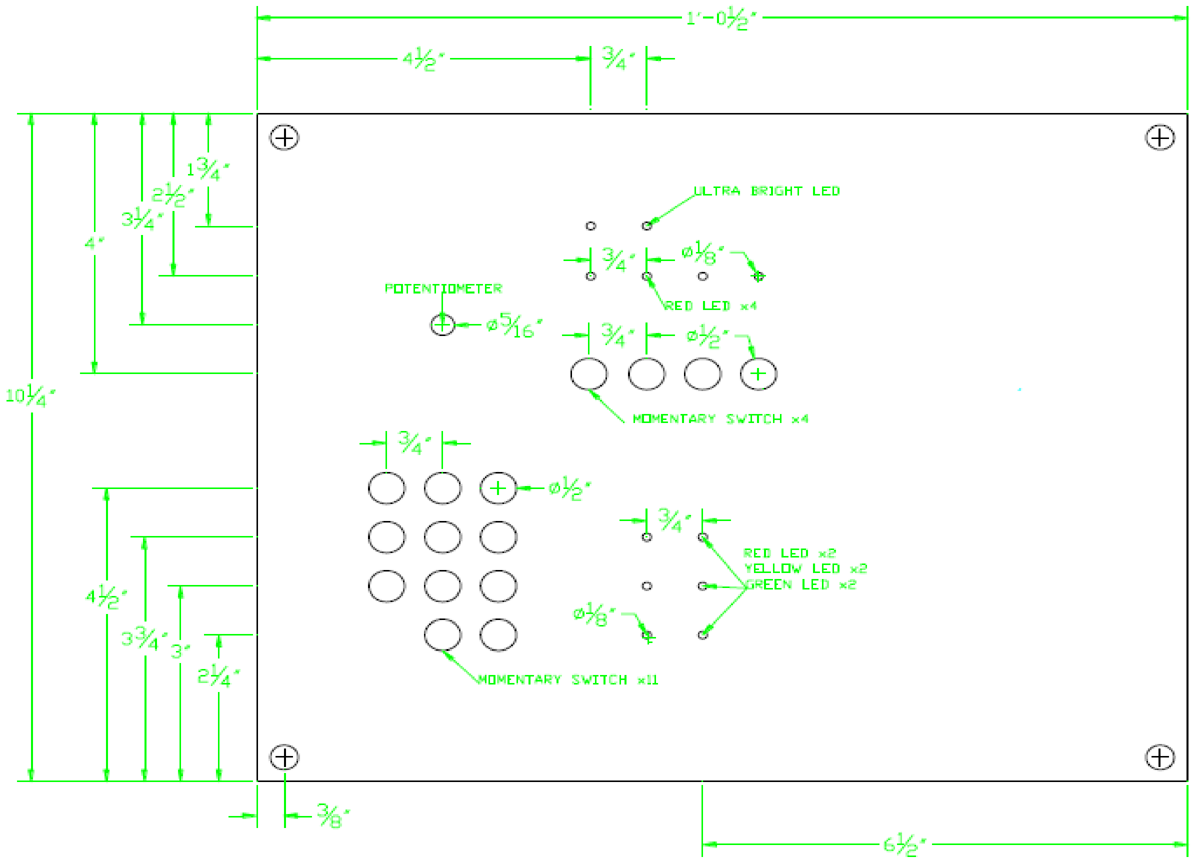
Saret Zone Franca, Edifico B
Alajuela Costa Rica

E-mail:sales@chml.com • Web Site: www.chml.com

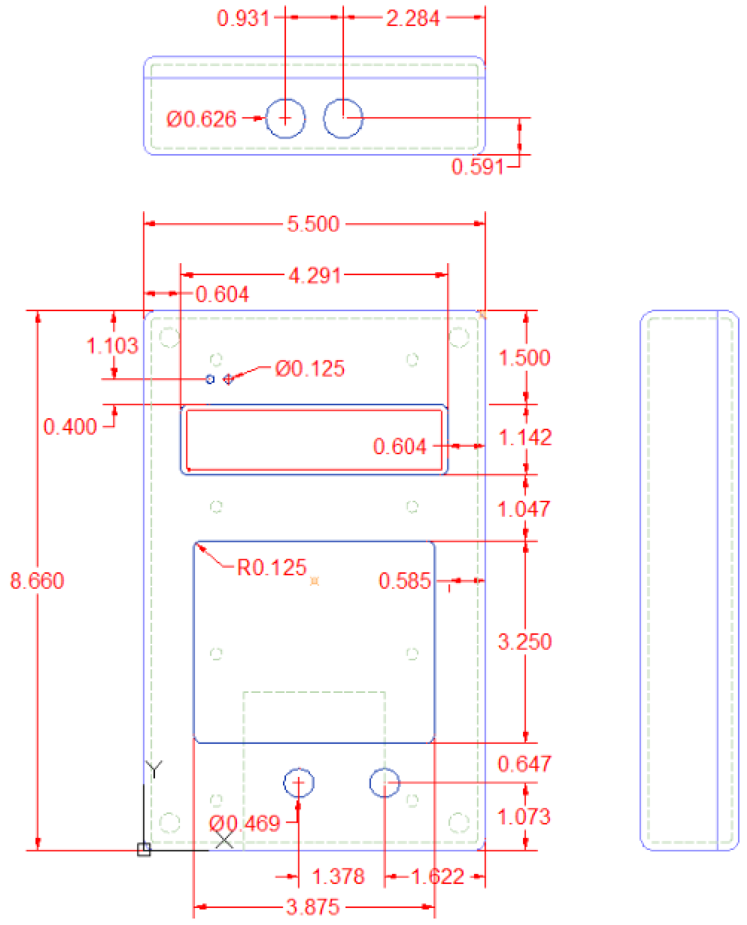
Chicago Miniature Lighting, LLC reserves the right to make specification revisions that enhance the design and/or performance of the product

Appendix E: AutoCAD Drawings

Trainer 1 AutoCAD Drawing



Trainer 2 AutoCAD Drawing



VITA

BRANDYN M. HOFFER

Personal Data:	Date of Birth: September 10, 1987
Education:	David Crockett High School, Jonesborough, Tennessee
	B.S. Engineering Technology (Electronic Engineering Technology Concentration), East Tennessee State University, Johnson City, Tennessee 2010
	M.S. Engineering Technology (Electronic Engineering Technology Concentration), East Tennessee State University, Johnson City, Tennessee 2012
Professional Experience:	Graduate Research Assistant, East Tennessee State University College of Business and Technology, 2010-2012
	Lead Student, U.S. Department of Energy Industrial Assessment Center; East Tennessee State University, Johnson City, Tennessee, 2009-2012
Honors and Awards:	Graduated Magna Cum Laude (B.S.), TELS Scholarship, SMART Grant, U.S. Department of Energy Industrial Assessment Center Certificate of Participation