

DESIGN OF LABORATORY EXPERIMENTS AND TRAINER BOARDS FOR ARDUINO USING GIZDUINO MICROCONTROLLER KIT

by

**Baldon, Michael John C.
De Leon, Kent Louis T.
Oballo, Kurt Romcel B.
Rigor, Paul Kenneth V.
Velayo, Maria Kym B.**

A Design Report Submitted to the School of Electrical Engineering,
Electronics Engineering, and Computer in Partial Fulfillment of the
Requirements for the Degree

Bachelor of Science in Computer Engineering

Mapua Institute of Technology
September 2011

Approval Sheet

Mapua Institute of Technology

This is to certify that we have supervised the preparation of and read the design report prepared by **Michael John Baldon, Kent Louis De Leon, Kurt Romcel Oballo, Paul Kenneth Rigor, and Maria Kym Velayo** entitled **Design Of Laboratory Experiments And Trainer Boards For Arduino Using Gizduino Microcontroller Kit** and that the said report has been submitted for final examination by the Oral Examination Committee.

Marites T. Fagaragan
Reader

Engr. Glenn V. Magwili
Design Adviser

As members of the Oral Examination Committee, we certify that we have examined this design report presented before the committee on **September, 2011**, and hereby recommended that it be accepted in fulfillment of the design requirements for the degree in **Bachelor of Science in Computer Engineering**.

Engr. Dionis A. Padilla
Panel Member

Engr. Jumelyn L. Torres
Panel Member

Engr. Glenn O. Avendaño
Chairman

This design report is hereby approved and accepted by the School of Electrical Engineering, Electronics Engineering, and Computer Engineering in partial fulfillment of the requirements for the degree in **Bachelor of Science in Computer Engineering**.

Dr. Felicito Caluyo
Dean, School of EECE

ACKNOWLEDGEMENT

First of all, the group is very grateful to the Lord Almighty for granting them the wisdom and knowledge that has kept them in standing up to the finish line.

The group wants to express their deepest gratitude to all those who have given their support in this design.

Their families, who gave not just financial support but also inspiration.

Their adviser, Engr. Glenn V. Magwili, for his full support to the group from the initial up to the final stage of the development of the design.

To Mr. Leonardo P. Nicdao, for allowing them to utilize the Microprocessors laboratory room.

Also, to the CDM and Electronics laboratory personnel, who helped them cut and develop their PCBs.

To their course coordinators, Engr. Noel Linsangan and Engr. Ayra Panganiban, who provided guidelines in making this design and taught them to be more responsible.

To their panels, Engr. Glenn O. Avendaño, Engr. Dionis A. Padilla, and Engr. Jumelyn L. Torres, who scrutinized the design and made them improve it.

To their peers who supported them in simple ways and contributed necessary information needed for this design.

TABLE OF CONTENTS

TITLE PAGE	i
APPROVAL SHEET	ii
ACKNOWLEDGEMENT	iii
TABLE OF CONTENTS	iv
LIST OF TABLES	vi
LIST OF FIGURES	viii
ABSTRACT	ix
Chapter 1: DESIGN BACKGROUND AND INTRODUCTION	1
Background	1
Statement of the Problem	3
Objectives of the Design	3
Significance and Impact of the Design	4
Scope and Delimitations	4
Definition of Terms	6
Chapter 2: REVIEW OF RELATED DESIGN LITERATURES AND STUDIES	7
Arduino	7
Gizduino	9
Related Studies	10
Chapter 3: DESIGN PROCEDURES	17
General Design Procedures	17
Hardware and Software Development	19
Prototype Development	21
Discussion of Components Used	27
Chapter 4: TESTING, PRESENTATION, AND INTERPRETATION OF DATA	31
Test Methodology	31
Laboratory Experiments Evaluation	33

	Trainer Board Evaluation	36
	Self-Assessment Evaluation	38
Chapter 5:	CONCLUSION AND RECOMMENDATION	41
	Conclusion	41
	Recommendation	42
REFERENCES		43
APPENDIX		
Appendix A:	Arduino Laboratory Manual	46
Appendix B:	Answer Key for Arduino Laboratory Manual Experiment 1	147
Appendix C:	Arduino Laboratory Assessment Form	157
Appendix D:	Pictures of Prototype	159

LIST OF TABLES

Table 4.1:	Weighted mean interpretation table.	32
Table 4.2:	Statements for the laboratory experiment section.	33
Table 4.3:	Laboratory experiment analysis table for experiment 1.	33
Table 4.4:	Laboratory experiment analysis table for experiment 2.	34
Table 4.5:	Laboratory experiment analysis table for experiment 3.	34
Table 4.6:	Laboratory experiment analysis table for experiment 4.	34
Table 4.7:	Laboratory experiment analysis table for experiment 5.	35
Table 4.8:	Laboratory experiment analysis table for experiment 6.	35
Table 4.9:	Laboratory experiment summary analysis table.	35
Table 4.10:	Statements for the trainer board section.	36
Table 4.11:	Trainer board analysis table for experiment 1.	36
Table 4.12:	Trainer board analysis table for experiment 2.	37
Table 4.13:	Trainer board analysis table for experiment 3.	37
Table 4.14:	Trainer board analysis table for experiment 4.	37
Table 4.15:	Trainer board analysis table for experiment 5.	37
Table 4.16:	Trainer board analysis table for experiment 6.	38
Table 4.17:	Trainer board summary analysis table.	38
Table 4.18:	Statements for the self-assessment section.	38
Table 4.19:	Self-assessment analysis table for experiment 1.	39
Table 4.20:	Self-assessment analysis table for experiment 2.	39
Table 4.21:	Self-assessment analysis table for experiment 3.	39

Table 4.22: Self-assessment analysis table for experiment 4.	39
Table 4.23: Self-assessment analysis table for experiment 5.	40
Table 4.24: Self-assessment analysis table for experiment 6.	40
Table 4.25: Self-assessment summary analysis table.	40

LIST OF FIGURES

Figure 2.1:	Arduino board. (photo courtesy of Arduino Team)	7
Figure 2.2:	A screenshot of the Arduino IDE.	8
Figure 2.3:	Gizduino. (photo courtesy of e-gizmo.com)	9
Figure 2.4:	Two photos of complete projects.	11
Figure 2.5:	Student projects comparison.	13
Figure 2.6:	Interactive system built using Arduino device.	15
Figure 3.1:	General design procedure flowchart.	17
Figure 3.2:	Hardware and software development flowchart.	20
Figure 3.3:	Trainer board 01.	21
Figure 3.4:	Trainer board 02.	22
Figure 3.5:	Trainer board 03.	23
Figure 3.6:	Trainer board 04.	24
Figure 3.7:	Trainer board 05.	25
Figure 3.8:	Trainer Board 06	26

ABSTRACT

The knowledge of using microcontrollers is necessary for engineering students since it will help them build working design prototypes which are normally required in thesis and other related subjects. The main objective of the design was to develop laboratory experiments and trainer boards for Arduino using Gizduino microcontroller kit. The developed laboratory experiments covered basic I/O, advanced I/O, analog-to-digital conversion, motor control, serial LCD display, and wireless communication. Each experiment consisted of its own objectives, discussion of the topic, trainer board reference schematic diagram, procedures and activities. The created laboratory experiments and trainer boards were administered to volunteer EECE students for testing. Likewise, evaluations on the laboratory experiments, trainer boards and self-assessment of the students were made by the volunteer EECE students after performing the experiments. The evaluation results showed that the laboratory experiments and the trainer boards helped the students in learning how to program and interface microcontrollers. Furthermore, the results on the self-assessment of the students showed that they are confident in applying the knowledge that they have learned from the experiments using Arduino and other kinds of microcontrollers. From the results, the researchers concluded that a dedicated microcontroller course that can help student learn programming and interfacing microcontrollers was successfully developed.

Chapter 1

DESIGN BACKGROUND AND INTRODUCTION

Background

A microcontroller is a small computer on a single integrated circuit which contains a processor core, memory, and programmable input/output peripherals. Since its introduction, microcontroller has been used in almost every application that requires certain amount of intelligence such as displays, printers, keyboards, modems, home appliances, and automobile engines. Because of its vast potentials and capabilities, engineering students find it ideal to use it in building their electronics projects. The knowledge of using microcontrollers is necessary for engineering students since it can help them build working design prototypes which are normally required in thesis and other related subjects.

In Mapua Institute of Technology, the courses Microprocessor Systems (COE121) and Microprocessors Systems Laboratory (COE121L) are required to be taken up by students under the School of Electrical, Electronics and Computer Engineering (EECE). Upon finishing these courses the students are expected to have learned concepts on microprocessors systems as well as their usage. However, with these courses offered, still many EECE students have difficulty on programming and interfacing microcontrollers. The said courses should include more topics that focus more on understanding, analyzing and designing microcontroller-based systems.

Because of this perceived problem, the group decided to create a laboratory course intended to teach students on how to program and interface microcontrollers. The group aimed to design laboratory experiments and trainer boards for Arduino using the Gizduino microcontroller kit. Arduino is an open-source electronics prototyping platform that uses flexible, easy-to-use hardware and software. The group chose Arduino because of its simple programming interfaces that makes it easy to learn tool for beginners, unlike the National Instruments' Freescale HCS12 Microcontroller Teaching Platform used in Microprocessor Systems Laboratory which is harder to program because of assembly language and is even impractical to use in an actual electronics project. On the other hand, instead of an actual Arduino board, the microcontroller kit that the group intended to use is Gizduino. Gizduino is a clone based on the board Arduino Diecimila; thus programming and interfacing are the same but cheaper and more accessible to students since actual Arduino boards could only be purchased overseas.

The application for this design was a formal laboratory course offered to EECE students of Mapua Institute of Technology. With this design, the students are guaranteed to learn basic and advanced input/output, analog-to-digital conversion, and interfacing of motors, serial LCDs and wireless communication devices (using XBees) using an Arduino board. Trainer boards were made available to complement the experiments. This design shall eventually aid students in designing and creating their own microcontroller-based systems.

Statement of the Problem

The main problem of this study is that there is no dedicated course in Mapua Institute of Technology that teaches students on how to program and interface microcontrollers. Specifically, it attempts to answer the following design problems:

1. Will the laboratory experiments to be proposed for Arduino help students learn programming and interfacing microcontrollers?
2. Will the trainer boards to be developed help the students in understanding the concepts presented in the laboratory experiments?

Objectives of the Design

The main objective of this study was to develop laboratory experiments and trainer boards for Arduino using Gizduino microcontroller kit. Specifically, it attempted to meet the following design objectives:

1. To design laboratory experiments for Arduino that will teach students basic and advanced input/output, analog-to-digital conversion, motor control, serial LCD display and wireless communication.
2. To design trainer boards that will help the students in understanding the concepts presented in the laboratory experiments.

Significance and Impact of the Design

The design will primarily benefit the students of the School of EECE who build electronic prototypes as a requirement for their thesis, design or other related subjects. It will, especially, benefit students who have little or no experience in programming and interfacing microcontrollers since Arduino uses flexible, easy-to-use hardware and software and has simple programming interfaces which make it an easy to learn tool for beginners.

The key impact of this design is that it will impart on the students the knowledge on microcontroller programming and interfacing which in effect shall serve as a stepping stone for them to learn other kinds of microcontrollers. In general, it shall strengthen their foundation on basic engineering concepts and will develop their capability to apply these learned concepts in engineering design.

Scope and Delimitations

Scopes:

1. The proposed laboratory experiments and the key components used for each are as follow:

Expt. #	Experiment Title	Key components
1	Basic Input/Output	DIP switch, tact switches, LEDs, seven-segment display

2	Advanced Input/Output	LED Matrix, keypad, buzzer
3	Analog-to-Digital Conversion	Light dependent resistor, potentiometers, RGB LEDs
4	Motor control	DC motor, servo motor, stepper motor
5	Serial LCD Display	16x2 Serial LCD
6	Wireless Communication	Xbee Module

2. There is one trainer board created for each experiment.

3. The parts of the experiment include the following:

- I. Experiment no. and experiment title
- II. Objectives
- III. Equipment and Materials
- IV. Discussion
- V. Trainer Board Schematic Diagram
- VI. Procedures
- VII. Activities

Delimitations:

- 1. Direct control of input/output peripherals (e.g. push buttons, LEDs, motors, etc.) via software running on a PC is not covered.
- 2. Guide questions shall be provided only on the first experiment.

3. For the experiment on wireless communication, only the minimum requirements needed to program the two XBees to be able to talk each other is presented. Advanced XBee programming is not covered.

Definition of Terms

Arduino- an open-source single-board microcontroller, designed to make the process of using electronics in multidisciplinary projects more accessible. *[Wikipedia, The Free Encyclopedia]*

Arduino Diecimila- the older model of the standard Arduino board.

Gizduino- is a tool, based on Arduino Diecimila, for implementing a program that have been designed *[E-gizmo]*

Microcontroller – are single-chip computers consisting of CPU (central processing unit), data and program memory, serial and parallel I/O (input/output), timers, external and internal interrupts *[Ibrahim, 2006]*

Windows – a series of software operating systems and graphical user interface produced by Microsoft. *[Wikipedia, The Free Encyclopedia]*

XBee - the brand name from Digi International for a family of form factor compatible radio modules. *[Wikipedia, The Free Encyclopedia]*

ZigBee - a specification for a suite of high level communication protocols using small, low-power digital radios based on an IEEE 802 standard for personal area networks. *[Wikipedia, The Free Encyclopedia]*

Chapter 2

REVIEW OF RELATED LITERATURE

Arduino

Arduino is an open-source electronics prototyping platform. According to Sarik and Kymissis in 2010, the Arduino provides a complete, flexible, easy-to-use hardware and software solution for designing take-home exercises. The exercises may be limited due to the availability of the other hardwares but it is possible to design cost-efficient and instructive exercises. It is widely used by artists, designers, and hobbyist.

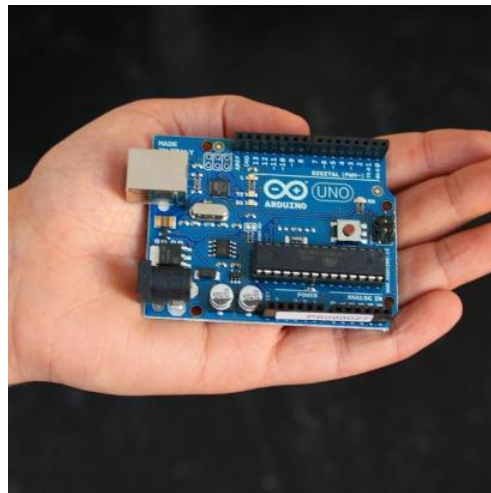


Figure 2.1. Arduino board (photo courtesy of Arduino team)

According to the Arduino team, the Arduino has the capability of sensing the environment by receiving input from a variety of sensors and can affect its surrounding by controlling lights, motors, and other actuators. The arduino board contains a microcontroller as seen on figure 2.1, taken by the arduino team.

Software

The microcontroller on the board is programmed using the Arduino programming language which is based on C/C++. As stated by Durfee and Waletzko in 2005, the Arduino integrated development environment (IDE) is available for Windows, OS X, and Linux systems and comes with numerous tutorials and extensive documentation. Students can familiarize themselves with the platform at their own place and use the troubleshooting pages to quickly and easily solve common problems.

A screenshot of the Arduino IDE window titled "Arduino - 0011 Alpha". The window has a menu bar with "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for running, stopping, saving, and other functions. The main text area displays the "Blink" sketch, which is a basic Arduino example that turns an LED on for one second, then off for one second, and so on. The code is as follows:

```
/*  
 * Blink  
 *  
 * The basic Arduino example. Turns on an LED on for one second,  
 * then off for one second, and so on... We use pin 13 because,  
 * depending on your Arduino board, it has either a built-in LED  
 * or a built-in resistor so that you need only an LED.  
 *  
 * http://www.arduino.cc/en/Tutorial/Blink  
 */  
  
int ledPin = 13;          // LED connected to digital pin 13  
  
void setup()              // run once, when the sketch starts  
{  
  pinMode(ledPin, OUTPUT); // sets the digital pin as output  
}  
  
void loop()              // run over and over again  
{  
  digitalWrite(ledPin, HIGH); // sets the LED on  
  delay(1000);                // waits for a second  
  digitalWrite(ledPin, LOW);  // sets the LED off  
  delay(1000);                // waits for a second  
}
```

The status bar at the bottom of the window shows "Done compiling." and "Binary sketch size: 1098 bytes (of a 14336 byte maximum)". The line number 22 is visible in the bottom left corner.

Figure 2.2. A screenshot of the Arduino IDE

Gizduino

Figure 2.3 shows a Gizduino that is based on the popular Arduino Diecimila, an open source computing platform based on a simple I/O board for implementing a program that is designed. It is ideal for beginner programmers and hobbyist because of its simplicity compared to other platforms. It can run on Windows, Macintosh, and Linux. It is programmable via USB cable, which makes it more accessible and allows communication with the computer. The microcontroller used in Gizduino is the ATmega168 which also has 16KB flash memory, 2KB of which is used for the boot-loader. It has 1KB SRAM and 512 bytes of EEPROM. There are 14 digital I/O pins, 6 of which are also analog output pins; pins 3, 5, 6, 9, 10 and 11. Pin 13 has a built in LED in the board. Analog input pins are used for reading external components with varying voltage reading such as sensors.

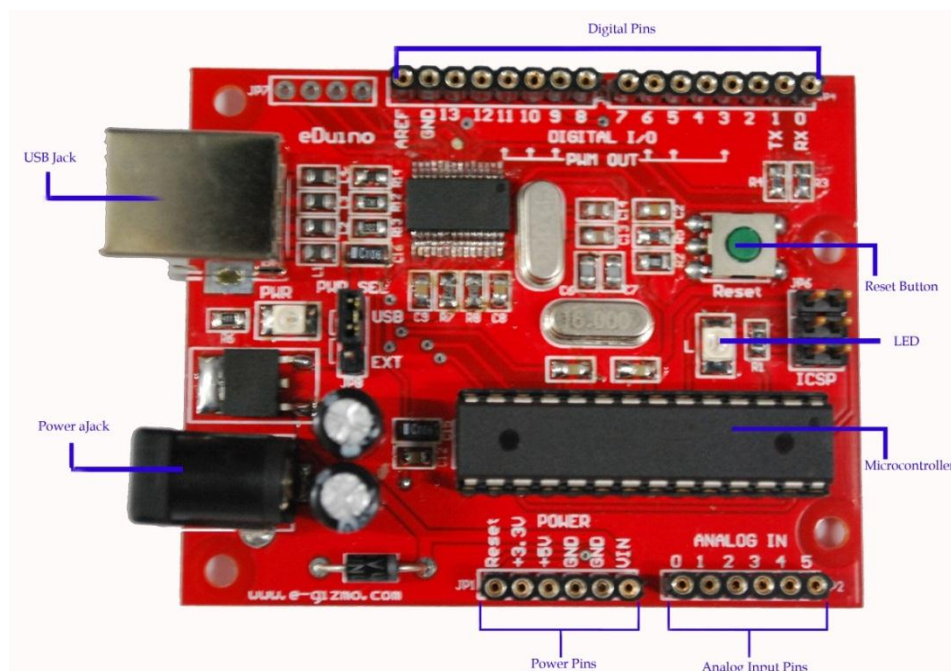


Figure 2.3. Gizduino (photo courtesy of e-gizmo.com)

Both Arduino and Gizduino microcontroller kits have the same functions in terms of its hardware and software. The Gizduino shall be used instead of the Arduino, because of its cheapness and accessibility to students here in the Philippines.

Related Studies

The main aim of the study was to equip the students the knowledge and strengthen their foundation on basic engineering concepts and one of which is microcontroller programming and interfacing. In order to do so, the study intended to teach students on how to program and interface microcontrollers by designing laboratory experiments and trainer boards for Arduino using the Gizduino microcontroller kit. Revision of different Microcontroller courses has common objectives. First, educators want students to understand the basic concepts and fundamentals of programming microcontrollers using languages which they prefer. Secondly, educators want students to have a basic understanding of various constraints in embedded systems since these are found in every aspect of people's daily lives these days. Lastly, educators want students to gain comprehensive view of various topics they have learned in their previous years. With the revised courses, students are expected to be able to design complex and quality systems. (Peng, J., 2009)

In a study about improving the effectiveness on microcontroller education, it was concluded that a successful microcontroller course is effective for equipping students to produce large, complex systems in their design courses. Educators admit that it is very challenging to teach microcontroller programming and hardware interfacing because of the complexity of the system. In addition to this, it is a time-consuming task for them. Also, hands-on laboratory experience is very significant for this part is where the students apply what they have learned. In conducting the study, the participants school, particularly the Mississippi State University shifted its microprocessor course from the traditional assembly language orientation to one that emphasizes embedded system concepts and hardware/software prototyping skills. After several transitions and modifications of the course, measurable improvements in the complexity and quality of projects produced by the students were seen.



Figure 2.4. Two photos of complete projects.

From the figure 2.4, the left photo, produced after the improvement of microcontroller courses was done, is an electromagnetic field detection circuit with audible buzzer that is built from simple discrete and ICs. The photo on the right is

from a 'smart light' controller, produced before the improvements were done. The latter project was said to be one of the experiments of the newly improved curriculum. (Jones and Reese, 2010) The improvement of microprocessor courses evidently resulted into good performance of the students enrolled in it.

Due to the rapid advancement of technology on embedded systems, it became necessary for engineering students, especially electrical and computer engineering students to take an introductory course with the area on microcontroller and microprocessor design. The purpose of the laboratory is to teach students how to build real projects and learn how to interface mostly used components in their designs. It was expected that upon successful completion of all the laboratory experiments, students will be able to extend their senior projects to a higher level of complexity, since they have acquired most of the basic learning during the experiments. (M.Hamad, A. Kassem, et.al., 2006)

Because of the flexibility and easy-to-use characteristics of the Arduino, the group considered to make use of it as the basic learning tool for beginners. The study entitled "Lab Kits Using the Arduino Prototyping Platform" of Sarik J., and Kymissis, I. presents a lab kit platform based on an Arduino microcontroller board. It will be used by students to complete their laboratory exercises at home when there is not enough time at school. It is said that most universities are already moving more courses online, and this lab kits will be used. It was assumed that students would have different backgrounds and different programming and prototyping abilities. The lab kits had to be accessible to students to perform

meaningful exercises. This is the reason why the kits were designed around the Arduino open-source electronics prototyping platform. Arduino provides a complete, flexible, easy-to-use hardware and software platform what is widely used by artists, designers and hobbyist. The kits use the Arduino Duemilanove, a microcontroller board with fourteen digital input/output pins, six analog inputs, and a USB port. The board is programmed using the Arduino language, which is based on C/C++.

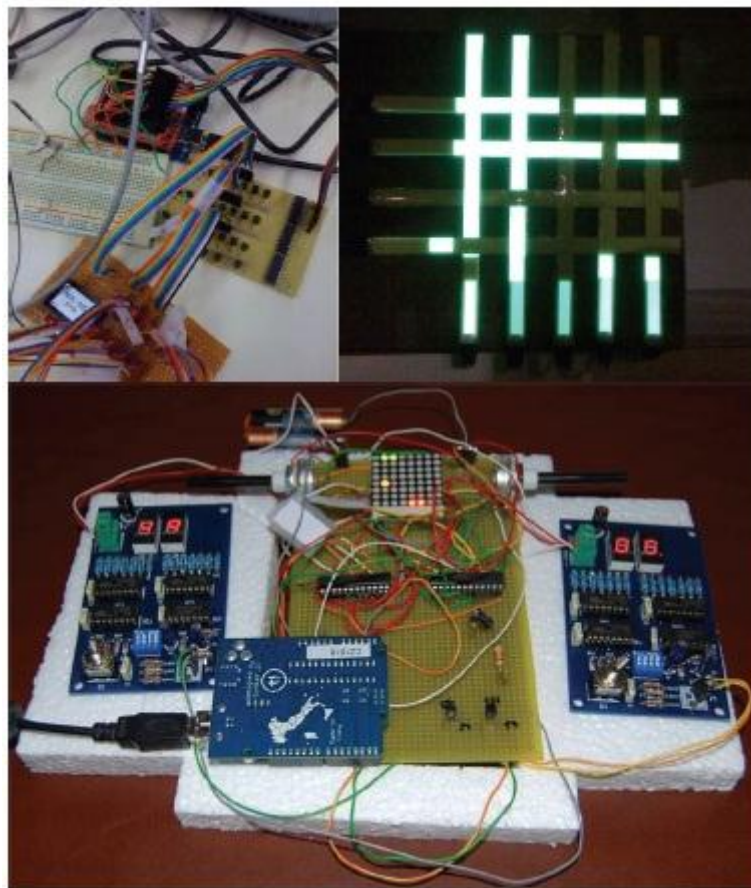


Figure 2.5. Student projects comparison.

The figure 2.5 shows two student projects made on-campus and using the labkit. The first project, shown on top, was built by two on-campus students. The students made use of 9 three-segmented EL displays to create a unique clock. The

second project, built by a remote student, implemented a game pong using dot matrix LED display, 7-segment LED displays and potentiometers. Both projects were well documented and the students demonstrated a solid understanding of the capabilities and limitations of their displays. In general, the on-campus students used resources available in the laboratory to experiment with new hardware, while the remote student used the well-established hardware from the labkit and focused on adding complexity by integrating more components and improving the control software. (Kymissis and Sarik, 2010)

Arduino is widely used in different categories of system developments. Negru, S. developed a conceptual architecture of an Arduino-based social-emotional interactive system. The study focused mainly on the social and emotional aspects of the human-computer interaction. The researcher considered building a new platform in which it can properly make use of emotions. Gathering data about emotions and properly identifying each and every single one can be time consuming and costly, as these are very hard to classify and understood by computers. Facial expressions, voice patterns, physical sensors, body gestures are just a few ways in which computers can identify emotions or behavior.

The interactive system is personalized for specific user. Giving it the ability to collaborate with other similar systems provides another important characteristic of an interactive system that of allowing the users to collaborate with one another. Their emotions are more intense in a group of people, than of an individual interacting with the system.

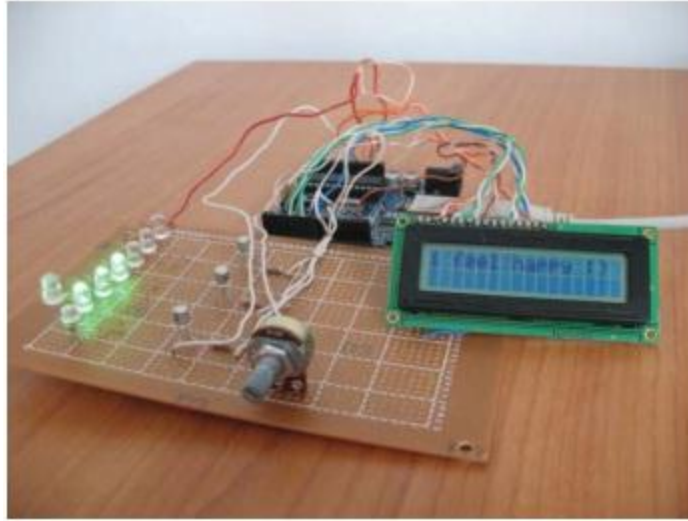


Figure 2.6. Interactive system built using Arduino device

The Arduino microcontroller, shown in figure 2.6, can be used to develop a physical user interface for the system, taking inputs from a variety of switches or sensors and controlling a variety of lights, motors, and other outputs from the environment. Making the interaction with any type of device more humane, represents a great step in improving the way we interact and use computers and devices. (Negru S., 2010)

The popularity of the Arduino in various system developments created an idea to create another programming environment for it. A visual programming environment for Arduino was created by Kato, Y. and named it Splish. It enables an icon-based visual programming to develop a program which runs on a microcontroller board family called Arduino. This programming environment is somehow similar to the standard Arduino IDE for it is used to compile and transfer code to the microcontroller board. It is said that physical computing attracts wide variety of people including those non-specialists and students, the visual

programming and the interactive debugging capabilities of Splish will accelerate the physical computing experience.

Chapter 3

DESIGN PROCEDURES

General Design Procedures

The general step-by-step procedure in developing the design is shown in the flowchart below:

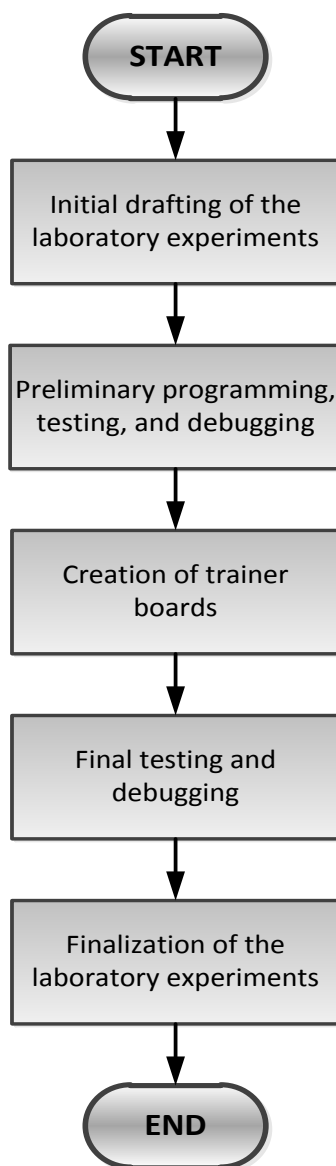


Figure 3.1. General Design Procedure Flowchart

The detailed discussions on each step are as follows:

1. Initial drafting of the laboratory experiments

- Objectives are formulated and exercises are created for each experiment.
Initial circuit schematic diagrams to be used are also designed.

2. Preliminary programming, testing, and debugging

- The Arduino program source codes for the exercises in each experiment are created. These exercises are performed and tested by downloading the source code to Gizduino and building the corresponding circuit diagram to be used on breadboard. Necessary adjustments to the program source codes and circuit schematic diagrams are also performed.

3. Creation of trainer boards

- With the circuit schematic diagrams for each experiment been designed and tested successfully, the circuit designs are transferred into printed circuit boards. Creating PCB layouts, etching, drilling, soldering, and debugging PCB connectivity are all performed in this step.

4. Final testing and debugging

- All the laboratory experiments are performed following the laboratory manual draft created earlier as reference. The trainer boards are tested and debugged for problems until all are successfully working.

5. Finalization of the laboratory experiments

- The initial draft for the laboratory experiments are finalized as a laboratory manual, following the format below for each laboratory experiment:

- I. Experiment no. and experiment title
- II. Objectives
- III. Equipment and Materials
- IV. Discussion
- V. Trainer Board Reference Schematic Diagram
- VI. Procedures
- VII. Activities

Appendices are also included in the finalized laboratory manual for reference.

Hardware and Software Development

The development of each laboratory experiment commenced by outlining the objectives and then creating the exercises which would be in line with the objectives of the said experiment.

The hardware and software development for all exercises of all laboratory experiments follow the flowchart below:

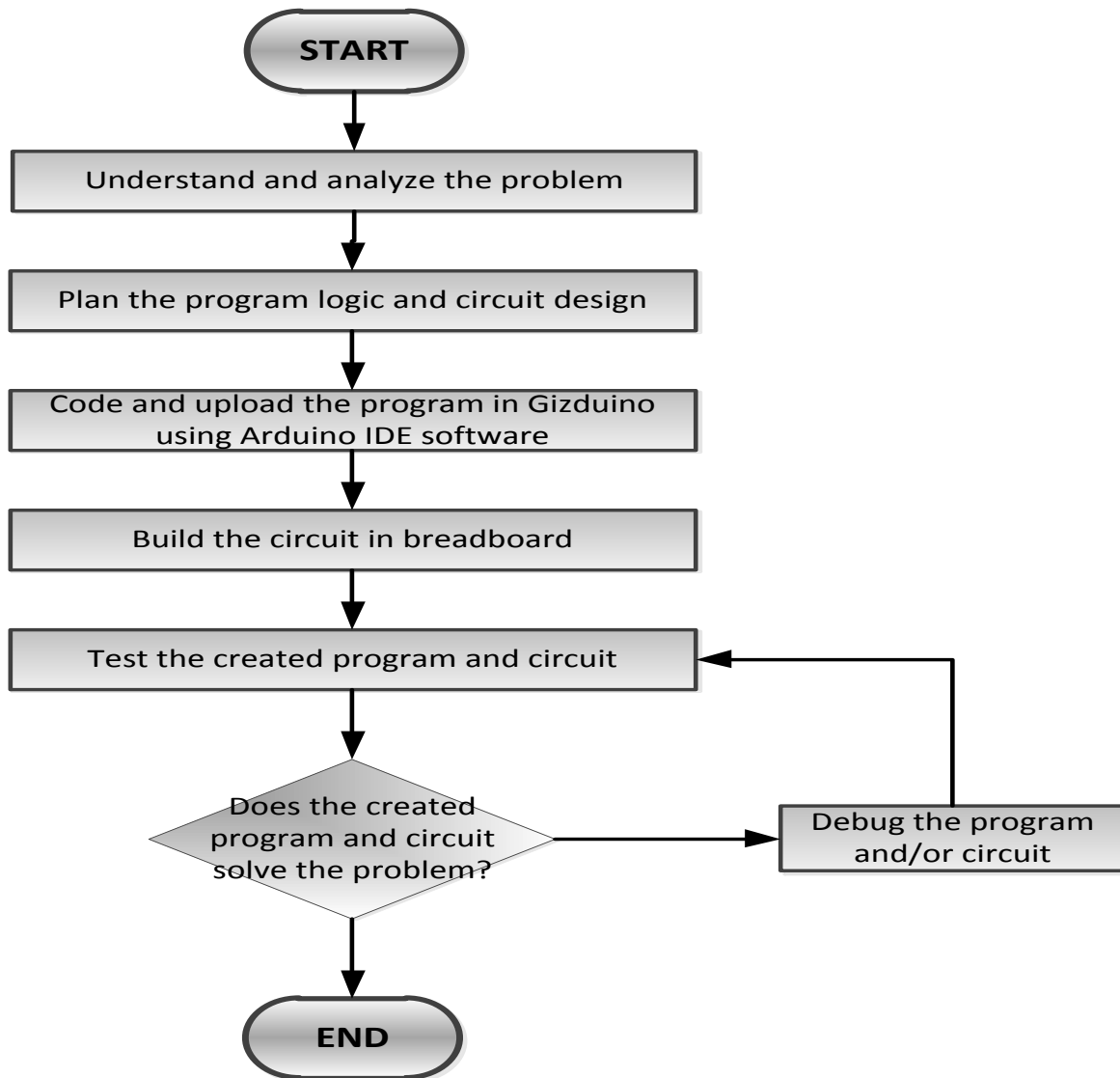


Figure 3.2. Hardware and Software Development Flowchart

Prototype Development

The schematic diagrams of the trainer boards for each experiment are shown on the Figures 3.3 to 3.8:

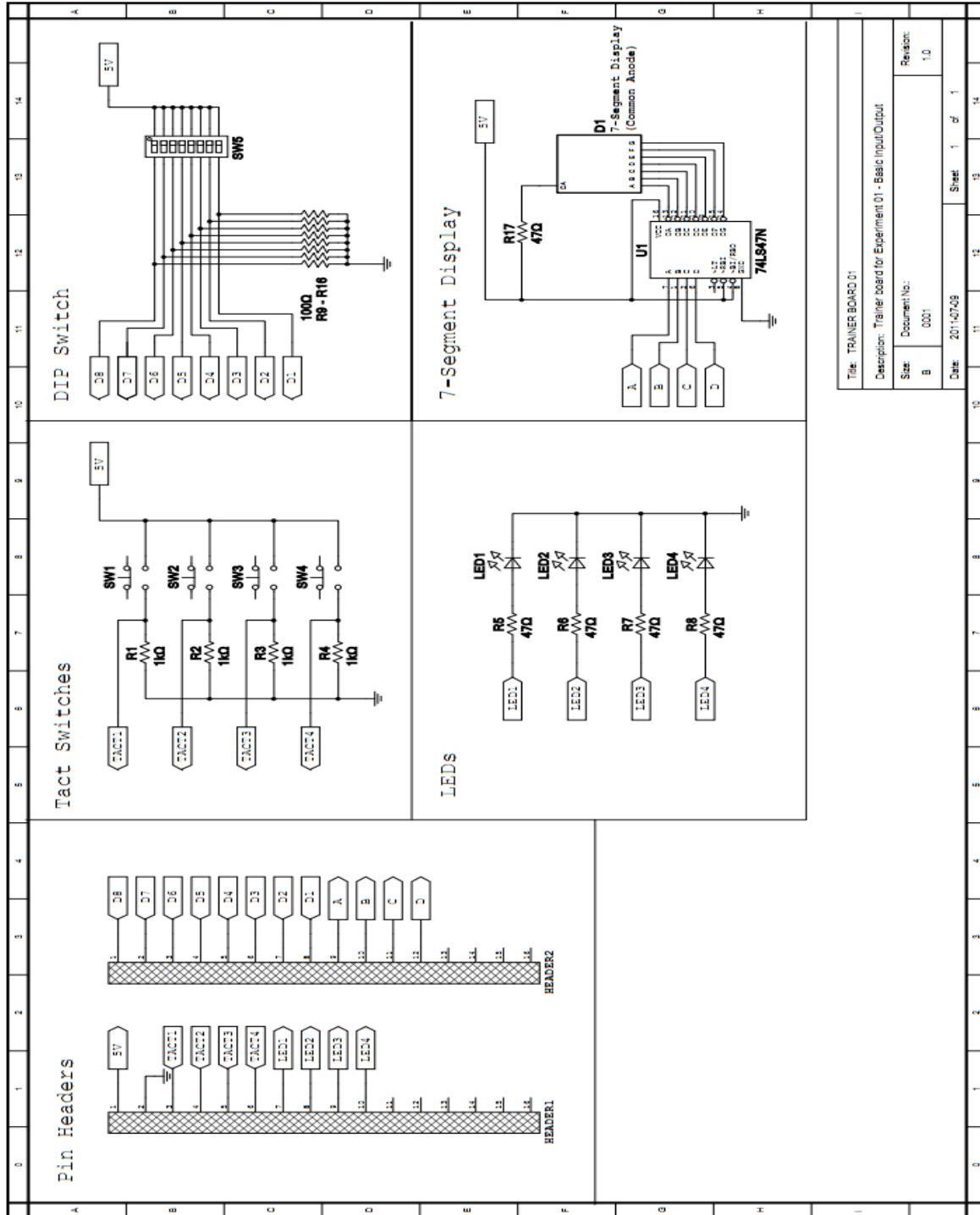


Figure 3.3. Trainer Board 01

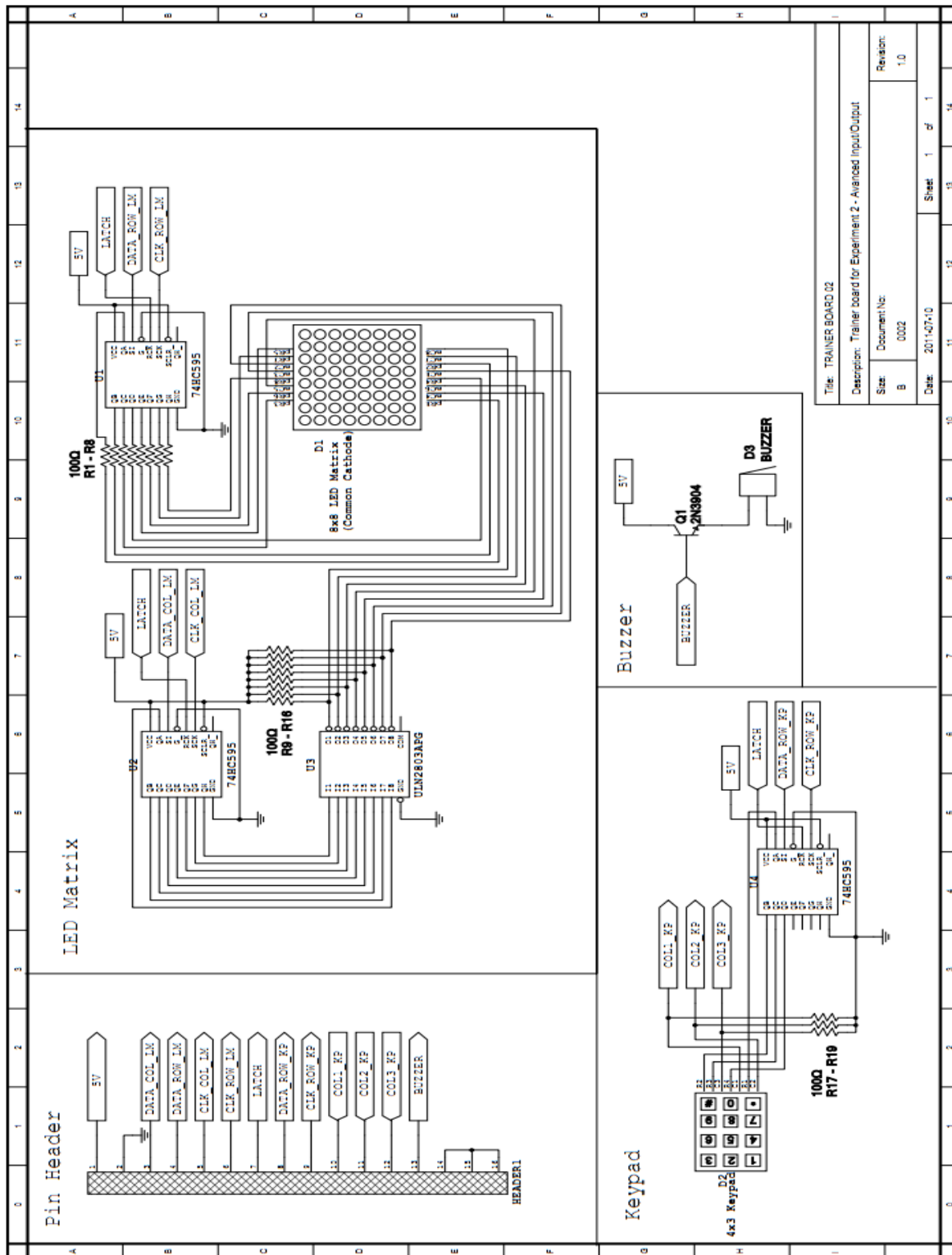


Figure 3.4. Trainer Board 02

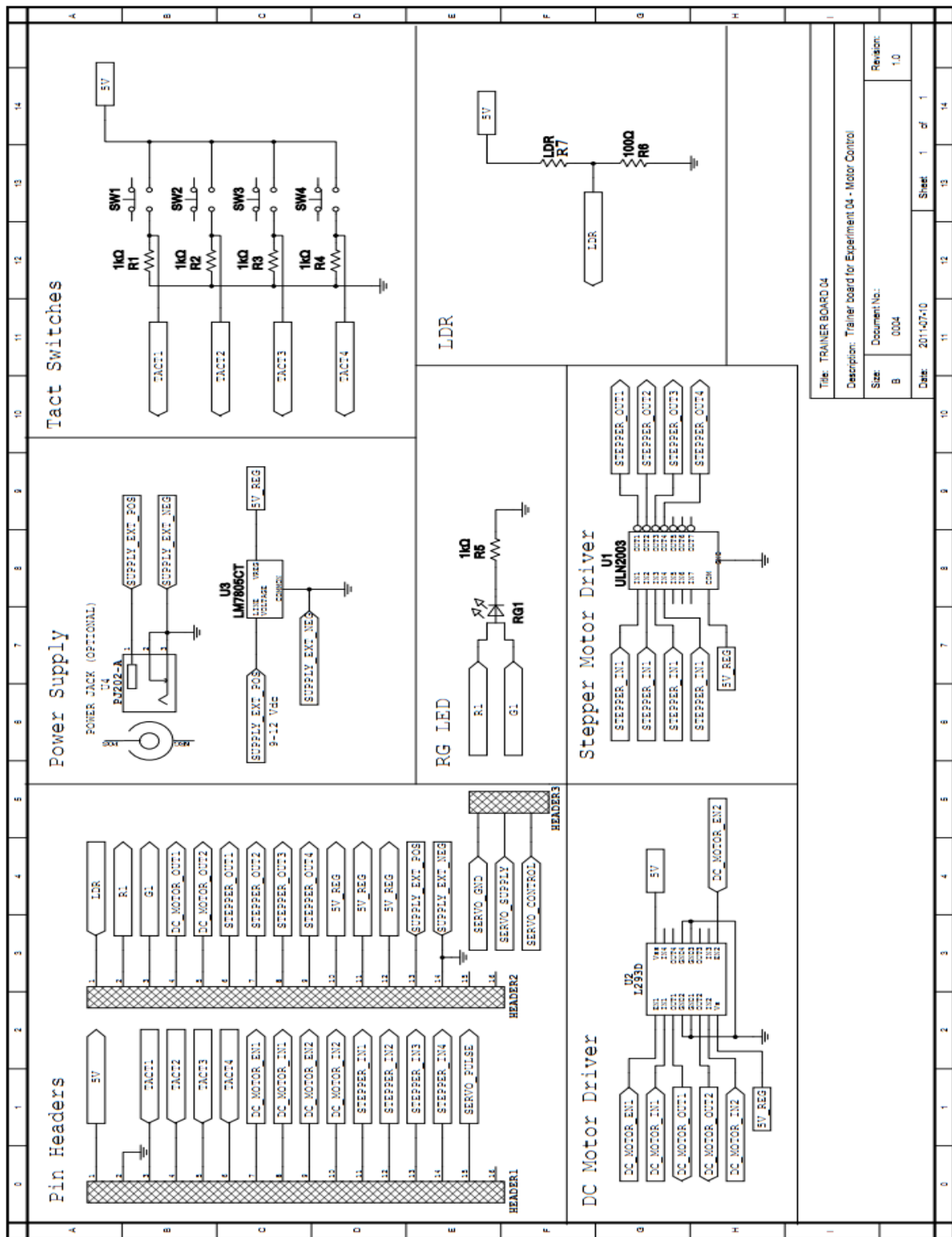


Figure 3.6. Trainer Board 04

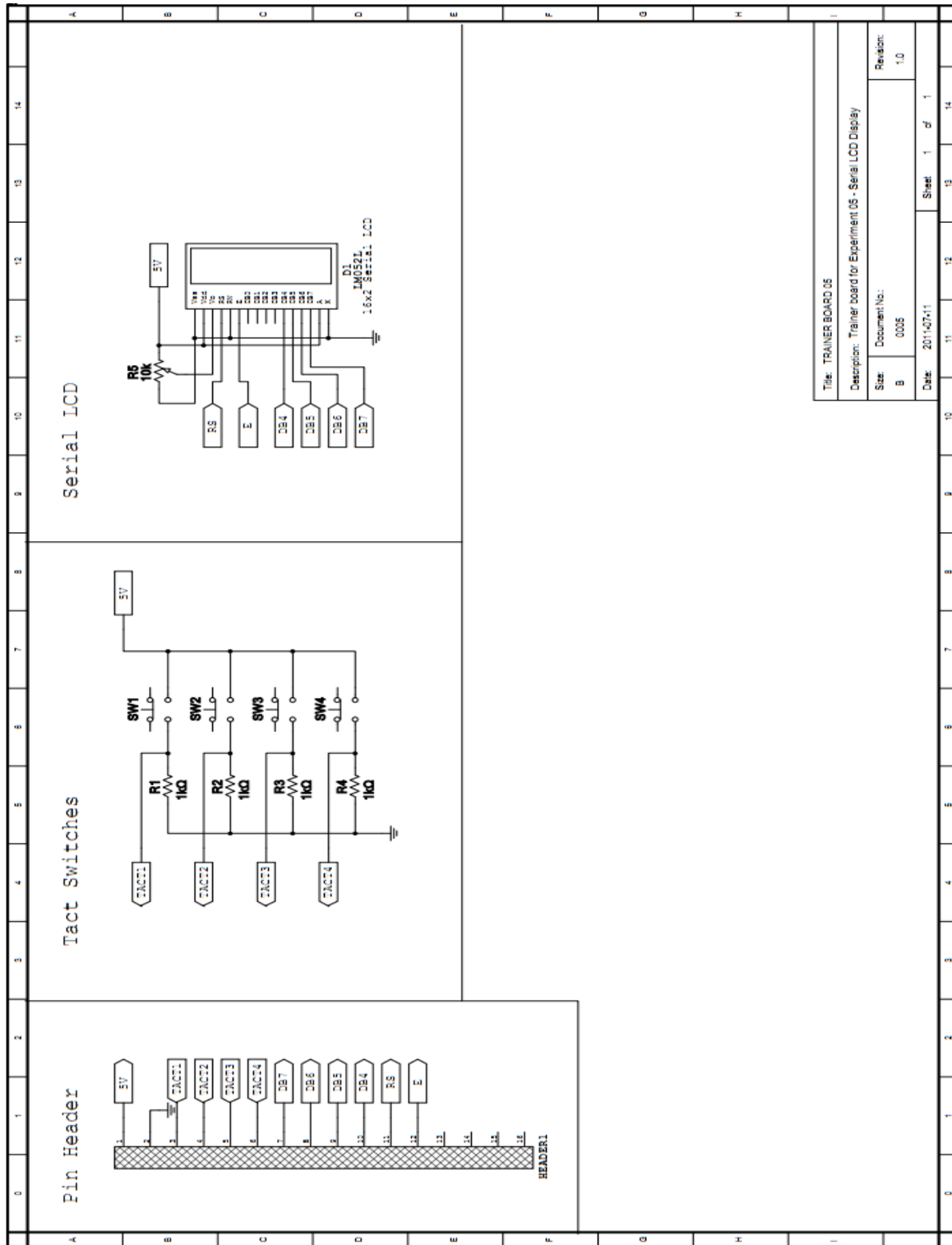


Figure 3.7. Trainer Board 05

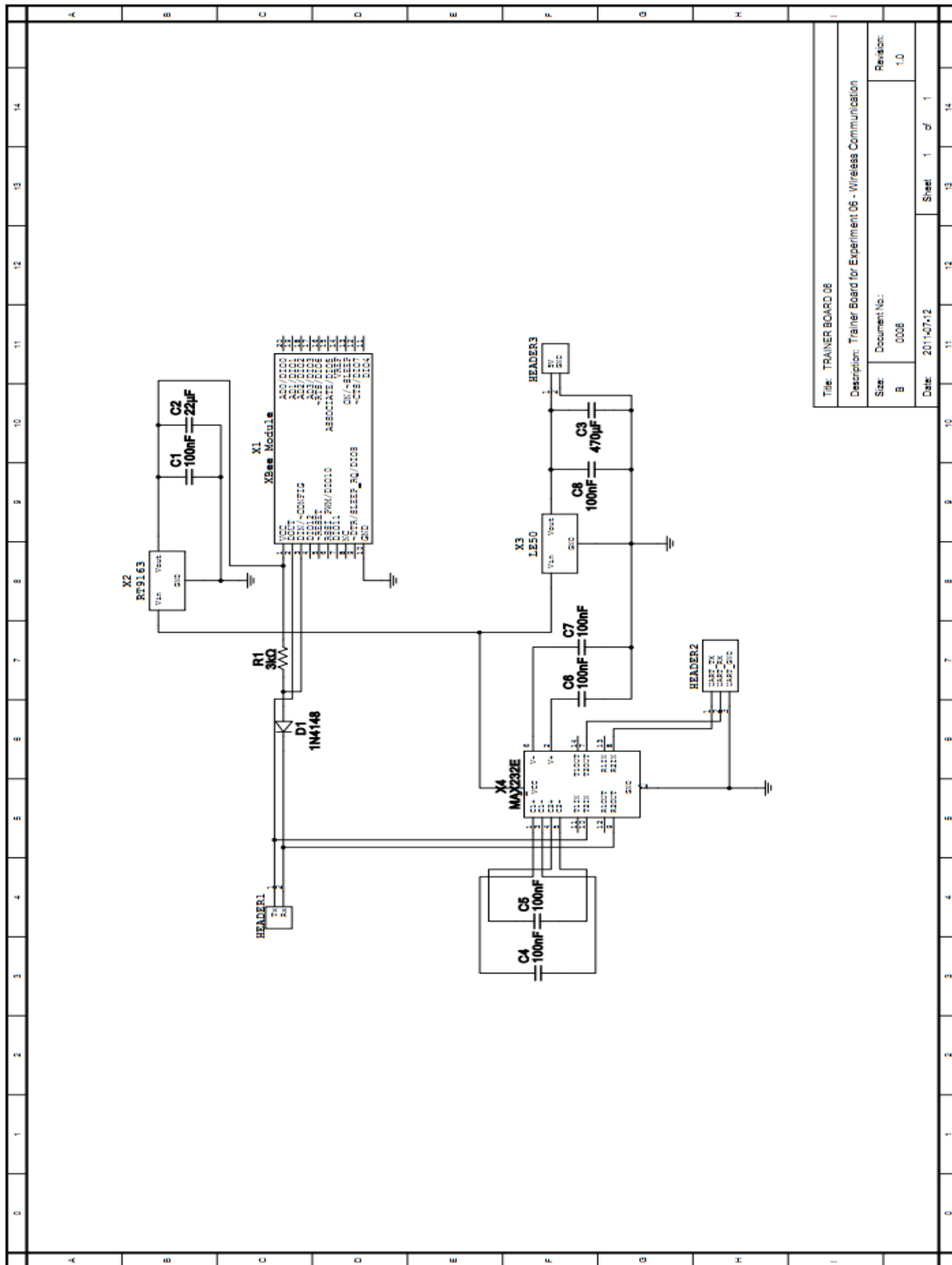


Figure 3.8. Trainer Board 06

Discussion of Components Used

1) Trainer Board 01

5.1) Pin Headers – These are used for making external connections from the trainer board to the Gizduino microcontroller kit. The pins of each pin header are associated with specific signals, in which the direction of the flow of each signal being indicated by arrow-pointed boxes containing the signal name as shown in the schematic diagram.

5.2) Tact Switches – These are four normally-open switches, with each outputting a high signal when pressed and a low signal when not pressed.

5.3) DIP Switch – 8 on-off switches in a DIP package. Each switch outputs a high signal in ON state and a low signal in OFF state

5.4) LEDs (light-emitting diode) – These are 4 LEDs with each turning on when provided with a high signal; otherwise each are turned off.

5.5) 7-Segment Display – This displays the decimal equivalent of an input 4-digit binary number ranging from 0000 – 1001 with the help of a 7-segment decoder (74LS47N).

2) Trainer Board 02

2.1) Pin Header – refer to 1.1.

2.2) LED Matrix – This is an 8x8 common cathode LED matrix. It utilizes two 8-bit shift registers (74HC595) and an 8-channel Darlington sink driver (ULN2803) to drive the rows and columns of the LED matrix.

2.3) Keypad – It is a 4x3 numeric keypad which uses an 8-bit shift register (74HC595) for scanning the rows of the keypad.

2.4) Buzzer – It is a 5 V rated buzzer which generates sound when the signal at the base of transistor Q1 (2N3904) receives a high signal.

3) Trainer Board 03

3.1) Pin Header – refer to 1.1.

3.2) Potentiometers – These are three 10k Ω potentiometers, with each outputting a variable voltage from 0 to 5 V as the shaft of the potentiometer is turned clockwise or counterclockwise.

3.3) RGB LEDs – These are two RGB (Red-Green-Blue) LEDs which could either produce the color red, green, blue, or a combination of those by bringing one and/or more cathode pins of the RGB LEDs to 0 V.

3.4) LDR – It stands for Light-Dependent Resistor; outputs a varying voltage from 0 to 5 V depending on the amount of light that it receives.

4) Trainer Board 04

4.1) Pin Header – refer to 1.1.

4.2) Power Supply – This provides the trainer board with regulated 5 V DC supply (different from the 5 V supplied by the Gizduino to the trainer board) which is used mainly to power up the different motors used in the experiment.

4.3) Tact Switches – refer to 1.2.

4.4)RG LED – These could either produce the color red, green or a combination of those two by sending a high signal to one and/or more anode pins of the RG LED.

4.5)LDR – refer to 3.4.

4.6)DC Motor Driver – It utilizes a push-pull four channel driver IC (L293D) which enables a DC motor to run in either forward or reverse direction with variable speed.

4.7)Stepper Motor Driver – This utilizes a Darlington array IC (ULN2003) in driving the stepper motor.

5) Trainer Board 05

5.1)Pin Header – refer to 1.1.

5.2)Tact Switches – refer to 1.2.

5.3)Serial LCD – This is a 16-row by 2-column alphanumeric liquid-crystal display; could display letters, numbers, and user-defined characters through serial communication of data between the Gizduino and the LCD.

6) Trainer Board 06

6.1)HEADER1 – This is a 2-pin male header with signals Tx (transmit) and Rx (receive) connected externally to the Rx and Tx pins of the Gizduino respectively. This header enables serial communication between the Gizduino and the XBee.

6.2)HEADER2 – This is a 3-pin male header with signals UART_TX, UART_RX and UART_GND connected externally to the pins Transmit, Receive, and

Ground of a serial port DB-9 connector respectively. This header enables serial communication between the PC and XBee.

6.3) HEADER3 – This is a 2-pin male header connected externally to a 5 V supply. This header supplies power to the trainer board.

6.4) XBee Module – This operates within the Zigbee protocol and supports the needs of low-cost and low-power wireless sensor networks. This module is responsible for wireless communication of data to other XBee modules.

6.5) RT9163-33PG – a voltage regulator which supplies 3.3V to the XBee module.

6.6) LE50 – It is a 5 V regulator specially used in low noise and low power systems.

6.7) MAX232E – It is a driver/receiver IC which translates EIA-232 inputs from the PC to TTL/CMOS voltage levels which is required to program the XBee module.

Chapter 4

TESTING, PRESENTATION, AND INTERPRETATION OF DATA

Test Methodology

The finalized laboratory experiments and trainer boards were administered to at least 8 pairs of students who have background on electronics, circuits and C++ programming but have little or no background on programming microcontrollers. Every time a pair finished an experiment they were both required to answer a laboratory assessment form to evaluate their experience on the performed experiment (refer to Appendix B for the laboratory assessment form).

The data gathered from the answered laboratory assessment forms were separated into three sections: the laboratory experiment section, trainer board section, and the self-assessment section. Data for each section were also organized into six tables which corresponds to the six experiments that have been performed and evaluated. In each table, were the statements that have been evaluated, the frequencies of respondents who answered under a particular scale, the total frequency, the weighted means, and the interpretations. The weighted mean was calculated using Equation 4.1:

$$x = \frac{f_1x_1 + f_2x_2 + f_3x_3 + f_4x_4 + f_5x_5}{f_T} \quad (4.1)$$

where:

\bar{X} = Weighted mean

$f_1 - f_5$ = Frequencies of respondents who answered under the scale items

Strongly Agree, Agree, Neither, Disagree, and Strongly Disagree respectively

$x_1 - x_5$ = numerical weights for the scale items Strongly Agree, Agree, Neither, Disagree, and Strongly Disagree; equivalent to the values 5, 4, 3, 2 and 1 respectively

f_T = Total frequency

The interpretation for the calculated weighted mean is shown in Table 4.1:

Range	Interpretation
1.00 – 1.80	Strongly Disagree
1.81 – 2.60	Disagree
2.61 – 3.40	Neither
3.41 – 4.20	Agree
4.21 – 5.00	Strongly Agree

Table 4.1. Weighted mean interpretation table.

Each section also included a summary table which summarized the data generated from the previous tables that has been produced. This table listed the statements that have been evaluated, their corresponding overall weighted mean and their corresponding interpretation. The overall weighted mean was computed using Equation 4.2:

$$X_{OWMN} = \frac{X_{N1} + X_{N2} + X_{N3} + X_{N4} + X_{N5} + X_{N6}}{6} \quad (4.2)$$

where:

X_{OWMN} = Overall weighted mean of statement N

$X_{N1} - X_{N6}$ = Weighted means of statement N at experiments 1 to 6

N = Number of statements

Laboratory Experiment Evaluation

The statements for the laboratory experiment section of the laboratory assessment form are listed in Table 4.2:

	Statements
1	The discussion is clear in introducing the basic concepts necessary to perform the experiment.
2	The schematic diagram is readable and understandable.
3	The procedures are clear and easy to follow.
4	The activities are easy to perform.
5	The objectives of the experiment have been met.

Table 4.2. Statements for the laboratory experiment section.

The following are the generated tables and charts from the collected data in the laboratory experiment section:

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	16	2	0	0	0	18	4.89	Strongly Agree
2	8	9	1	0	0	18	4.39	Strongly Agree
3	8	8	2	0	0	18	4.33	Strongly Agree
4	2	1	12	3	0	18	3.11	Neither
5	10	8	0	0	0	18	4.56	Strongly Agree

Table 4.3. Laboratory experiment evaluation table for experiment 1.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	14	4	0	0	0	18	4.78	Strongly Agree
2	10	8	0	0	0	18	4.56	Strongly Agree
3	11	6	0	1	0	18	4.50	Strongly Agree
4	1	4	10	3	0	18	3.17	Neither
5	14	4	0	0	0	18	4.78	Strongly Agree

Table 4.4. Laboratory experiment evaluation table for experiment 2.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	15	3	0	0	0	18	4.83	Strongly Agree
2	11	7	0	0	0	18	4.61	Strongly Agree
3	6	11	0	1	0	18	4.22	Strongly Agree
4	1	2	14	1	0	18	3.17	Neither
5	12	6	0	0	0	18	4.67	Strongly Agree

Table 4.5. Laboratory experiment evaluation table for experiment 3.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	14	4	0	0	0	18	4.78	Strongly Agree
2	10	8	0	0	0	18	4.56	Strongly Agree
3	7	8	2	1	0	18	4.17	Agree
4	2	2	11	3	0	18	3.17	Neither
5	9	9	0	0	0	18	4.50	Strongly Agree

Table 4.6. Laboratory experiment evaluation table for experiment 4.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	13	5	0	0	0	18	4.72	Strongly Agree
2	6	12	0	0	0	18	4.33	Strongly Agree
3	6	10	2	0	0	18	4.22	Strongly Agree
4	2	1	10	5	0	18	3.00	Neither
5	11	7	0	0	0	18	4.61	Strongly Agree

Table 4.7. Laboratory experiment evaluation table for experiment 5.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	9	8	1	0	0	18	4.44	Strongly Agree
2	9	8	1	0	0	18	4.44	Strongly Agree
3	7	8	3	0	0	18	4.22	Strongly Agree
4	0	1	15	2	0	18	2.94	Neither
5	6	12	0	0	0	18	4.33	Strongly Agree

Table 4.8. Laboratory experiment evaluation table for experiment 6.

Statement	Overall Weighted Mean	Interpretation
1	4.74	Strongly Agree
2	4.48	Strongly Agree
3	4.28	Strongly Agree
4	3.09	Neither
5	4.57	Strongly Agree

Table 4.9. Laboratory experiment evaluation summary table.

From Table 4.9, the interpretation for the overall weighted means for statements 1, 2, 3 and 5 shows that most of the respondents strongly agree that the discussion in the experiments is clear, the schematic diagram is understandable and readable, the procedures are clear, and the objectives of the performed experiment have been met. However, the interpretation for the overall weighted mean for statement 4 shows that respondents neither agree nor disagree that the activities are easy to perform.

Trainer Board Evaluation

The statements for the trainer board section of the laboratory assessment form are listed in Table 4.10:

	Statements
1	The trainer board is suited to the experiment to be performed.
2	The trainer board is helpful in understanding and performing the experiment.

Table 4.10. Statements for the trainer board section.

The following are the generated tables and charts from the collected data in the trainer board section:

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	13	5	0	0	0	18	4.72	Strongly Agree
2	15	3	0	0	0	18	4.83	Strongly Agree

Table 4.11. Trainer board evaluation table for experiment 1.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	15	3	0	0	0	18	4.83	Strongly Agree
2	12	6	0	0	0	18	4.67	Strongly Agree

Table 4.12. Trainer board evaluation table for experiment 2.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	15	3	0	0	0	18	4.83	Strongly Agree
2	14	4	0	0	0	18	4.78	Strongly Agree

Table 4.13. Trainer board evaluation table for experiment 3.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	13	5	0	0	0	18	4.72	Strongly Agree
2	11	7	0	0	0	18	4.61	Strongly Agree

Table 4.14. Trainer board evaluation table for experiment 4.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	14	4	0	0	0	18	4.78	Strongly Agree
2	11	7	0	0	0	18	4.61	Strongly Agree

Table 4.15. Trainer board evaluation table for experiment 5.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	8	10	0	0	0	18	4.44	Strongly Agree
2	8	10	0	0	0	18	4.44	Strongly Agree

Table 4.16. Trainer board evaluation table for experiment 6.

Statement	Overall Weighted Mean	Interpretation
1	4.72	Strongly Agree
2	4.66	Strongly Agree

Table 4.17. Trainer board evaluation analysis table.

From Table 4.17, the interpretation for the overall weighted mean for statement 1 shows that most of the respondents strongly agree that each trainer board is suitable to the experiment in which it is supposed to be used. Similarly, for the overall weighted mean for statement 2, it shows that most of the respondents strongly agree that the trainer boards are helpful in understanding and performing the experiments.

Self-Assessment Evaluation

The statements for the self-assessment evaluation section of the laboratory assessment form are listed in Table 4.18:

	Statements
1	I am confident that I could apply the concepts I learned in the experiment using an Arduino board.
2	I am confident that the knowledge I obtained will be helpful in learning and using other kinds of microcontrollers.

Table 4.18. Statements for the self-assessment section.

The following are the generated tables and charts from the collected data in the self-assessment section:

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	13	5	0	0	0	18	4.72	Strongly Agree
2	11	7	0	0	0	18	4.61	Strongly Agree

Table 4.19. Self-assessment evaluation table for experiment 1.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	10	8	0	0	0	18	4.56	Strongly Agree
2	11	7	0	0	0	18	4.61	Strongly Agree

Table 4.20. Self-assessment evaluation table for experiment 2.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	14	4	0	0	0	18	4.78	Strongly Agree
2	14	4	0	0	0	18	4.78	Strongly Agree

Table 4.21. Self-assessment evaluation table for experiment 3.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	10	8	0	0	0	18	4.56	Strongly Agree
2	10	8	0	0	0	18	4.56	Strongly Agree

Table 4.22. Self-assessment evaluation table for experiment 4.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	12	6	0	0	0	18	4.67	Strongly Agree
2	12	6	0	0	0	18	4.67	Strongly Agree

Table 4.23. Self-assessment evaluation table for experiment 5.

Statement	Frequency						Weighted Mean	Interpretation
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree	Total Frequency		
1	8	5	0	0	0	13	4.62	Strongly Agree
2	9	4	0	0	0	13	4.69	Strongly Agree

Table 4.24. Self-assessment evaluation table for experiment 6.

Statement	Overall Weighted Mean	Interpretation
1	4.65	Strongly Agree
2	4.65	Strongly Agree

Table 4.25. Self-assessment evaluation summary table.

From Table 4.25, the interpretation for the overall weighted means of statements 1 and 2 shows that most of the respondents strongly agree that they are confident in applying the concepts that they have learned from the experiments using an Arduino board or any other types of microcontrollers.

Chapter 5

CONCLUSION AND RECOMMENDATION

Conclusion

The evaluation results on the respondents on the created laboratory experiments show that the created laboratory experiments helped the students in learning how to program and interface microcontrollers. Similarly, results on the evaluation of the respondents on the created trainer boards show that it helped the students learn the concepts presented in the laboratory experiments. The results on the assessment of the respondents' confidence on applying the knowledge that they have learned in using Arduino or other kinds of microcontrollers further verifies that the created laboratory experiments and trainer boards are helpful in teaching the students programming and interfacing microcontrollers.

From the results gathered, the researchers conclude that the design of laboratory experiments and trainer boards for Arduino using Gizduino microcontroller kit were successfully developed. A dedicated microcontroller course that helps students learn programming and interfacing microcontrollers was successfully developed.

Recommendation

There are other improvements on the design that can be done. First, guide questions for Experiments 2 to 6 may be provided. The guide questions can help the students to understand more the concepts discussed in the experiments.

A base platform could be created so that the Gizduino and the trainer boards are stable when used. It is more comfortable to work if the boards that the students are working on are fixed in place.

. Lastly, supplementary experiments may be created to increase the proficiency of the students in using microcontrollers. Topics on two-wire interfacing, serial peripheral interfacing, Ethernet interfacing and many more can be included.

REFERENCES

Allaoua B., et.al.(2009). Neuro-Fuzzy DC Motor Speed Control Using Particle Swarm Optimization

Arduino < <http://arduino.cc>>

Durfee W., P. Li and D. Waletzko (2005). At-Home System and Controls Laboratories. *Proc. 2005 ASEE Annual Conference & Exposition*

e-Gizmo, Mechatronics Central <e-gizmo.com>

Garrels, M.(2008). Introduction to Linux

Hamad, M., et. Al. (2006). A PIC-Based Microcontroller Design Laboratory. *The 6th International Workshop on System on Chip for Real Time Applications 2006 IEEE International Conference*

Hao Yongsheng, Han Lujie and Wang Guanglong (2009). Design of virtual maintenance training system based on circuit component model. *Power Electronics and Motion Control Conference, 2009. IPEMC '09. IEEE 6th International*, pp.2634-2637,

Ibrahim and Dogan (2006). PIC BASIC Projects 30 Projects Using PIC BASIC and PIC BASIC PRO. *Amsterdam: Elsevier, 2006. Print.*

Linsangan, Noel B. (2009). Logic Circuits and Switching Theory Laboratory Manual 4th Edition

McComb, G. (2001). The Robot Builder's Bonanza

Negru, S. (2010). A conceptual architecture of an arduino-based social-emotional interactive system. *Intelligent Computer Communication and Processing (ICCP), 2010 IEEE International Conference pp.93-98*

Ontimare, Cyrel O. (2008). *Microprocessor Laboratory Manual*, January 2008

Peng, Jian (2009). Revision of a Microcontroller Course for a Mixed Student Body. *In 39th ASEE/IEEE Frontiers in Education Conference San Antonio, TX, 2009.*

Sarik, J. and I. Kymissis (2010). Lab kits using the Arduino prototyping platform. *IEEE Frontiers in Education Conference (FIE), Oct. 2010*

Wikipedia, the Free Encyclopedia. Web. 22 Aug. 2010. <<http://en.wikipedia.org/>>.

Zhang Gang and Liu Shuguang (2010). Study on electrical switching device junction temperature monitoring system based on zigbee technology. *Computer Application and System Modeling (ICCASM), International Conference, October 2010*

APPENDICES

APPENDIX A

Arduino Laboratory Manual

Experiment No. 1
BASIC INPUT/OUTPUT

Objectives

At the end of the experiment the student should be able to:

1. Familiarize with Arduino and the Gizduino microcontroller board.
2. Familiarize with the Arduino development environment.
3. Read and understand an Arduino program.
4. Program in Arduino IDE and upload it on an Arduino board.
5. Understand the basic input and output operations in Arduino.
6. Use a trainer board by reading and understanding its schematic diagram.
7. Interface DIP switches, tact switches, LEDs and 7-segment displays on an Arduino board.

Equipment and Materials

- 1 pc Gizduino
- 1 pc Trainer Board 01
- 1 pc USB Standard A-B Cable
- 1 set Connecting Wires
- 1 set Desktop computer

Discussion

Arduino

Arduino is an open-source single-board microcontroller designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open hardware design for the Arduino board with an Atmel AVR processor and on-board I/O support. The software consists of a standard programming language compiler and the boot loader that runs on the board.

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (similar to C++ with some simplifications and modifications) and the Arduino development environment. Arduino projects can be standalone or they can communicate with software running on a computer.

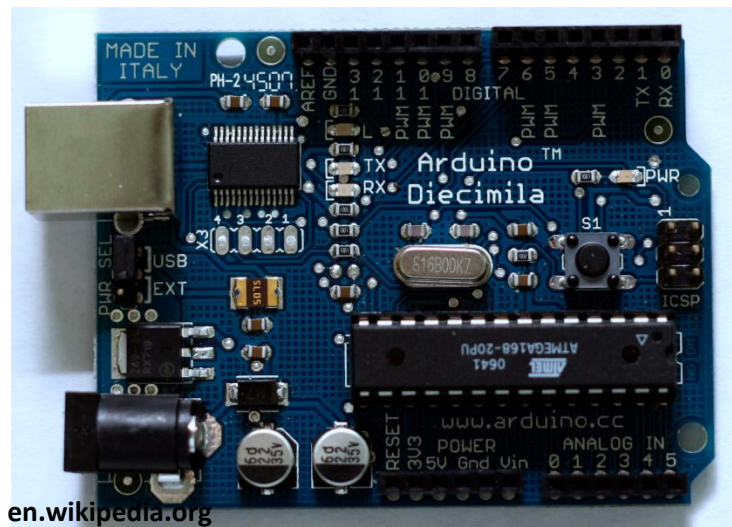


Figure 1.1. An Arduino board.

Gizduino

Gizduino is based on the popular Arduino board which is the Arduino Diecimilia. It is an open source computing platform based on a simple input/output (I/O) board and the use of standard programming language. *Gizduino* is programmed using the Arduino development environment, which can be downloaded free from the Arduino website (<http://arduino.cc>).

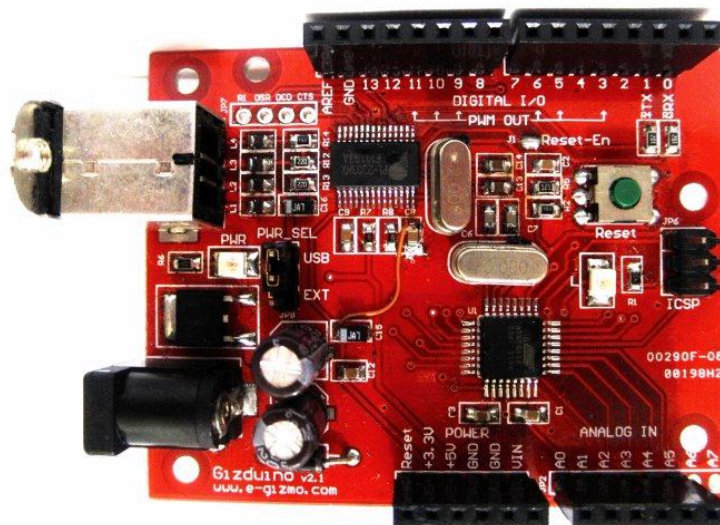


Figure 1.2. Gizduino.

Specifications:

Microcontroller	ATmega168
Operating Voltage	5 V
Recommended Input Voltage	7-12 V
Input Voltage Limits	6-20 V
Digital I/O Pins	14 (6 provides PWM output)
Analog Input Pins	6
Flash Memory	16 KB (2 KB used by bootloader)

Table 1.1. Gizduino Specifications.

Memory – Gizduino uses an ATmega168 microcontroller which has 16KB flash memory, 2KB of which is used for the bootloader. It has 1KB SRAM and 512 bytes of EEPROM.

Power – Gizduino can be powered by the USB connection or an external power supply. The external power supply can be an AC-DC adapter connected to the power jack or a battery with the positive end connected to VIN power pin and negative pin to GND pin.

Digital I/O Pins – There are 14 digital input/output pins in the Gizduino. Six of which are also analog output pins: pins 3, 5, 6, 9, 10, and 11. Pin 13 has a built-in light-emitting diode (LED) in the board.

Analog Input Pins – Used for reading external components with varying voltage reading such as sensors.

Arduino Development Environment

The Arduino development environment contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions, and a series of menus (see Figure 1.3). It connects to the Arduino hardware to upload programs and communicate with them.

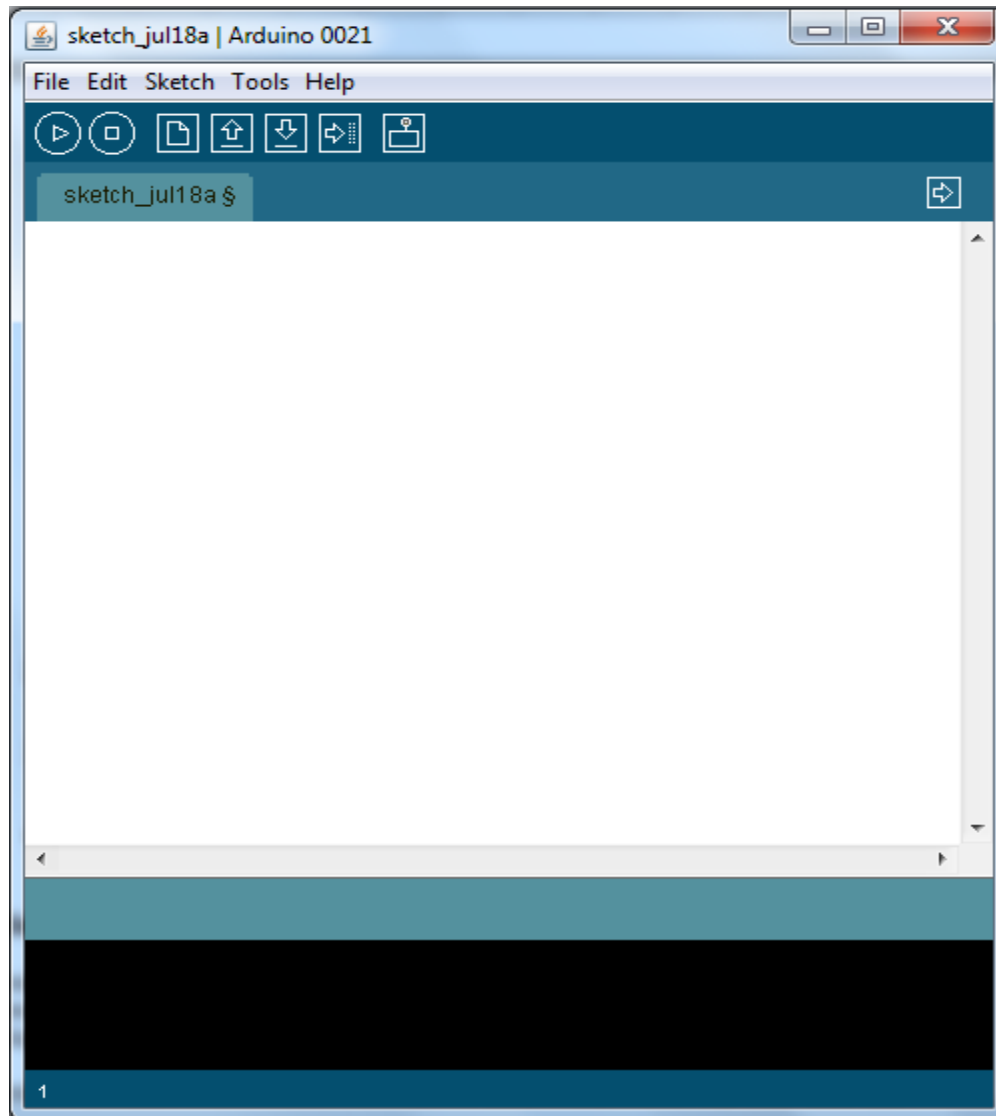





Figure 1.3. Arduino Development Environment.

Software written using Arduino are called *sketches*. These sketches are written in the text editor. It has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino environment including complete error messages and other information. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor:

-  *Verify/Compile*
Checks your code for errors.
-  *Stop*
Stops the serial monitor, or unhighlight other buttons.
-  *New*
Creates a new sketch.



Open

Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window.

Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the **File > Sketchbook** menu instead.



Save

Saves your sketch.



Upload to I/O Board

Compiles your code and uploads it to the Arduino I/O board.



Serial Monitor

Opens the serial monitor.

Additional commands are found within the five menus: File, Edit, Sketch, Tools, and Help. The menus are context sensitive which means only those items relevant to the work currently being carried out are available.

Sketchbook

The Arduino environment includes the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the **File > Sketchbook** menu or from the **Open** button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the Preferences dialog.

Uploading Sketches

Before uploading your sketch, you need to select the correct items from the **Tools > Board** and **Tools > Serial Port** menus. On Windows, to find out which COM port is used by your Arduino board, look for the USB serial device in the Ports (COM & LPT) section of the Windows Device Manager.

Once you've selected the correct serial port and board, press the upload button in the toolbar or select the **Upload to I/O Board** item from the File menu. Current Arduino boards will reset automatically and begin the upload. With older boards that lack auto-reset, you'll need to press the reset button on the board just before starting the upload. On most boards, you'll see the Rx and Tx LEDs blink as the sketch is uploaded. The Arduino environment will display a message when the upload is complete, or show an error.

When you upload a sketch, you're using the Arduino bootloader, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

Arduino Programming Basics

Sketch

A *sketch* is the name that Arduino uses for a program. It is the unit of code that is uploaded to and run on an Arduino board.

Comments

The first few lines of the “Blink” sketch below are a *comment*:

```
/*
 * Blink
 *
 * The basic Arduino example. Turns on an LED on for one second,
 * then off for one second, and so on... We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */
```

Everything between the `/*` and `*/` is ignored by the Arduino when it runs the sketch (the `*` at the start of each line is only there to make the comment look pretty, and isn't required). It's there for people reading the code: to explain what the program does, how it works, or why it's written the way it is. It's a good practice to comment your sketches, and to keep the comments up-to-date when you modify the code. This helps other people to learn from or modify your code.

There's another style for short, single-line comments. These start with `//` and continue to the end of the line. For example, in the line:

```
int ledPin = 13; // LED connected to digital pin 13
```

The message "LED connected to digital pin 13" is a comment.

Variables

A *variable* is a place for storing a piece of data. It has a name, a type, and a value. For example, the line from the Blink sketch declares a variable with the name `ledPin`, the type `int`, and an initial value of 13. It's being used to indicate which Arduino pin the LED is connected to. Every time the name `ledPin` appears in the code, its value will be retrieved. In this case, the person writing the program could have chosen not to bother creating the `ledPin` variable and instead have simply written 13 everywhere they needed to specify a pin number. The advantage of using a variable is that it's easier to move the

LED to a different pin: you only need to edit the one line that assigns the initial value to the variable.

Often, however, the value of a variable will change while the sketch runs. Most of the time variables are used to store the value read from an input.

Functions

A *function* (otherwise known as a *procedure* or *sub-routine*) is a named piece of code that can be used from elsewhere in a sketch. For example, here's the definition of the `setup()` function from the Blink example:

```
void setup()
{
  pinMode(ledPin, OUTPUT); // sets the digital pin as output
}
```

The first line provides information about the function, like its name, "setup". The text before and after the name specify its return type and parameters. The code between the { and } is called the *body* of the function: what the function does.

You can *call* a function that's already been defined (either in your sketch or as part of the Arduino language). For example, the line `pinMode(ledPin, OUTPUT);` calls the `pinMode()` function, passing it two *parameters*: `ledPin` and `OUTPUT`. These parameters are used by the `pinMode()` function to decide which pin and mode to set. You will learn more functions in the succeeding experiments.

setup() and loop()

There are two special functions that are a part of every Arduino sketch: `setup()` and `loop()`. The `setup()` is called once, when the sketch starts. It's a good place to do setup tasks like setting pin modes or initializing libraries. The `loop()` function is called over and over and is heart of most sketches. You need to include both functions in your sketch, even if you don't need them for anything.

pinMode(), digitalWrite(), and delay()

The `pinMode()` function configures a pin as either an input or an output. To use it, you pass it the number of the pin to configure and the constant `INPUT` or `OUTPUT`. When configured as an input, a pin can detect the state of a sensor like a pushbutton; as an output, it can drive an actuator like an LED. Its usage is shown below:

```
pinMode(13, OUTPUT); //configure pin 13 as an output pin
```

The `digitalWrite()` function outputs a value on a pin. For example, the line

```
digitalWrite(7, HIGH);
```

sets pin 7 to HIGH, or 5 volts. Writing a LOW to this pin connects it to ground, or 0 volts.

The `delay()` causes the Arduino to wait for the specified number of milliseconds before continuing on to the next line. There are 1000 milliseconds in a second, so the line

```
delay(1000);
```

creates a delay of one second.

Blink a LED Without Using `delay()`

Sometimes you need to do two things at once. For example you might want to blink a LED (or some other time-sensitive function) while reading a button press or other input. In this case, you can't use `delay()`, or you'd stop everything else the program while the LED blinked. The program might miss the button press if it happens during the `delay()`. This could be done by keeping track of the last time the Arduino turned the LED on or off using the `millis()` function. Then, each time through `loop()`, an `if` statement checks if a long enough interval has passed. If it has, it toggles the LED on or off.

Button State Change

Once you've got a pushbutton working, you often want to do some action based on how many times the button is pushed. To do this, you need to know when the button changes state from off to on (or on to off in some circumstances). This is called *state change detection* or *edge detection*.

Edge detection compares the button's state to its state the last time through the main loop. If the current button state is different from the last button state and the current button state is high then the button changed from off to on. Every time the button state changes you could trigger some special action, like toggling a LED on or off.

Trainer Board Schematic Diagram

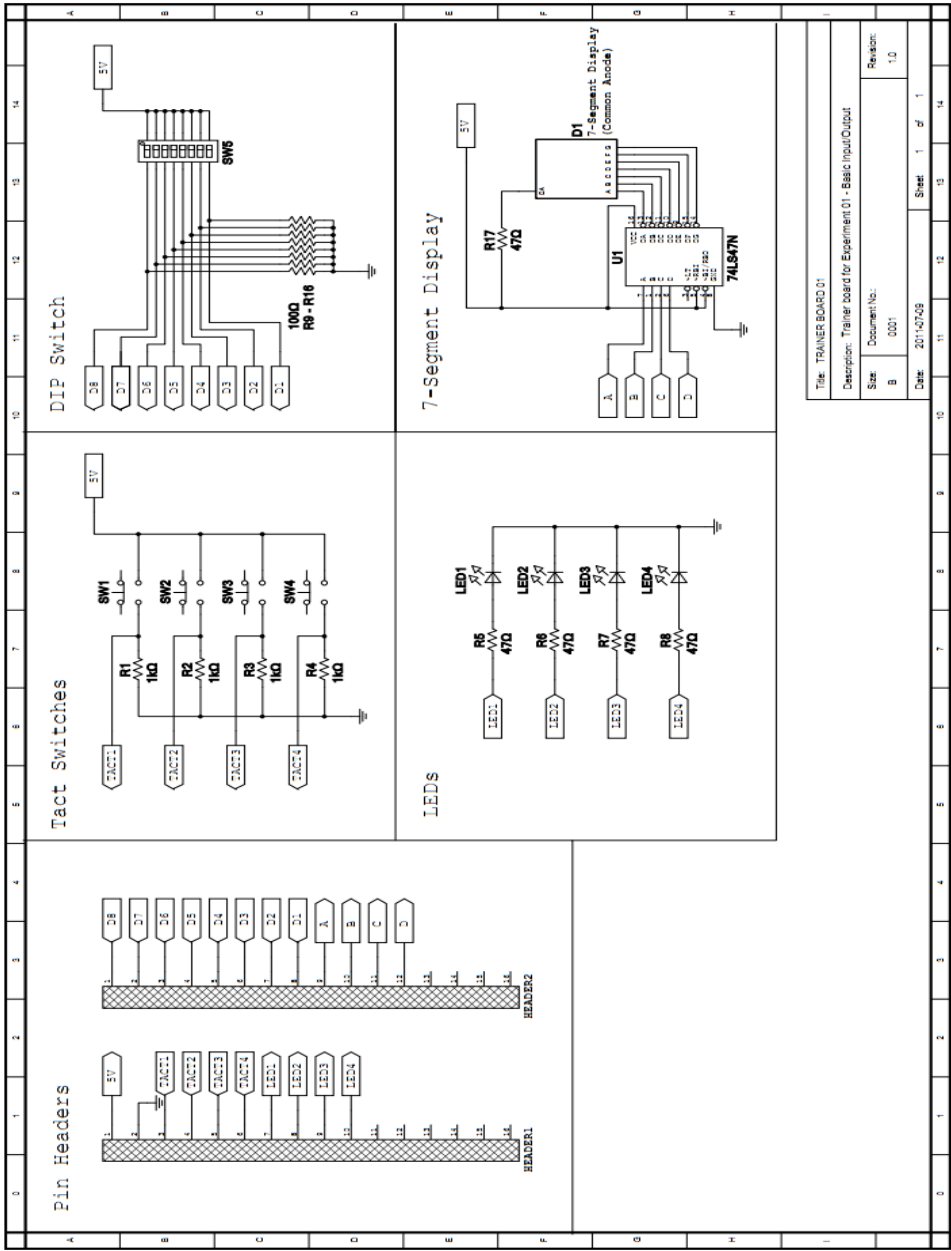


Figure 1.4. Trainer Board 01

Procedures

Part I. Blinking LED


A. Blink a LED using the delay() function

1. Open Arduino IDE. Encode the sketch below:


```
//set blink delay to 1000 ms (1 s):
int interval = 1000;

void setup()
{
    // initialize digital pin 13 as output.
    pinMode(13, OUTPUT);
}

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

2. Click the  (Verify) button, found at the toolbar, to check for errors. If there are errors, review and edit the part of the code where the error occurred. Repeat this step until the verification is successful.
3. Once the program has been successfully verified, connect the Gizduino to the computer using a USB standard A-B cable. Specify the COM port where the Gizduino is connected in **Tools > Serial Port > [COM port used]**.

TIP: You could find which COM port the Gizduino is connected by expanding the *Ports (COM & LPT)* item in the Windows Device Manager. The Gizduino will be connected to *Prolific USB-to-Serial Comm Port (COM#)*.

4. Specify the board to use by checking the appropriate Arduino board under **Tools > Board**. Since the Gizduino is a clone of Arduino Diecimila, check **Arduino Diecimila, Duemilanove, or Nano w/ ATmega168**.
5. Click the  (Upload) button. This will now compile and burn the program to the Gizduino.

TIP: You could actually skip Step 2 since the Upload button automatically verifies your program before uploading it to your Arduino board.

6. Refer to Figure 1.4. Determine which pins on the pin headers of the trainer board are connected to the LEDs. Connect one of these pins to pin 13 of Gizduino.

7. Refer again to Figure 1.4. Determine the voltage supply pins of the trainer board and connect it to +5V and GND pins of the Gizduino.

NOTE: Be careful not to interchange the voltage supply connections of the trainer board to the Gizduino.

8. Observe the LED that you connected to the Gizduino.

Guide Questions I.A

1. Describe the behavior of the LED that you connected to the Gizduino. What functions cause the LED to exhibit this kind of behavior?

2. Disconnect the trainer board from the Gizduino then observe the onboard LED on the Gizduino. What can you conclude from this?

B. Blink a LED without using the delay() function

1. Encode the sketch below and upload it to Gizduino. Refer to Part A on how to verify and upload a sketch to an Arduino board.

```
//assign a variable to a pin number
int ledPin = 13;

//variable used to set the LED
int ledState = LOW;

//will store last time LED was updated
long previousMillis = 0;

//interval at which to blink (milliseconds)
long interval = 1000;

void setup()
{
    //set ledPin as output
    pinMode(ledPin, OUTPUT);
}
```

```

}

void loop()
{
    unsigned long currentMillis = millis();

    /*check to see if it's time to blink the LED; that is, if
    the difference between the current time and last time
    you blinked the LED is bigger than the interval at which
    you want to blink the LED. */

    if(currentMillis - previousMillis > interval)
    {
        //save the last time you blinked the LED
        previousMillis = currentMillis;

        //if the LED is off turn it on and vice-versa:
        if (ledState == LOW)
            ledState = HIGH;
        else
            ledState = LOW;

        //set the LED with the ledState of the variable:
        digitalWrite(ledPin, ledState);
    }
}

```

2. Refer to Figure 1.4. Connect a LED on the trainer board to pin 13 of the Gizduino.
3. Connect the supply pins of the trainer board to +5V and GND pins of the Gizduino.
4. Observe the LED that you connected to the Gizduino.

Guide Questions I.B

1. What does the `millis()` function do?

2. How does the blinking LED sketch in this part differ in the blinking LED sketch in Part A?

Part II. Digital Input/Output

A. Read input from switches and output to LEDs

1. Encode the sketch below and upload it to the Arduino board:

```
//assign variables to pin numbers
int buttonPin1 = 0;
int buttonPin2 = 1;
int buttonPin3 = 2;
int buttonPin4 = 3;

int ledPin1 = 13;
int ledPin2 = 12;
int ledPin3 = 11;
int ledPin4 = 10;

// variable for reading the pushbutton status
int buttonState = 0;

void setup()
{
    //initialize the LED pins as outputs
    pinMode(ledPin1, OUTPUT);
    pinMode(ledPin2, OUTPUT);
    pinMode(ledPin3, OUTPUT);
    pinMode(ledPin4, OUTPUT);
    //initialize the pushbutton pins as inputs
    pinMode(buttonPin1, INPUT);
    pinMode(buttonPin2, INPUT);
    pinMode(buttonPin3, INPUT);
    pinMode(buttonPin4, INPUT);
}

void loop()
{
    // check if first tact switch is pressed:
    if (digitalRead(buttonPin1) == HIGH)
    {
        digitalWrite(ledPin1, HIGH); //turn LED1 on
    }
    else
    {
        digitalWrite(ledPin1, LOW); //turn LED1 off
    }
    // check if second tact switch is pressed:
    if (digitalRead(buttonPin2) == HIGH)
    {
        digitalWrite(ledPin2, HIGH); //turn LED2 on
    }
    else
```

```

{
    digitalWrite(ledPin2, LOW); //turn LED2 off
}
// check if third tact switch is pressed:
if (digitalRead(buttonPin3) == HIGH)
{
    digitalWrite(ledPin3, HIGH); //turn LED3 on
}
else
{
    digitalWrite(ledPin3, LOW); //turn LED3 off
}
//check if fourth tact switch is pressed:
if (digitalRead(buttonPin4) == HIGH)
{
    digitalWrite(ledPin4, HIGH); //turn LED4 on
}
else
{
    digitalWrite(ledPin4, LOW); //turn LED4 off
}
}

```

2. Connect pins 0-3 and 10-13 of the Gizduino to the tact switches and LEDs on the trainer board respectively.
3. Connect the voltage supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Now push the tact switches on the trainer board. Observe the LED outputs.

Guide Questions II.A

1. Describe the behavior of the LEDs as you push the tact switches.

2. What function detects the state of the tact switch when pressed?

B. Detect button state change

1. Encode the sketch below and upload it to the Arduino board:

```

//button and LED pins
int buttonPin = 2;
int ledPin = 13;

```

```

//current state of the button
int buttonState = LOW;
//previous state of the button
int lastButtonState = LOW;
//variable used to set the LED
int ledState = LOW;

void setup()
{
    //initialize I/O pins
    pinMode(buttonPin, INPUT);
    pinMode(ledPin, OUTPUT);

    //initially set the LED off
    digitalWrite(ledPin, LOW);
}
void loop()
{
    //read the current state of the pushbutton
    buttonState = digitalRead(buttonPin);

    //compare the buttonState to its previous state
    if (buttonState != lastButtonState)
    {
        //if the current state is HIGH then the button
        //went from off to on
        if (buttonState == HIGH)
        {
            //toggle the state of the LED on or off
            ledState = !ledState;
        }
    }

    //save the current state as the last state,
    //for next time through the loop
    lastButtonState = buttonState;
    //write the value of ledState to the LED
    digitalWrite(ledPin, ledState);
}

```

2. Connect pins 2 and 13 of the Gizduino to a tact switch and a LED on the trainer board respectively.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Now push the tact switch several times. Observe the behavior of the LED as the tact switch is pressed.

Guide Questions II.B

1. Describe the behavior of the LED as you push the tact switch. How does the behavior of the LED in this part differ from the behavior of the LEDs in Part A when the tact switches are pressed?

C. Read input from a DIP switch and output to a 7-segment display

1. Encode the sketch below and upload it to the Arduino board:

```
void setup()
{
    //pins 0-7 connect to the DIP switch
    pinMode(0, INPUT);
    pinMode(1, INPUT);
    pinMode(2, INPUT);
    pinMode(3, INPUT);
    pinMode(4, INPUT);
    pinMode(5, INPUT);
    pinMode(6, INPUT);
    pinMode(7, INPUT);

    //pins 8-11 connect to the 7-segment display driver
    pinMode(8, OUTPUT);
    pinMode(9, OUTPUT);
    pinMode(10, OUTPUT);
    pinMode(11, OUTPUT);
}

void loop()
{
    //check if D8 is on
    if (digitalRead(7) == HIGH)
    {
        digitalWrite(11,HIGH);
        digitalWrite(10,LOW);
        digitalWrite(9,LOW);
        digitalWrite(8,LOW);
    }

    //check if D7 is on
    else if (digitalRead(6) == HIGH)
    {
        digitalWrite(11,LOW);
        digitalWrite(10,HIGH);
        digitalWrite(9,HIGH);
        digitalWrite(8,HIGH);
    }
}
```



```

else if (digitalRead(5) == HIGH)
{
    digitalWrite(11,LOW);
    digitalWrite(10,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(8,LOW);
}
else if (digitalRead(4) == HIGH)
{
    digitalWrite(11,LOW);
    digitalWrite(10,HIGH);
    digitalWrite(9,LOW);
    digitalWrite(8,HIGH);
}
else if (digitalRead(3) == HIGH)
{
    digitalWrite(11,LOW);
    digitalWrite(10,HIGH);
    digitalWrite(9,LOW);
    digitalWrite(8,LOW);
}
else if (digitalRead(2) == HIGH)
{
    digitalWrite(11,LOW);
    digitalWrite(10,LOW);
    digitalWrite(9,HIGH);
    digitalWrite(8,HIGH);
}
else if (digitalRead(1) == HIGH)
{
    digitalWrite(11,LOW);
    digitalWrite(10,LOW);
    digitalWrite(9,HIGH);
    digitalWrite(8,LOW);
}
else if (digitalRead(0) == HIGH)
{
    digitalWrite(11,LOW);
    digitalWrite(10,LOW);
    digitalWrite(9,LOW);
    digitalWrite(8,HIGH);
}
else
{
    digitalWrite(11,LOW);
    digitalWrite(10,LOW);
    digitalWrite(9,LOW);
    digitalWrite(8,LOW);
}
}

```

2. Connect pins 0-7 and 8-11 of the Gizduino to pins D1-D8 and A-D of the trainer board respectively. Check Figure 1.4 for reference.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Now push the switches on the DIP switch. Observe the 7-segment display as the switches on the DIP switch are turned on and off.

Guide Questions II.C

1. Describe the behavior of the 7-segment display as switches in the DIP switch are pushed.

Review Questions

1. What is Arduino? How does it differ from other microcontroller boards?

2. What is Gizduino? What is its advantage compared to original Arduino boards?

3. What is a sketch? What are the settings that must be checked in order to successfully upload a sketch to an Arduino board?

4. List and briefly describe the toolbar buttons commonly used in the Arduino development environment.

5. What is the difference between the `void setup()` and `void loop()` sections in an Arduino sketch?.

6. What does the function `pinMode()` do? Briefly explain the parameters that are passed to this function.

7. Explain the advantage and disadvantage of not using the `delay()` function in a sketch which requires the use of time delays.

8. What does the function `digitalRead()` and `digitalWrite()` do? Briefly explain the parameters that are passed to this function.

9. How does the detection of the state of inputs read to Arduino work? When is state change detection performed?

10. Refer to the trainer board schematic diagram in Figure 1.4. How does the flow or direction of the signal depicted in this schematic diagram?

Activities

- Create a blinking LED program that uses 3 tact switches as inputs. Each of the three tact switches should set different blinking delays for the LED. The blinking delays should be .5 s, 1 s, and 2 s. (Note: Do not use the `delay()` function).
- Create a sketch that simulates the behavior of a negative-edge triggered JK flip-flop. Use tact switches for the inputs (J, K and Clock) and LEDs (Q and $\sim Q$) for the output. (Hint: Detect the state change of the Clock input from on to off).






J	K	Clock	Q	$\sim Q$
X	X		Q	$\sim Q$
0	0		Q	$\sim Q$
0	1		0	1
1	0		1	0
1	1		$\sim Q$	Q

Table 1.2. Negative-edge triggered JK flipflop Truth Table.

- c. Create
a sketch for a 4-bit adder using D8-D5 as the addend and D4-D1 as the augend (refer to Figure 1.4). Write the sum to 4 LEDs. (Note: In case of overflow, the answer should be 0000, i.e., all LEDs are off.)
- d. Refer to Part C in Part II. This time there will be two additional inputs, TACT1 and TACT2. Initially the 7-segment display should be 0. When TACT1 is pressed, the output should display the input read from the dip switches. When TACT2 is pressed, the output should be incremented. (Note: In case the output exceeds 9, it should cycle back to 0).

References

<http://arduino.cc/en/>
<http://en.wikipedia.org/wiki/Arduino>
<http://e-gizmo.com/wordpress/?p=369>
<http://www.arduino.cc/en/Guide/Environment>
<http://arduino.cc/en/Tutorial/Sketch>

Experiment No. 2
ADVANCED INPUT/OUTPUT

Objectives

At the end of the experiment the student should be able to:

1. Understand the principle of operation of a LED matrix and a numeric keypad.
2. Understand how to output different tones on a buzzer from an Arduino board.
3. Interface LED matrices, numeric keypads, and buzzers to an Arduino board.
4. Familiarize with Serial Monitor.

Equipment and Materials

- 1 pc Gizduino
- 1 pc Trainer Board 02
- 1 pc USB Standard A-B Cable
- 1 set Connecting Wires
- 1 set Desktop computer

Discussion

LED Matrix

A LED matrix is a simple form of dot matrix display. It consists of a 2-D matrix of LEDs with their cathodes joined in rows and their anodes joined in columns (or vice versa).

By controlling the flow of electricity through each row and column pair it is possible to control each LED individually. By scanning across rows or columns, quickly flashing the LEDs on and off, it is possible to create characters or pictures to display information to the user. The LED Matrix may be driven by certain ICs or by the use of microcontrollers to control the flow of electricity which then controls the rate of the flicker.

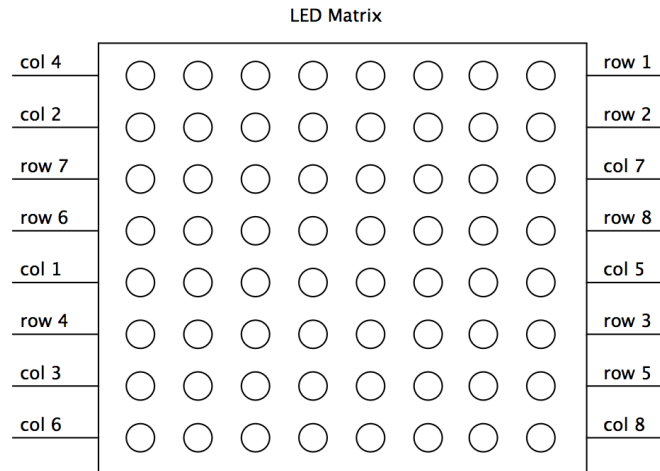


Figure 2.1. An 8x8 LED Matrix display with pin assignments

Keypad



Figure 2.2. Keypad

A keypad is a set of buttons arranged in a block or "pad" which usually bear digits, symbols and usually a complete set of alphabetical letters. Similar to a telephone keypad (or a touchtone keypad), it is arranged from 1 to 0, downwards, with the number zero placed in between the asterisk and the pound sign. Each number has designated letters in the alphabet.

shiftOut()

The function `shiftOut()` is used to shift out a byte of data one bit at a time. It starts from either the most (i.e. the leftmost) or least (rightmost) significant bit. Each bit is written in turn to a data pin, after which a clock pin is pulsed to indicate that the bit is available.

Syntax: `shiftOut(dataPin, clockPin, bitOrder, value)`

Parameters:

dataPin – the pin on which to output each bit (int)

clockPin – the pin to toggle once the dataPin has been set to the correct value (int)

bitOrder – the order to which to shift out the bits; either **MSBFIRST** (most significant bit first) or **LSBFIRST** (least significant bit first).

value – the data to shift out. (byte)

The `shiftout()` command is most applicable in controlling an LED matrix which has a shift register (commonly a 74HC595) by scanning the columns and the rows and the same time, communicating a data byte from the register bit by bit. A clock pin is pulsed to delineate between bits. The `bitOrder` parameter distinguishes which type of order is to be used, whether the most significant bit or the least significant bit of a byte is to be shifted out first. The `value` parameter serves as the data byte to be shifted out. It may be declared in binary or hexadecimal format. The same concept applies to controlling a keypad. Below shows a sample program which controls a 74HC595 shift register:

```
//Pin connected to pin 12 of 74HC595
int latchPin = 8;
//Pin connected to pin 11 of 74HC595
int clockPin = 12;
//Pin connected to pin 14 of 74HC595
int dataPin = 11;

void setup()
{
  //set pins to output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}
void loop()
{
  //count up routine
  for (int j = 0; j < 256; j++)
  {
    //ground latchPin and hold low for as long as you are //transmitting
    digitalWrite(latchPin, LOW);

    //output the data indicated by the variable j
    shiftOut(dataPin, clockPin, LSBFIRST, j);

    //return the latch pin high to signal chip that it
    //no longer needs to listen for information
    digitalWrite(latchPin, HIGH);
    delay(1000);
  }
}
```

```

}
}

```

Arduino Tone Library (Tone.h)

The Tone.h library enables you to produce square-wave of the specified frequency on any Arduino pin. A duration can optionally be specified; otherwise the wave continues until `stop()` is called. The pin can be connected to a piezo buzzer or other speaker to play tones.

Tone.h library contains the tone constants with defined frequencies. Below is the listing of tone constants and their corresponding frequencies:

Constant Name	Frequency (Hz)	Constant Name	Frequency (Hz)
NOTE_B2	123	NOTE_G5	831
NOTE_C3	131	NOTE_A5	880
NOTE_CS3	139	NOTE_AS5	932
NOTE_D3	147	NOTE_B5	988
NOTE_DS3	156	NOTE_C6	1047
NOTE_E3	165	NOTE_CS6	1109
NOTE_F3	175	NOTE_D6	1175
NOTE_FS3	185	NOTE_DS6	1245
NOTE_G3	196	NOTE_E6	1319
NOTE_GS3	208	NOTE_F6	1397
NOTE_A3	220	NOTE_FS6	1480
NOTE_AS3	233	NOTE_G6	1568
NOTE_B3	247	NOTE_GS6	1661
NOTE_C4	262	NOTE_A6	1760
NOTE_CS4	277	NOTE_AS6	1865
NOTE_D4	294	NOTE_B6	1976
NOTE_DS4	311	NOTE_C7	2093
NOTE_E4	330	NOTE_CS7	2217
NOTE_F4	349	NOTE_D7	2349
NOTE_FS4	370	NOTE_DS7	2489
NOTE_G4	392	NOTE_E7	2637
NOTE_GS4	415	NOTE_F7	2794
NOTE_A4	440	NOTE_FS7	2960
NOTE_AS4	466	NOTE_G7	3136
NOTE_B4	494	NOTE_GS7	3322
NOTE_C5	523	NOTE_A7	3520
NOTE_CS5	554	NOTE_AS7	3729
NOTE_D5	587	NOTE_B7	3951
NOTE_DS5	622	NOTE_C8	4186
NOTE_E5	659	NOTE_CS8	4435
NOTE_F5	698	NOTE_D8	4699
NOTE_FS5	740	NOTE_DS8	4978
NOTE_G5	784		

Table 2.1. List of tone constants

Methods:

- `begin(pin)` – prepares a pin for playing a tone.
- `isPlaying()` – returns true if tone is playing, false if not.
- `play(frequency [, duration])` – play a tone.
 - *frequency* is in Hertz, and the *duration* is in milliseconds.
 - *duration* is optional. If *duration* is not given, tone will play continuously until `stop()` is called.
 - `play()` is non-blocking. Once called, `play()` will return immediately. If *duration* is given, the tone will play for that amount of time, and then stop automatically.
- `stop()` – stop playing a tone.

Sample Program:

```
#include <Tone.h> //import Tone.h library

Tone tone1; //instantiate a Tone object named tone1

void setup()
{
    tone1.begin(13); //connect the buzzer or speaker to pin 13
}

void loop()
{
    tone1.play(NOTE_A4); // plays the note A4
}
```

Serial Monitor

All Arduino boards have at least one serial port (also known as a UART or USART): **Serial**. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

Arduino environment's Serial Monitor lets you communicate with the Arduino board. In Serial Monitor you could see anything the Arduino sends to its serial port. It is very useful for debugging purposes, like tracking variable values and reading input and output values coming to and from the Arduino.

The common functions used in this library are listed below.

- `begin(speed)` – sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate. The common setting for the baud rate is 9600 bps.
- `end()` - disables serial communication, allowing the RX and TX pins to be used for general input and output. To re-enable serial communication, call `Serial.begin()`.
- `print(val)` – prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is. For example:

```
Serial.print(78) //gives "78"
Serial.print(1.23456) //gives "1.23"
Serial.print(byte(78)) //gives "N" (whose ASCII value is 78)
Serial.print('N') //gives "N"
Serial.print("Hello world.") //gives "Hello world."
```

- `println(val)` – prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or `'\r'`) and a newline character (ASCII 10, or `'\n'`). This command takes the same forms as `Serial.print()`.

Trainer Board Schematic Diagram

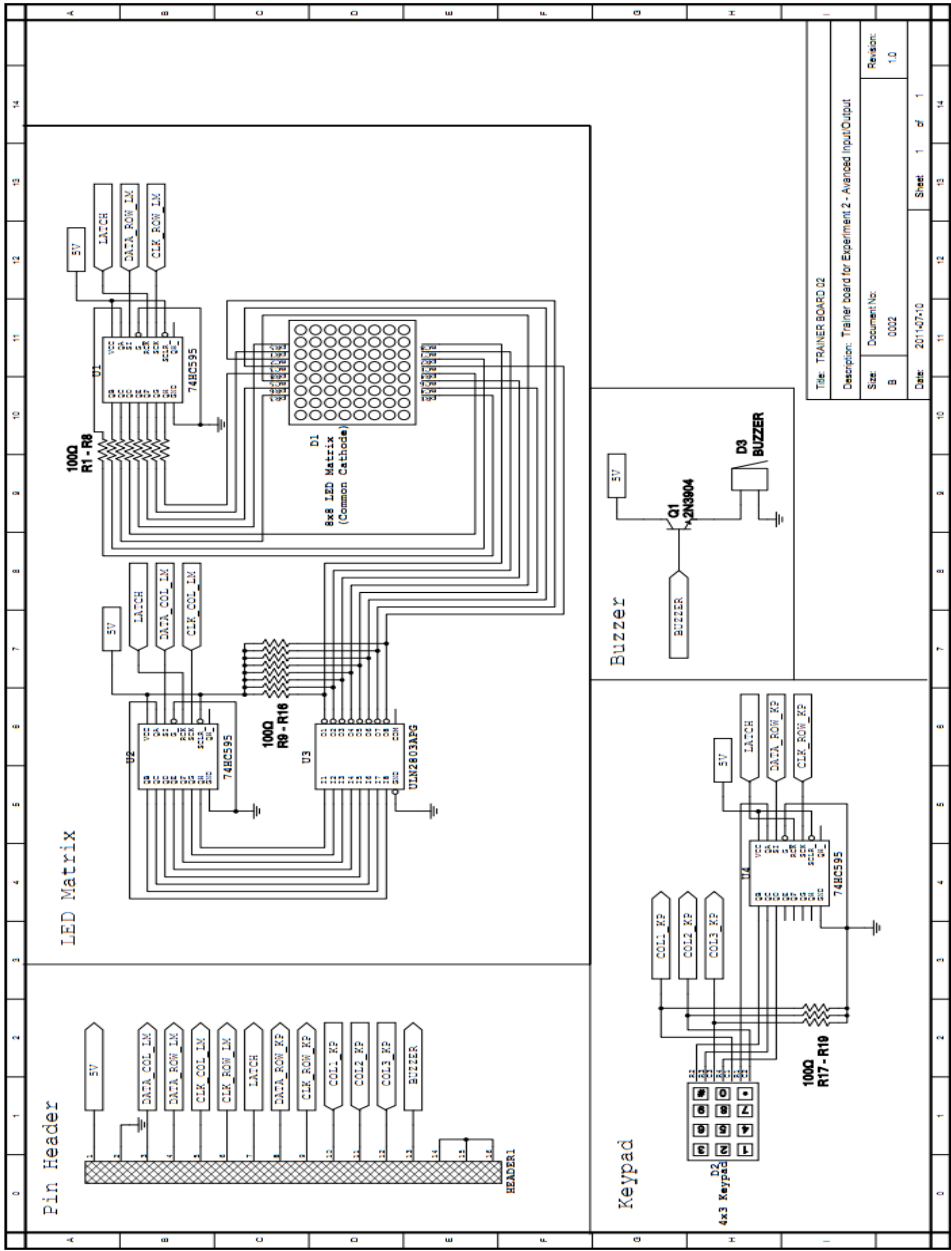


Figure 2.3. Trainer Board 02

Procedures

Part I. Numeric keypad

A. Read keypad press

1. Encode the sketch below and upload it to the Arduino board:

```
int latchPin = 12;
int clockPin = 11;
int dataPin = 9;
int c1=7;
int c2=6;
int c3=5;
int j;

//row scan signals; this is used to generate a
//continuous bit shifting to detect key presses
//along the 4 rows of the keypad
int mat[4] = {B0001,B0010,B0100,B1000};

void setup()
{
  Serial.begin(9600);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(c1, INPUT);
  pinMode(c2, INPUT);
  pinMode(c3, INPUT);
}

void loop()
{
  //Scan the 4 rows of the keypad for key presses
  for (j=0; j<=3; j++)
  {
    //generate row scan signal
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, MSBFIRST, mat[j]);
    digitalWrite(latchPin, HIGH);

    //check if a key in the first column has been
    //pressed
    if (digitalRead(c1)==HIGH)
    {
      //check if scan signal is in row 1
      if(j==0)
      {
```

```

        Serial.println("1"); //key pressed is 1
    }
    //check if scan signal is in row 2
    else if (j==1)
    {
        Serial.println("4"); //key pressed is 4
    }
    //check if scan signal is in row 3
    else if (j==2)
    {
        Serial.println("7"); //key pressed is 7
    }
    //check if scan signal is in row 4
    else if (j==3)
    {
        Serial.println("*"); //key pressed is *
    }
}
//check if a key in the second column has been
//pressed
else if (digitalRead(c2)==HIGH)
{
    //check if scan signal is in row 1
    if(j==0)
    {
        Serial.println("2"); //key pressed is 2
    }
    //check if scan signal is in row 2
    else if (j==1)
    {
        Serial.println("5"); //key pressed is 5
    }
    //check if scan signal is in row 1
    else if (j==2)
    {
        Serial.println("8"); //key pressed is 8
    }
    //check if scan signal is in row 1
    else if (j==3)
    {
        Serial.println("0"); //key pressed is 0
    }
}
//check if a key in the third column has been
//pressed
else if (digitalRead(c3)==HIGH)
{
    //check if scan signal is in row 1
    if(j==0)
    {
        Serial.println("3"); //key pressed is 3
    }
}

```

```

        //check if scan signal is in row 1
        else if (j==1)
        {
            Serial.println("6"); //key pressed is 6
        }
        //check if scan signal is in row 1
        else if (j==2)
        {
            Serial.println("9"); //key pressed is 9
        }
        //check if scan signal is in row 1
        else if (j==3)
        {
            Serial.println("#"); //key pressed is #
        }
    }

    //nothing is pressed
    else
    {
        Serial.println(" ");
    }
}
}

```

2. Connect pins 12, 11, 9, 7, 6 and 5 of the Gizduino to pins LATCH, CLK_ROW_KP, DATA_ROW_KP, COL1_KP, COL2_KP and COL3_KP of the trainer board respectively (refer to Figure 2.3).
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Open Serial Monitor. Press the keys in the keypad and observe the output displayed on Serial Monitor.

Part II. LED Matrix

A. Display a single character

1. Encode the sketch below and upload it to the Arduino board:

```

int latchPin = 8;
int clockPinR = 9;
int clockPinC = 10;
int dataPinR = 11;
int dataPinC = 12;

//column scan signals; equivalent to shifting values of 1
//which activates 1 column of the LED matrix at a time
int matC[8] = {B00000001,

```

```

        B00000010,
        B00000100,
        B00001000,
        B00010000,
        B00100000,
        B01000000,
        B10000000};

//row scan signals; contain 8 bytes of data which lights up
//specific LEDs in a row for each column of the LED matrix.
//These 8 bytes constitute the character to be formed in
//the LED matrix.

int matR[8] = {0xFF,0xFE,0x0C,0x18,0x18,0x0C,0xFE,0xFF};

void setup()
{
    //initialize I/O pins
    pinMode(latchPin, OUTPUT);
    pinMode(clockPinR, OUTPUT);
    pinMode(clockPinC, OUTPUT);
    pinMode(dataPinR, OUTPUT);
    pinMode(dataPinC, OUTPUT);
}

void loop()
{
    //scan the 8 columns of the LED matrix
    for (int i=0; i<=7;i++)
    {
        digitalWrite(latchPin, LOW);
        //byte of data which lights up specific LEDs in a //row
        shiftOut(dataPinR, clockPinR, MSBFIRST, matR[i]);
        //current column scanned
        shiftOut(dataPinC, clockPinC, MSBFIRST, matC[i]);
        digitalWrite(latchPin, HIGH);
        delay(1);
    }
}

```

2. Connect pins 8-12 of the Gizduino to pins LATCH, CLK_ROW_LM, CLK_COL_LM, DATA_ROW_LM, and DATA_COL_LM of the trainer board respectively.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Observe the output displayed on the LED matrix.

B. Display multiple characters

1. Encode the sketch below and upload it to the Arduino board:

```
int latchPin = 8;
int clockPinR = 9;
int clockPinC = 10;
int dataPinR = 11;
int dataPinC = 12;
int i = 0, j = 0;
unsigned long currentMillis;
long previousMillis = 0;
long interval = 3000;

//column scan signals
int matC[8] = {B00000001,
               B00000010,
               B00000100,
               B00001000,
               B00010000,
               B00100000,
               B01000000,
               B10000000};

//row scan signals; this is a 2D matrix containing the 8
//characters to be displayed in the LED matrix
int matR2[8][8] =
{{0xFF,0xFE,0x0C,0x18,0x18,0x0C,0xFE,0xFF},
 {0xC3,0xC3,0xC3,0xFF,0xFF,0xC3,0xC3,0xC3},
 {0xFF,0xFF,0xC3,0xC3,0xC3,0xC3,0xC3,0xC3},
 {0xFF,0xFF,0x33,0x33,0x33,0x33,0xF3,0xBE},
 {0xFF,0xFF,0xC3,0xC3,0xC3,0xC3,0xFF,0xFF},
 {0xFF,0xFF,0xC0,0xC0,0xC0,0xC0,0xC0,0xC0},
 {0xF8,0xFC,0x36,0x33,0x33,0x36,0xFC,0xF8},
 {0xFF,0xFF,0xDB,0xDB,0xDB,0xDB,0xFF,0x66}};

void setup()
{
    //set pins to output
    pinMode(latchPin, OUTPUT);
    pinMode(clockPinR, OUTPUT);
    pinMode(clockPinC, OUTPUT);
    pinMode(dataPinR, OUTPUT);
    pinMode(dataPinC, OUTPUT);
}

void loop()
{
    currentMillis = millis(); //get current time
```



```

//check if 3 seconds has passed
if(currentMillis - previousMillis > interval)
{
  //save the last time you displayed a character to
  //the LED matrix
  previousMillis = currentMillis;

  //check if the last character has been displayed
  if (j == 7)
  {
    j=0; //display the first character
  }
  else
  {
    j++; //display the next character
  }
}

//scan the 8 columns of the LED matrix
for (i=0; i<=7;i++)
{
  digitalWrite(latchPin, LOW);
  //byte of data for row
  shiftOut(dataPinR, clockPinR, MSBFIRST, matr2[j][i]);
  //current column scanned
  shiftOut(dataPinC, clockPinC, MSBFIRST, matC[i]);
  digitalWrite(latchPin, HIGH);
  delay(1);
}
}

```

2. Connect pins 8-12 of the Gizduino to pins LATCH, CLK_ROW_LM, CLK_COL_LM, DATA_ROW_LM, and DATA_COL_LM of the trainer board respectively.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Observe the output displayed on the LED matrix.

Part III. Buzzer

A. Play tones

1. Encode the program below and upload it to the Arduino board:

```

#include <Tone.h>

//notes to play
int doremi[8] =

```

```

{
NOTE_C6,
NOTE_D6,
NOTE_E6,
NOTE_F6,
NOTE_G6,
NOTE_A6,
NOTE_B6,
NOTE_C7,
};

Tone tone1; //instantiate a Tone object
int thisNote;

void setup()
{
    tone1.begin(13); //attach buzzer to pin 13
}

void loop()
{

    //play do-re-mo-fa-sol-la-ti-do
    for (thisNote = 0; thisNote <= 7; thisNote++)
    {
        tone1.play(doremi[thisNote], 2500);
        delay(1000);
    }
    tone1.stop(); //stop playing tone
    delay(2000);
    //play do-ti-la-sol-fa-mi-re-do
    for (thisNote = 7; thisNote >= 0; thisNote--)
    {
        tone1.play(doremi[thisNote], 2500);
        delay(1000);
    }
    tone1.stop(); //stop playing tone
    delay(2000);
}

```

2. Connect pin 13 of the Gizduino to the BUZZER pin on the trainer board.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Listen to the output produced by the buzzer.

Activities

- a. Create a program that accepts input from a keypad and then outputs the key pressed in the LED matrix. All the keys in the keypad should be used.
- b. Create a program that generates a tone whenever the keypad is pressed. Define your own tone frequencies for each of the key in the keypad.
- c. Scrolling text on led displays are commonly used in advertising displays, promotional displays and information displays for shop windows, casinos, clubs, bars, restaurants, museums and other public areas. Create a program that scrolls the text “ARDUINO” on the LED matrix. Use three keys on the keypad to set the scroll speed of the text to slow, medium and fast.

References

www.arduino.cc/en/Tutorial/ShiftOut

<http://code.google.com/p/roguetone/wiki/ToneLibraryDocumentation>

<http://arduino.cc/en/Reference/Serial>

Experiment No. 3

ANALOG-TO-DIGITAL CONVERSION

Objectives

At the end of the experiment the student should be able to:

1. Understand how the Arduino converts analog input signals to digital signals.
2. Control LED and RGB LED outputs from an analog input.
3. Detect the ambient light intensity using a light-dependent resistor.
4. Use Pulse Width Modulation (PWM) in controlling LED and RGB LED outputs.
5. Calibrate analog input readings.
6. Map an input analog reading to a desired range of output values.

Equipment and Materials

- 1 pc Gizduino
- 1 pc Trainer Board 03
- 1 pc USB Standard A-B Cable
- 1 set Connecting Wires
- 1 set Desktop computer

Discussion

Analog-to-Digital Conversion

Connecting digital circuitry to sensor devices is simple if the sensor devices are inherently digital themselves. Switches, relays, and encoders are easily interfaced with gate circuits due to the on/off nature of their signals. However, when analog devices are involved, interfacing becomes much more complex. Electronically translating analog signals into digital (binary) quantities is possible with the use of an *analog-to-digital converter*.

Analog-to-digital conversion is the process of converting a continuous quantity to a discrete time digital representation. The electronic device used for this process is an analog-to-digital converter (ADC). ADCs convert an input analog voltage or current to a digital number proportional to the magnitude of the voltage or current.



Figure 3.1. A 4-channel stereo multiplexed analog-to-digital converter chip placed on a sound card.

ADCs are used virtually everywhere where an analog signal has to be processed, stored, or transported in digital form. Fast video ADCs are used, for example, in TV tuner cards. Slow on-chip 8, 10, 12, or 16 bit ADCs are common in microcontrollers. Very fast ADCs are needed in digital oscilloscopes, and are crucial for new applications like software defined radio.

Pulse Width Modulation (PWM)

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In Figure 3.2 below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to `analogWrite()` is on a scale of 0 - 255, such that `analogWrite(255)` requests a 100% duty cycle (always on), and `analogWrite(127)` is a 50% duty cycle (on half the time) for example.

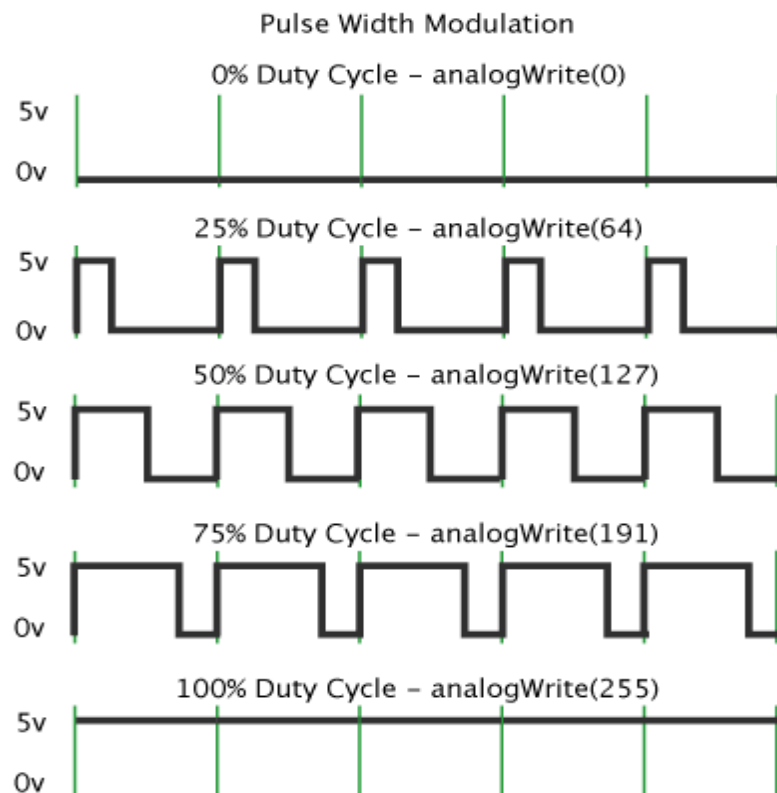


Figure 3.2. Various PWM signals.

Light Dependent Resistor (LDR)

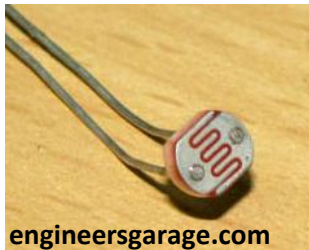


Figure 3.3. An LDR.

Photoresistors or light dependent resistors (LDRs) are very useful especially in light/dark sensor circuits. As its name suggests, it offers resistance in response to the ambient light. The resistance decreases as the intensity of incident light increases, and vice versa. In the absence of light, LDR exhibits a resistance of the order of mega-ohms which decreases to few hundred ohms in the presence of light. It can act as a sensor, since a varying voltage drop can be obtained in accordance with the varying light.

Photoresistors come in many different types. The most common are the inexpensive cadmium sulfide cells which could be found in many consumer items such as camera light meters, street lights, clock radios, alarm devices, and outdoor clocks.

RGB LED

An RGB LED is actually three LEDs in one bulb. The housing contains separate red, blue and green LEDs which share a common cathode, or negative terminal. The brightness of each color is determined by its input voltage. By combining the three colors in different amounts, you can turn the LED any color you want. Usually, the output color can be controlled by sending PWM signals to the leads of the RGB LED.



Figure 3.4. An RGB LED.

Calibrating Analog Input

Sometimes analog readings doesn't fit exactly the entire range of ADC output values (0 – 1023) of the Arduino. This happens mostly on interfacing sensors in which only a subset range of the possible input voltage range (0 – 5V) is utilized. Hence in this case, rescaling of the ADC output would be necessary in order for the Arduino to use a valid range of values which might be needed in some calculations. This could be done by initial calibration of the sensor before starting the main program.

This example below demonstrates one technique for calibrating sensor input. The Arduino takes sensor readings for five seconds during the startup, and tracks the highest and lowest values it gets. These sensor readings during the first five seconds of the sketch execution define the minimum and maximum of expected values for the readings taken during the loop.

```
const int sensorPin = A0;    // pin that the sensor is attached to
const int ledPin = 9;        // pin that the LED is attached to

//The initialization below may seem backwards. Initially, you set the
//minimum high and read for anything lower than that, saving it as the
//new minimum. Likewise, you set the maximum low and read for anything
```

```

//higher as the new maximum.
int sensorValue = 0;           // the sensor value
int sensorMin = 1023;          // minimum sensor value
int sensorMax = 0;             // maximum sensor value

void setup() {
  // turn on LED to signal the start of the calibration period:
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);

  // calibrate during the first five seconds
  while (millis() < 5000) {
    sensorValue = analogRead(sensorPin);

    // record the maximum sensor value
    if (sensorValue > sensorMax) {
      sensorMax = sensorValue;
    }

    // record the minimum sensor value
    if (sensorValue < sensorMin) {
      sensorMin = sensorValue;
    }
  }

  // signal the end of the calibration period
  digitalWrite(13, LOW);
}

void loop() {
  // read the sensor:
  sensorValue = analogRead(sensorPin);

  // apply the calibration to the sensor reading
  sensorValue = map(sensorValue, sensorMin, sensorMax, 0, 255);

  // fade the LED using the calibrated value:
  analogWrite(ledPin, sensorValue);
}

```

Useful Arduino ADC Functions

- `analogRead(pin)`

Description

Reads the value from the specified analog pin. The Gizduino board contains a 6 channel 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit.

Parameters

pin: the number of the analog input pin to read from. (A0-A5 or 0-5)

Returns

An integer value ranging from 0 to 1023

Example

```
// potentiometer wiper (middle terminal) connected to analog pin 3;
// outside leads to ground and +5V
int analogPin = 3;

int val = 0; //variable to store the value read

void setup()
{
  Serial.begin(9600); //setup serial
}

void loop()
{
  val = analogRead(analogPin);    //read the input pin
  Serial.println(val);            //print value
}
```

- `analogWrite(pin, value)`

Description

Writes an analog value (PWM wave) to a pin. It can be used to light a LED at varying brightness or driving a motor at various speeds. After a call to **analogWrite()**, the pin will generate a steady square wave of the specified duty cycle until the next call to **analogWrite()** (or a call to **digitalRead()** or **digitalWrite()** on the same pin). The frequency of the PWM signal is approximately 490 Hz. You do not need to call `pinMode()` to set the pin as an output before calling `analogWrite()`. This function has nothing whatsoever to do with the analog pins or the `analogRead()` function.

Parameters

pin: the pin to write to (works only on pins 3, 5, 6, 9, 10 and 11 of the Gizduino)

value: the duty cycle: between 0 (0% duty cycle, always off) and 255 (100% duty cycle, always on).

Returns

Nothing.

Example

The sketch below sets the output to the LED proportional to the value read from the potentiometer.


```

int ledPin = 9;          //LED connected to digital pin 9
int analogPin = 3;      //potentiometer connected to analog pin 3
int val = 0;            //variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT); //set the pin as output
}

void loop()
{
  val = analogRead(analogPin); //read the input pin

  //analogRead values go from 0 to 1023, analogWrite values from
  //0 to 255
  analogWrite(ledPin, val / 4);
}

```

- `map(value, fromLow, fromHigh, toLow, toHigh)`

Description

Re-maps a number from one range to another. That is, a **value** of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.

Note that the "lower bounds" of either range may be larger or smaller than the "upper bounds" so the `map()` function may be used to reverse a range of numbers, for example

```
y = map(x, 1, 50, 50, 1);
```

The function also handles negative numbers well, so that this example

```
y = map(x, 1, 50, 50, -100);
```

is also valid and works well.

The `map()` function uses integer math so will not generate fractions, when the math might indicate that it should do so. Fractional remainders are truncated, and are not rounded or averaged.

Parameters

value: the number to map

fromLow: the lower bound of the value's current range

fromHigh: the upper bound of the value's current range

toLow: the lower bound of the value's target range

toHigh: the upper bound of the value's target range

Returns

The mapped value.

Example

```
/* Map an analog value to 8 bits (0 to 255) */
int ledPin = 9;
int potPin = A0;

void setup() {}

void loop()
{
    int val = analogRead(potPin); //read analog input
    val = map(val, 0, 1023, 0, 255); //map read analog input
    analogWrite(ledPin, val); //output mapped value to a LED
}
```

Trainer Board Schematic Diagram

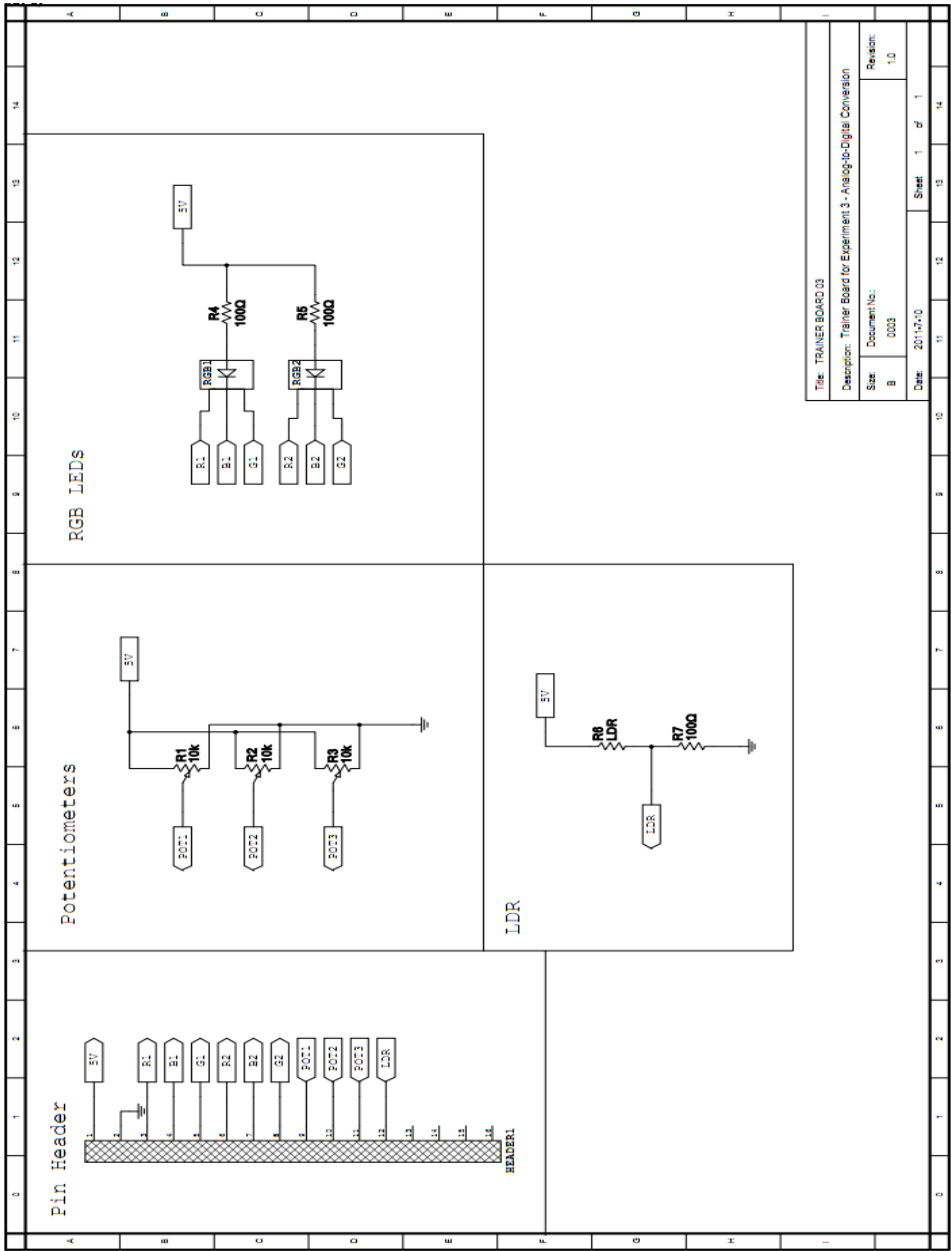


Figure 3.5. Trainer Board 03

Procedures

Part I. Basic Analog I/O

A. Control blink delay of a LED using a potentiometer

1. Encode the sketch below and upload it to the Arduino board:

```
//input pin for the potentiometer
int sensorPin = A0;
//use onboard LED
int ledPin = 13;
// variable to store the value coming from the sensor
int sensorValue = 0;

void setup()
{
    //declare the ledPin as an OUTPUT:
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    //read the value from the sensor:
    sensorValue = analogRead(sensorPin);

    //turn the ledPin on
    digitalWrite(ledPin, HIGH);

    //stop the program for <sensorValue> milliseconds:
    delay(sensorValue);

    //turn the ledPin off
    digitalWrite(ledPin, LOW);

    //stop the program for <sensorValue> milliseconds:
    delay(sensorValue);
}
```

2. Connect pin A0 of the Gizduino to a potentiometer in the trainer board (refer to Figure 3.5).
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Turn the potentiometer clockwise and counterclockwise. Observe the behavior of the onboard LED in the Gizduino as you turn the potentiometer.

B. Fade LED output using PWM

1. Encode the program below and upload it to the Arduino board:

```
int ledPin = 9;      //output LED pin
int fadeVal = 0;     //stores LED fade value
void setup()
{
    // nothing happens in setup
}

void loop()
{
    //fade in from min to max in increments of 5 points:
    for(fadeVal = 0 ; fadeVal <= 255; fadeVal +=5)
    {
        //output a PWM signal (range from 0 to 255):
        analogWrite(ledPin, fadeVal);
        //wait for 30 milliseconds to see the dimming effect
        delay(30);
    }

    //fade out from max to min in increments of 5 points:
    for(fadeVal = 255 ; fadeVal >= 0; fadeVal -=5)
    {
        //output a PWM signal (range from 0 to 255):
        analogWrite(ledPin, fadeVal);
        //wait for 30 milliseconds to see the dimming effect
        delay(30);
    }
}
```

2. Connect pin 9 of the Gizduino to one of the pins of the RGB LEDs (R1, G1, B1, R2, G2 or B2) in the trainer board (refer to Figure 3.1).
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Observe the RGB LED that you connected to the Gizduino.

C. Control LED brightness using PWM

1. Encode the program below and upload it to the Arduino board:

```
//Analog input pin that the pot is attached to
int analogInPin = A0;
//Analog output pin that the LED is attached to
int analogOutPin = 9;
//value read from the pot
int sensorValue = 0;
```

```

//value output to the PWM (analog out)
int outputValue = 0;

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

void loop()
{
    // read the analog input value:
    sensorValue = analogRead(analogInPin);
    // map it to the range of the analog out:
    outputValue = map(sensorValue, 0, 1023, 0, 255);
    // change the analog out value:
    analogWrite(analogOutPin, outputValue);

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

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

```

2. Connect pin 9 of the Gizduino to one of the pins of the RGB LEDs (R1, B1, G1, R2, B2 or G2) in the trainer board. Also, connect pin A0 of the Gizduino to a potentiometer in the trainer board.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Observe the RGB LED that you connected to the Gizduino.

Part II. Advanced Analog I/O

A. Detect ambient light intensity using LDR

1. Encode the program below and upload it to the Arduino board:

```

//LDR pin
const int sensorPin = A0;
//min sensor reading, discovered through calibration
int senMin;
//max sensor reading, discovered through calibration

```

```

int senMax;
//current sensor reading
int senReading;

void setup()
{
    //initialize serial communication:
    Serial.begin(9600);

    //use onboard LED
    pinMode(13, OUTPUT);
    //signal start of calibration
    digitalWrite(13, HIGH);

    //calibrate during the first five seconds
    while (millis() < 5000)
    {
        senReading = analogRead(sensorPin);
        Serial.println("Calibrating sensor...");
        //record the maximum sensor value
        if (senReading > senMax)
        {
            senMax = senReading;
        }

        //record the minimum sensor value
        if (senReading < senMin)
        {
            senMin = senReading;
        }
    }

    //signal the end of the calibration period
    digitalWrite(13, LOW);
}

void loop()
{
    //read the sensor:
    senReading = analogRead(A0);
    //map the sensor range to a range of four options:
    int range = map(senReading, senMin, senMax, 0, 3);

    //do something different depending on the
    //range value:
    switch (range)
    {
        case 0: //your hand is on the sensor
            Serial.println("dark");
            break;
        case 1: //your hand is close to the sensor
            Serial.println("dim");
    }
}

```

```

        break;
    case 2: //your hand is a few inches from the sensor
        Serial.println("medium");
        break;
    case 3: //your hand is nowhere near the sensor
        Serial.println("bright");
        break;
    }
}

```

2. Connect pin A0 of the Gizduino to the LDR in the trainer board.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Open Serial Monitor. Try varying the light received by the LDR by covering it or shining a flashlight on it. Observe the messages displayed in the Serial Monitor.

B. Control an RGB LED using PWM

1. Encode the program below and upload it to the Arduino board:

```

//stores potentiometer readings
int potValue1 = 0;
int potValue2 = 0;
int potValue3 = 0;

//PWM signals used to drive the RGB LED
int outRed = 0;
int outGreen = 0;
int outBlue = 0;

void setup()
{
    //nothing happens in setup()
}

void loop()
{
    //read the analog input values:
    potValue1 = analogRead(A0);
    potValue2 = analogRead(A1);
    potValue3 = analogRead(A2);
    //map analog input range to PWM output range
    outRed = map(potValue1, 0, 1023, 0, 255);
    outGreen = map(potValue2, 0, 1023, 0, 255);
    outBlue = map(potValue3, 0, 1023, 0, 255);
    //change the analog out value:
    analogWrite(11, outRed);
}

```



```

    analogWrite(10, outBlue);
    analogWrite(9, outGreen);

    //wait for 10 ms for the ADC to settle
    delay(10);
}

```

2. Connect pins A0, A1 and A2 of the Gizduino to the potentiometers in the trainer board. Also, connect pins 9-11 of the Gizduino to pins R1, G1 and B1 of the trainer board respectively.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Vary the three potentiometers then observe the RGB LED output.

Activities

- a. Create a program that uses blinks and fades two RGB LEDs. Define your own blinking and fading sequences and delays.
- b. Create a program that detects two states (0 and 1) for each three potentiometers and outputs the results to two RGB LEDs. Print the states of the potentiometers to Serial monitor. Follow the input-output relation in the table below:

POT1	POT2	POT3	RGB1	RGB2
0	0	0	RED	BLUE
0	0	1	BLUE	GREEN
0	1	0	GREEN	RED
0	1	1	RED	GREEN
1	0	0	BLUE	RED
1	0	1	PURPLE	CYAN
1	1	0	CYAN	YELLOW
1	1	1	YELLOW	PURPLE

Note:

RED + BLUE = PURPLE

BLUE+GREEN = CYAN

RED+GREEN=YELLOW

- c. Hue is the property of light by which the color of an object is classified as red, blue, green, or yellow. Create a program that controls the hue of a RGB LED using a potentiometer. Map the input analog values to the range of the valid hue values (0-360). Display this hue value and its equivalent RGB value in the Serial Monitor. Use the HSV (hue, saturation, value) to RGB conversion pseudocode

below to help you in creating your program. (Note: Define S and V as constants equal to 1 since we only want to control the hue, not the saturation and value.)

```
//HSV to RGB pseudocode
if ( S == 0 )                               //HSV from 0 to 1
{
    R = V * 255
    G = V * 255
    B = V * 255
}
else
{
    var_h = H * 6
    if (var_h == 6)
    {
        var_h = 0
    }
    var_i = int(var_h)
    var_1 = V * (1 - S)
    var_2 = V * (1 - S * (var_h - var_i))
    var_3 = V * (1 - S * (1 - (var_h - var_i)))

    if (var_i == 0)
    {
        var_r = V;
        var_g = var_3;
        var_b = var_1;
    }
    else if (var_i == 1)
    {
        var_r = var_2;
        var_g = V;
        var_b = var_1;
    }
    else if (var_i == 2)
    {
        var_r = var_1;
        var_g = V;
        var_b = var_3
    }
    else if (var_i == 3)
    {
        var_r = var_1;
        var_g = var_2;
        var_b = V;
    }
    else if (var_i == 4)
    {
        var_r = var_3;
        var_g = var_1;
        var_b = V;
    }
}
```

```
    }  
    else  
    {  
        var_r = V;  
        var_g = var_1;  
        var_b = var_2;  
    }  
    //RGB results from 0 to 255  
    R = var_r * 255  
    G = var_g * 255  
    B = var_b * 255  
}
```

References

http://en.wikipedia.org/wiki/Analog-to-digital_converter
http://www.allaboutcircuits.com/vol_4/chpt_13/1.html
<http://www.engineersgarage.com/electronic-components/ldr-light-dependant-resistor>
<http://en.wikipedia.org/wiki/Photoresistor>
<http://arduino.cc/en/Tutorial/PWM>
<http://www.arduino.cc/en/Reference/AnalogRead>
<http://www.arduino.cc/en/Reference/Map>
<http://www.arduino.cc/en/Reference/AnalogWrite>

Experiment No. 4
MOTOR CONTROL

Objectives

At the end of the experiment the student should be able to:

1. Understand how DC motors, servo motors and stepper motors work.
2. Understand how to use interrupts in Arduino.
3. Interface DC motors, servo motors and stepper motors to an Arduino board.

Equipment and Materials

- 1 pc Gizduino
- 1 pc Trainer Board 04
- 1 pc USB Standard A-B Cable
- 1 pc 9 V DC adaptor
- 1 pc DC motor
- 1 pc Servo motor
- 1 pc Stepper motor
- 1 set Connecting Wires

Discussion

DC Motor

A DC motor is a fairly simple electronic motor that uses electricity and a magnetic field to produce torque, which turns the motor. There are two controllable parameters of a DC motor; direction and speed. To control the direction, the polarity of the motor is reversed. To control the speed, the input voltage is varied using pulse width modulation (PWM).



optms.net
Figure 4.1. A DC Motor.

Direction Control

To control a DC motor from a microcontroller, a switching arrangement known as *H-bridge* is used (see Figure 4.2).

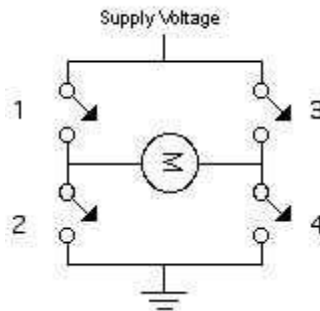


Figure 4.2. Schematic of an H-bridge circuit.

When switches 1 and 4 are closed and 2 and 3 are open, voltage flows from the supply to 1 to the motor to 4 to ground. When 2 and 3 are closed and 1 and 4 are open, polarity is reversed, and voltage flows from the supply to 3 to the motor to 2 to ground.

Although an H-bridge is easy to construct, it is usually easier to use a controller manufactured specifically for the job. A pre-manufactured H-bridge chip will include diodes to protect the transistors from back voltage, sometimes a current sensing pin to sense the current the motor is drawing, and much more. One available chip is the one used in Trainer Board 04, which is the L293D push-pull four channel driver. This chip is capable of controlling two DC motors at once.

Speed

The speed of a DC motor is proportional to the supplied voltage. If the voltage drops too far, the motor won't get enough power to turn, but within a certain range, usually 50% of the rated voltage, the motor will run at varying speeds. The most effective way to adjust the speed is by using pulse width modulation (PWM). This means that you pulse the motor on and off at varying rates, to simulate a voltage. Below are some examples of pulse widths and the voltages they would simulate:

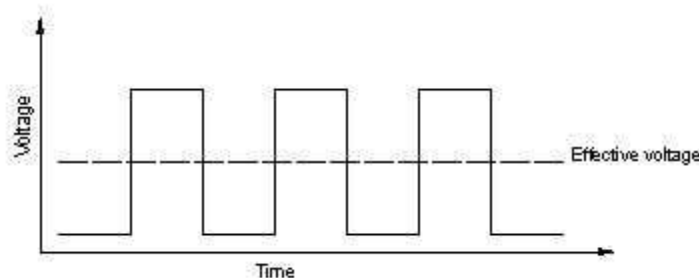


Figure 4.3. PWM signal with a 50% duty cycle. A 50% duty cycle results to an effective voltage which is about half the total voltage.

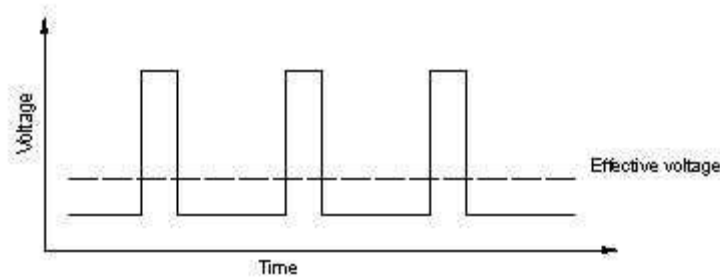


Figure 4.4. PWM signal with a 25% duty cycle. A 25% duty cycle results to an effective voltage which is about a quarter of the total voltage.

A PWM signal could be produced easily in Arduino by using the command `analogWrite()`. In this experiment, speed control is achieved by feeding the PWM signal to one of the “enable” pins of the motor driver chip L293D, which in turn switches on and off the DC motor based on the input PWM signal pattern and thus effectively simulating different voltage levels.

Servo Motors



Figure 4.5. A Servo motor.

A servo motor is an electromechanical device in which an electrical input determines the position of the armature of a motor. Servos have integrated gears and a shaft that can precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.

Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow or orange and should be connected to pin 9 on the Arduino board.

Arduino Servo Library (Servo.h)

The Servo library allows control of up to 12 Servo motors on most Arduino boards and 48 on the Arduino Mega. On boards other than the Mega, use of the library disables the `analogWrite()` functionality on pins 9 and 10, whether or not there is a Servo on those pins.

The most commonly used methods in the Servo.h library are listed below.

- `attach(pin)` – attaches the Servo variable to a pin.

- `write(angle)` – writes a value to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation. On a continuous rotation servo, this will set the speed of the servo (with 0 being full-speed in one direction, 180 being full speed in the other, and a value near 90 being no movement).
- `read()` – read the current angle of the servo (the value passed to the last call to `write()`).
- `detach()` – detach the Servo variable from its pin. If all Servo variables are detached, then pins 9 and 10 can be used for PWM output with `analogWrite()`.

Sample Program:

```
#include <Servo.h> //import Servo.h library

Servo myservo; //instantiate a Servo object

void setup()
{
  myservo.attach(9); // attach the servo on pin 9
}

void loop()
{
  //set the angle of the servo to 90 degrees
  myservo.write(90);
}
```

Stepper Motor

A stepper motor is a motor controlled by a series of electromagnetic coils. The center shaft has a series of magnets mounted on it, and the coils surrounding the shaft are alternately given current or not, creating magnetic fields which repulse or attract the magnets on the shaft, causing the motor to rotate. This design allows for very precise control of the motor: by proper pulsing, it can be turned in very accurate steps of set degree increments (e.g. two-degree increments, half-degree increments, etc.). They are commonly used in printers, disk drives, and other devices where precise positioning of the motor is necessary.



Figure 4.6. Stepper motors.

There are two basic types of stepper motors, unipolar stepper motors and bipolar stepper motors. Unipolar stepper motors typically have two coils per phase; one for

each direction of magnetic field. Bipolar stepper motors typically have one coil per phase, and current flows in both directions through this coil. Thus, of two motors of identical size, the bipolar would be able to produce twice as much torque, since at any given time, the unipolar motor is only using half of its windings.

Stepping Modes

Stepper motors are driven by converting electrical pulses into discrete mechanical movements. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence. The following are the most common drive modes (refer to Figure 4.7 for the following discussions):

- Single Stepping mode (Wave Drive)
 - In this mode only one winding is energized at any given time. The stator is energized according to the sequence $A \rightarrow B \rightarrow \bar{A} \rightarrow \bar{B}$ and the rotor steps from position $8 \rightarrow 2 \rightarrow 4 \rightarrow 6$.
- High Torque Stepping mode (Full Step Drive)
 - In this mode two phases are energized at any given time. The stator is energized according to the sequence $AB \rightarrow \bar{A}\bar{B} \rightarrow \bar{A}B \rightarrow A\bar{B}$ and the rotor steps from position $1 \rightarrow 3 \rightarrow 5 \rightarrow 7$. This mode results in the same angular movement as the single stepping mode but with greater torque output.
- Half Stepping mode
 - Half stepping mode combines both single stepping and high torque stepping modes. For every second step only one phase is energized and during the other steps one phase on each stator. The stator is energized according to the sequence $AB \rightarrow B \rightarrow \bar{A}\bar{B} \rightarrow \bar{A} \rightarrow \bar{A}B \rightarrow \bar{B} \rightarrow A\bar{B} \rightarrow A$ and the rotor steps from position $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$. These results in angular movements that are half of those in the previous stepping modes, hence allowing for more precise movements.

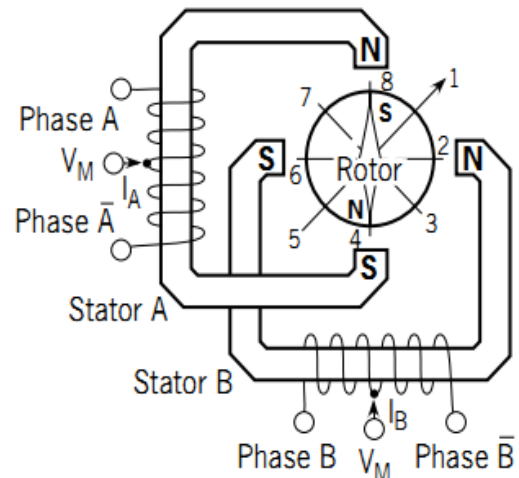


Figure 4.7. Internal windings of a unipolar stepper motor.

The excitation sequences for the above driving modes are summarized in Figure 4.8.

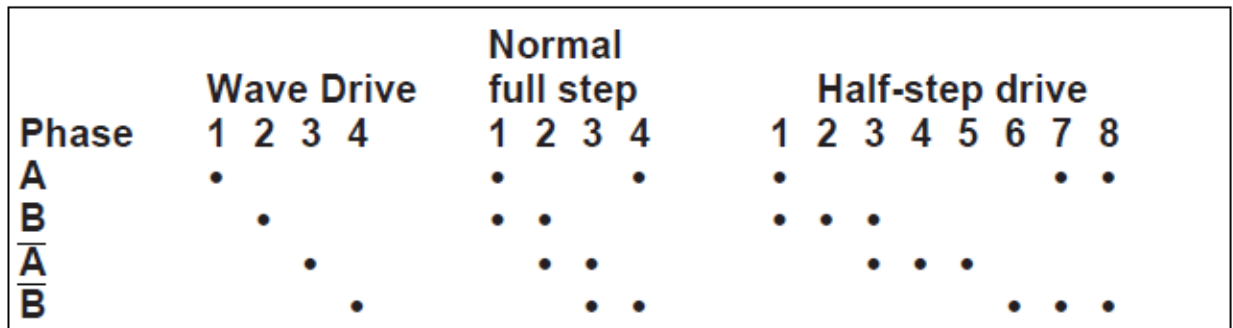


Figure 4.8. Excitation sequences for different stepping modes.

Interrupts

Interrupts are useful for making things happen automatically in microcontroller programs, and can help solve timing problems. A good task for using an interrupt might be reading a rotary encoder, monitoring user input.

If you wanted to insure that a program always caught the pulses from a rotary encoder, never missing a pulse, it would make it very tricky to write a program to do anything else, because the program would need to constantly poll the sensor lines for the encoder, in order to catch pulses when they occurred. Other sensors have a similar interface dynamic too, such as trying to read a sound sensor that is trying to catch a click, or an infrared slot sensor (photo-interrupter) trying to catch a coin drop. In all of these situations, using an interrupt can free the microcontroller to get some other work done while not missing the doorbell.

Functions:

- `attachInterrupt(interrupt, function, mode)`

Description

Specifies a function to call when an external interrupt occurs. Replaces any previous function that was attached to the interrupt. Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3).

Parameters

interrupt: the number of the interrupt (0 or 1)

function: the function to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an *interrupt service routine*.

mode defines when the interrupt should be triggered. Four constants are predefined as valid values:

- LOW to trigger the interrupt whenever the pin is low,
- CHANGE to trigger the interrupt whenever the pin changes value
- RISING to trigger when the pin goes from low to high,

- FALLING for when the pin goes from high to low.

Note

Inside the attached function, `delay()` won't work and the value returned by `millis()` will not increment. Serial data received while in the function may be lost. You should declare as volatile any variables that you modify within the attached function.

Example

```
int pin = 13;
volatile int state = LOW;

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}

void loop()
{
  digitalWrite(pin, state);
}

void blink()
{
  state = !state;
}
```

- `detachInterrupt(interrupt)`

Description

Turns off the given interrupt.

Parameters

interrupt: the number of interrupt to disable (0 or 1).

- `noInterrupts()` and `interrupts()`

Description

`noInterrupts()` disables interrupts while `interrupts()` re-enables interrupts (after they've been disabled by `noInterrupts()`). Interrupts allow certain important tasks to happen in the background and are enabled by default. Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.

Example

```
void setup() {}

void loop()
{
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

Trainer Board Schematic Diagram

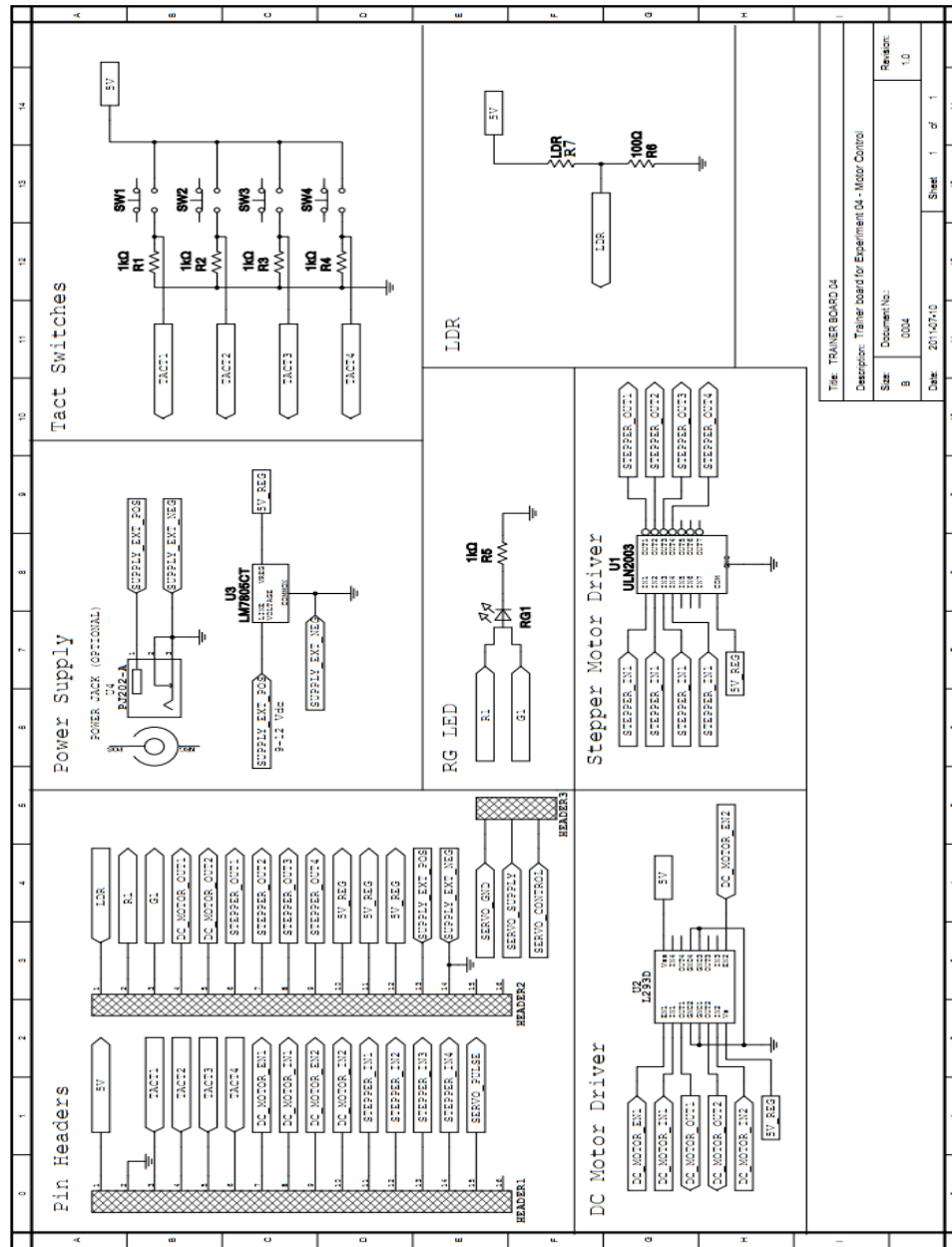


Figure 4.9. Trainer Board 04

Procedures

Part I. DC Motor Control

A. Forward/Reverse direction control

1. Encode the sketch below and upload it to the Arduino board:

```
//tact switch input pin
int inputPin = 1;
//output pins used to control the DC motor
int motorPin1 = 5, motorPin2 = 6;
//direction of rotation of the DC motor
int dir = LOW;
//used for detecting the state change of the tact switch
int prevState = 0, currentState = 0;

void setup()
{
    //initialize I/O pins
    pinMode(inputPin, INPUT);
    pinMode(motorPin1, OUTPUT);
    pinMode(motorPin2, OUTPUT);
}

void loop()
{
    //detect state change in the tact switch
    currentState = digitalRead(inputPin);

    if (currentState != prevState)
    {
        //if the current state is HIGH then the button
        //went from off to on
        if (currentState == HIGH)
        {
            //if the tact switch is pressed, toggle the
            //direction of rotation of the DC motor
            dir = !dir;
        }
    }

    prevState = currentState;

    //drive the DC motor clockwise or counterclockwise
    //as indicated by dir
    if (dir == HIGH)
    {
        digitalWrite(motorPin1, HIGH);
        digitalWrite(motorPin2, LOW);
    }
}
```

```

    }
    else
    {
        digitalWrite(motorPin1, LOW);
        digitalWrite(motorPin2, HIGH);
    }
}

```

2. Connect pins 1, 5 and 6 of the Gizduino to a tact switch, DC_MOTOR_IN1 pin and DC_MOTOR_IN2 pin of the trainer board respectively (refer to Figure 4.9).
3. Connect the pins DC_MOTOR_OUT1 and DC_MOTOR_OUT2 of the trainer board to the supply terminals of the DC motor.
4. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
5. Connect the 9 V DC adaptor to the DC jack in the trainer board.
6. Push the tact switch connected to the Gizduino. Observe the DC motor as you push the tact switch.

B. Variable speed control

1. Encode the sketch below and upload it to the Arduino board:

```

//analog input pin
int LDRPin = A1;
//output pins used to control the DC motor
int motorPin1 = 5, motorPin2 = 6, motorEnable = 9;
//stores PWM value used to control speed of the motor
int motorSpeed;
//used for calibration
int sensorMin, sensorMax, sensorReading;

void setup()
{
    //initialize I/O pins
    pinMode(motorPin1, OUTPUT);
    pinMode(motorPin2, OUTPUT);
    pinMode(motorEnable, OUTPUT);
    pinMode(13, OUTPUT);
    //signal start of calibration
    digitalWrite(13, HIGH);

    //calibrate during the first five seconds
    while (millis() < 5000)
    {

```

```

        sensorReading = analogRead(LDRPin);
        if (sensorReading > sensorMax)
        {
            sensorMax = sensorReading;
        }

        if (sensorReading < sensorMin)
        {
            sensorMin = sensorReading;
        }
    }

    //signal the end of calibration
    digitalWrite(13, LOW); //set the pin13 to LOW

    //run the DC motor
    digitalWrite(motorPin1, HIGH);
    digitalWrite(motorPin2, LOW);
}

void loop()
{
    //read LDR value
    int x = analogRead(LDRPin);

    //map to valid PWM output range
    motorSpeed = map(x, sensorMin, sensorMax, 0, 255);

    //output PWM signal
    analogWrite(motorEnable, motorSpeed);

    //ADC settling time delay
    delay(15);
}

```

2. Connect pins A1, 5, 6 and 9 of the Gizduino to pins LDR, DC_MOTOR_IN1, DC_MOTOR_IN2 and DC_MOTOR_EN1 of the trainer board respectively.
3. Connect the pins DC_MOTOR_OUT1 and DC_MOTOR_OUT2 of the trainer board to the supply terminals of the DC motor.
4. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
5. Connect the 9 V DC adaptor to the DC jack in the trainer board.
6. Vary the light intensity around the LDR then observe the behavior of the DC motor.

Part II. Servo Motor Control

A. Sweep

1. Encode the program below and upload it to the Arduino board:

```
#include <Servo.h>

//instantiate a Servo object
Servo myservo;

//variable to store the servo position
int pos = 0;
void setup()
{
    //attaches the servo on pin 9 to the Servo object
    myservo.attach(9);
}

void loop()
{
    //go from 0 degrees to 180 degrees
    //in steps of 1 degree
    for(pos = 0; pos < 180; pos += 1)
    {
        //tell servo to go to position in variable 'pos'
        myservo.write(pos);
        //waits 15ms for the servo to reach the position
        delay(15);
    }
    //go from 180 degrees to 0 degrees
    //in steps of 1 degree
    for(pos = 180; pos>=1; pos-=1)
    {
        myservo.write(pos);
        delay(15);
    }
}
```

2. Connect pin 9 of the Gizduino to the SERVO_PULSE pin in the trainer board.
3. Connect the 3-pin female connector of the servo motor to HEADER3 in the trainer board.

NOTE: Never reverse the connection of the servo motor to HEADER3 (refer to the appropriate connection of the servo motor to HEADER3 in Figure 5.1).

4. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.

5. Observe the servo motor.

B. Position control

1. Encode the sketch below and upload it to the Arduino board:

```
#include <Servo.h>

//create a Servo object to control a servo motor
Servo myservo;

//indicates position of the servo
int dir1=LOW, dir2=LOW;

void setup()
{
    //initialize I/O pins
    pinMode(2, INPUT);
    pinMode(3, INPUT);

    //attaches the servo on pin 9 to the Servo object
    myservo.attach(9);
    //initialize servo to 0 degrees
    myservo.write(0);
}

void loop()
{
    //call the function posControl1 when signal received
    //from pin 2 changes from LOW to HIGH

    attachInterrupt(0, posControl1, RISING);

    //call the function posControl2 when signal received
    //from pin 3 changes from LOW to HIGH
    attachInterrupt(1, posControl2, RISING);
}

void posControl1()
{
    dir1 = !dir1; //toggle dir1
    if (dir1 == HIGH)
    {
        myservo.write(90); //position servo to 90 degrees
    }
    else
    {

```

```

        myservo.write(180); //position servo to 180 degrees
    }
}

void posControl2()
{
    dir2 = !dir2; //toggle dir2
    if (dir2 == HIGH)
    {
        myservo.write(45); //position servo to 45 degrees
    }
    else
    {
        myservo.write(135); //position servo to 135 degrees
    }
}

```

2. Connect pins 2, 3 and 9 of the Gizduino to two tact switches and the SERVO_PULSE pin in the trainer board respectively.
 3. Connect the 3-pin female connector of the servo motor to HEADER3 in the trainer board.
- NOTE:** Never reverse the connection of the servo motor to HEADER3 (refer to the appropriate connection of the servo motor to HEADER3 in Figure 5.1).
4. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
 5. Push the tact switches that you connected to the Gizduino. Observe the behavior of the servo motor.

Part III. Stepper motor

A. Single stepping mode

1. Encode the sketch below and upload it to the Arduino board:

```

//stepper motor pins
int motorPin1 = 8;
int motorPin2 = 9;
int motorPin3 = 10;
int motorPin4 = 11;

//time delay for
int delayTime = 15;

void setup()

```

```

{
    //initialize I/O pins
    pinMode(motorPin1, OUTPUT);
    pinMode(motorPin2, OUTPUT);
    pinMode(motorPin3, OUTPUT);
    pinMode(motorPin4, OUTPUT);
}
void loop()
{
    for (i = 0; i <13; i++)
    {
        cw();
    }
    delay(2000);
    for (i = 0; i <13; i++)
    {
        cw();
    }
    delay(2000);

    for (i = 0; i < 13; i++)
    {
        ccw();
    }
    delay(2000);
    for (i = 0; i < 13; i++)
    {
        ccw();
    }
    delay(2000);
    for (i = 0; i < 13; i++)
    {
        ccw();
    }
    delay(2000);
}

void cw()
{
    digitalWrite(motorPin1, HIGH);
    digitalWrite(motorPin2, LOW);
    digitalWrite(motorPin3, LOW);
    digitalWrite(motorPin4, LOW);
    delay(delayTime);
    digitalWrite(motorPin1, LOW);
    digitalWrite(motorPin2, HIGH);
    digitalWrite(motorPin3, LOW);
    digitalWrite(motorPin4, LOW);
    delay(delayTime);
    digitalWrite(motorPin1, LOW);
    digitalWrite(motorPin2, LOW);
    digitalWrite(motorPin3, HIGH);

```

```

        digitalWrite(motorPin4, LOW);
        delay(delayTime);
        digitalWrite(motorPin1, LOW);
        digitalWrite(motorPin2, LOW);
        digitalWrite(motorPin3, LOW);
        digitalWrite(motorPin4, HIGH);
        delay(delayTime);
    }

    void ccw()
    {
        digitalWrite(motorPin1, LOW);
        digitalWrite(motorPin2, LOW);
        digitalWrite(motorPin3, LOW);
        digitalWrite(motorPin4, HIGH);
        delay(delayTime);
        digitalWrite(motorPin1, LOW);
        digitalWrite(motorPin2, LOW);
        digitalWrite(motorPin3, HIGH);
        digitalWrite(motorPin4, LOW);
        delay(delayTime);
        digitalWrite(motorPin1, LOW);
        digitalWrite(motorPin2, HIGH);
        digitalWrite(motorPin3, LOW);
        digitalWrite(motorPin4, LOW);
        delay(delayTime);
        digitalWrite(motorPin1, HIGH);
        digitalWrite(motorPin2, LOW);
        digitalWrite(motorPin3, LOW);
        digitalWrite(motorPin4, LOW);
        delay(delayTime);
    }

```

2. Connect pins 8-11 of the Gizduino to pins STEPPER_IN1 to STEPPER_IN4 in the trainer board respectively.
3. Connect the 4-pin male connector of stepper motor to pins STEPPER_OUT1 to STEPPER_OUT4 of the trainer board. Also, connect the other 2-pin male connector of the stepper motor to two 5V_REG pins in the trainer board.
4. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
5. Connect the 9 V DC adaptor to the DC jack in the trainer board.
6. Observe the behavior of the stepper motor.

B. Half stepping mode

1. Encode the program below and upload it to the Arduino board:

```
int A=8, B=9, _A=10, _B=11;
int cwpin=2, ccwpin=3, LDRpin=A0;
int deg, reading, cycles;
int sensorMin, sensorMax;
int delayTime = 10;

void setup()
{
  //initialize I/O pins
  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(_A, OUTPUT);
  pinMode(_B, OUTPUT);
  pinMode(cwpin, INPUT);
  pinMode(ccwpin, INPUT);

  Serial.begin(9600);
  //signal start of calibration
  digitalWrite(13,HIGH);
  //calibrate for 5 s
  while (millis() < 5000)
  {
    Serial.println("Calibrating sensor...");
    reading = analogRead(LDRpin);
    if (reading > sensorMax)
    {
      sensorMax = reading;
    }
    if (reading < sensorMin)
    {
      sensorMin = reading;
    }
  }
  //signal end of calibration
  digitalWrite(13,LOW);
}

void loop()
{
  reading = analogRead(LDRpin);
  deg = map(reading, sensorMin, sensorMax, 0, 360);
  Serial.print("Angle = ");
  Serial.print(deg);
  Serial.println("");
  attachInterrupt(0, cw, RISING);
  attachInterrupt(1, ccw, RISING);
  delay(10);
}
```

```

}

void cw()
{
    cycles = deg/3.6;
    for (int i=0; i <= cycles; i++)
    {
        digitalWrite(A, HIGH);
        digitalWrite(B, LOW);
        digitalWrite(_A, LOW);
        digitalWrite(_B, LOW);
        delay(delayTime);
        digitalWrite(A, HIGH);
        digitalWrite(B, HIGH);
        digitalWrite(_A, LOW);
        digitalWrite(_B, LOW);
        delay(delayTime);
        digitalWrite(A, LOW);
        digitalWrite(B, HIGH);
        digitalWrite(_A, LOW);
        digitalWrite(_B, LOW);
        delay(delayTime);
        digitalWrite(A, LOW);
        digitalWrite(B, HIGH);
        digitalWrite(_A, HIGH);
        digitalWrite(_B, LOW);
        delay(delayTime);
        digitalWrite(A, LOW);
        digitalWrite(B, LOW);
        digitalWrite(_A, HIGH);
        digitalWrite(_B, LOW);
        delay(delayTime);
        digitalWrite(A, LOW);
        digitalWrite(B, LOW);
        digitalWrite(_A, HIGH);
        digitalWrite(_B, HIGH);
        delay(delayTime);
        digitalWrite(A, LOW);
        digitalWrite(B, LOW);
        digitalWrite(_A, LOW);
        digitalWrite(_B, HIGH);
        delay(delayTime);
        digitalWrite(A, HIGH);
        digitalWrite(B, LOW);
        digitalWrite(_A, LOW);
        digitalWrite(_B, HIGH);
        delay(delayTime);
    }
}

void ccw()
{

```

```

cycles = deg/3.6;
for (int j=0; j <= cycles; j++) {
    digitalWrite(_B, HIGH);
    digitalWrite(_A, LOW);
    digitalWrite(B, LOW);
    digitalWrite(A, LOW);
    delay(delayTime);
    digitalWrite(_B, HIGH);
    digitalWrite(_A, HIGH);
    digitalWrite(B, LOW);
    digitalWrite(A, LOW);
    delay(delayTime);
    digitalWrite(_B, LOW);
    digitalWrite(_A, HIGH);
    digitalWrite(B, LOW);
    digitalWrite(A, LOW);
    delay(delayTime);
    digitalWrite(_B, LOW);
    digitalWrite(_A, HIGH);
    digitalWrite(B, HIGH);
    digitalWrite(A, LOW);
    delay(delayTime);
    digitalWrite(_B, LOW);
    digitalWrite(_A, LOW);
    digitalWrite(B, HIGH);
    digitalWrite(A, LOW);
    delay(delayTime);
    digitalWrite(_B, LOW);
    digitalWrite(_A, LOW);
    digitalWrite(B, HIGH);
    digitalWrite(A, HIGH);
    delay(delayTime);
    digitalWrite(_B, LOW);
    digitalWrite(_A, LOW);
    digitalWrite(B, LOW);
    digitalWrite(A, HIGH);
    delay(delayTime);
    digitalWrite(_B, HIGH);
    digitalWrite(_A, LOW);
    digitalWrite(B, LOW);
    digitalWrite(A, HIGH);
    delay(delayTime);
}
}

```

2. Connect pins 8-11 of the Gizduino to pins STEPPER_IN1 to STEPPER_IN4 in the trainer board respectively. Also, connect pins A0, 2 and 3 to the LDR and two tact switches on the trainer board respectively.
3. Connect the 4-pin male connector of stepper motor to pins STEPPER_OUT1 to STEPPER_OUT4 of the trainer board. Also, connect

the other 2-pin male connector of the stepper motor to two 5V_REG pins in the trainer board.

4. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
5. Connect the 9 V DC adaptor to the DC jack in the trainer board.
6. Vary the light intensity around the LDR and push the tact switches that you connected to the Gizduino. Observe the behavior of the stepper motor.

Activities

- a. Create a program that controls the speed and direction of rotation of a DC motor. Use a tact switch to toggle between the forward and reverse rotation of the DC motor and an LDR to vary its speed. Use an RG LED to indicate its current direction of rotation.
- b. Using a LDR as an input, create a program that controls the position of the servo motor. If the ambient light is “dark”, turn the servo motor at 0 degrees; “medium dark”, 45 degrees; “medium”, 90 degrees; “medium bright”, 135 degrees; and “bright”, at 180 degrees. Initially perform a 5-second calibration to provide accurate readings. During this time the servo should sweep to indicate that the sensor is in calibration period. Print the detected ambient light intensity and servo’s angle to the Serial Monitor.
- c. Another stepping mode used to drive stepper motors is the high torque stepping. Create a program that utilizes this mode for a stepper motor. Use a tact switch to toggle the stepper’s direction of rotation (clockwise or counterclockwise) and another 3 tact switches to set the position of the stepper motor to 60, 120, and 180 degrees respectively.

References

<http://www.arduino.cc/en/Reference/Servo>
www.ehow.com/servos/
<http://www.tigoe.net/pcomp/code/circuits/motors/stepper-motors>
<http://digital.ni.com/public.nsf/allkb/6C55312B6F61C1D686256C6200730BBD>
<http://www.solarbotics.net/library/pdf/lib/pdf/motorbas.pdf>
<http://www.arduino.cc/en/Reference/Interrupts>

Experiment No. 5
SERIAL LCD DISPLAY

Objectives

At the end of the experiment the student should be able to:

1. Demonstrate how to interface a serial LCD display to an Arduino board.
2. Display characters on a serial LCD.
3. Control and format output on a serial LCD.

Equipment and Materials

- 1 pc Gizduino
- 1 pc Trainer Board 01
- 1 pc USB Standard A-B Cable
- 1 set Connecting Wires
- 1 set Desktop computer

Discussion

LCD

Liquid Crystal Display (LCD) uses liquid crystal fluid as the active medium to create images. As shown in figure 5.1, a very large number of thin crystals are suspended in a fluid. The liquid reservoir is sandwiched between two thin layers of glass. Each layer of glass has a transparent conductive electrode bonded to it.

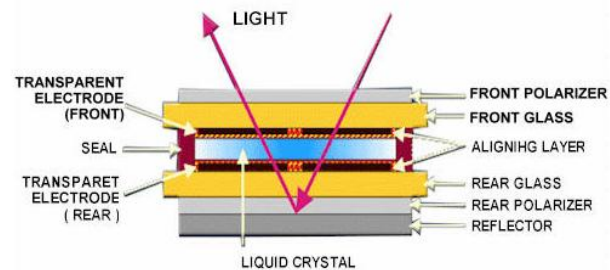


Figure 5.1. Liquid crystal.

Polarizing material are placed on the outside surface of both front and rear glasses. The top polarizer can polarize the incident light of random polarization into its polarization direction. Before an electric field is applied on the conductors, the crystals are aligned in a spiral pattern. The light is then changed by the spiral pattern of the crystals. The bottom polarizer is aligned opposite of the top polarizer. When the light reaches the bottom polarizer, they will align with each other. The light can pass through and as shown in Figure 5.2, the LCD looks transparent.

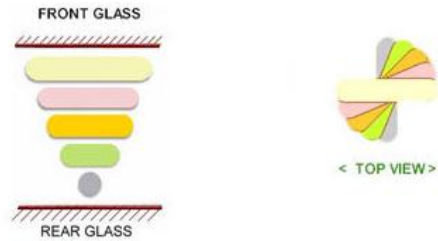


Figure 5.2. Spiral pattern alignment of crystals.

When in an electric field, the crystals align themselves with the field so the light can pass through the crystal without being deflected, as shown in Figure 5.3. The light is then out of phase with the bottom polarizer when the light is absorbed by the polarizer. The light can't pass through and the LCD looks dark or opaque. By turning on/off electric field, we can create desired display patterns.

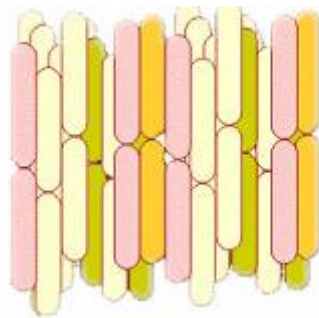


Figure 5.3. Crystals aligned with the field.

Hitachi HD44780 driver



Figure 5.4. 2x16 LCD Hitachi
HD44780 driver

An HD44780 Character LCD is a de facto industry standard liquid crystal display (LCD) display device designed for interfacing with embedded systems. These screens come in a variety of configurations including 8x1, which is one row of eight characters, 16x2, and 20x4. The most commonly manufactured configuration is 40x4 characters, which requires two individually addressable HD44780 controllers with expansion chips as the HD44780 can only address up to 80 characters. (see Figure 5.4)

The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

A **register select (RS)** pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

A **Read/Write (R/W)** pin that selects reading mode or writing mode

An **Enable pin** that enables writing to the registers

8 data pins (D0 -D7). The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.

There's also a **display constrast pin (Vo)**, **power supply pins (+5V and Gnd)** and **LED Backlight (Bklt+ and Bklt-)** pins that you can use to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.

The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register.

The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit. The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins.

LiquidCrystal Library (LiquidCrystal.h)

This library allows an Arduino board to control LiquidCrystal displays (LCDs) based on the Hitachi HD44780 (or a compatible) chipset, which is found on most text-based LCDs. The library works with in either 4- or 8-bit mode (i.e. using 4 or 8 data lines in addition to the rs, enable, and, optionally, the rw control lines).

- `LiquidCrystal(rs,enable,d4,d5,d6,d7)` - Creates a variable of type `LiquidCrystal`. The display can be controlled using 4 or 8 data lines. If the former, omit the pin numbers for d0 to d3 and leave those lines unconnected. The RW pin can be tied to ground instead of connected to a pin on the Arduino; if so, omit it from this function's parameters.
- `begin(cols,rows)` – specifies the dimensions (width and height) of the display.
- `clear()` - Clears the LCD screen and positions the cursor in the upper-left corner.

- `home()` - Positions the cursor in the upper-left of the LCD. That is, use that location in outputting subsequent text to the display. To also clear the display, use the `clear()` function instead.
- `setCursor(col,row)` - Position the LCD cursor; that is, set the location at which subsequent text written to the LCD will be displayed.
- `write(data)` - Write a character to the LCD.
- `print(data)` - Prints text to the LCD.
- `cursor()` - Display the LCD cursor: an underscore (line) at the position to which the next character will be written.
- `noCursor()` - Hides the LCD cursor.
- `blink()` - Display the blinking LCD cursor. If used in combination with the `cursor()` function, the result will depend on the particular display.
- `noBlink()` - Turns off the blinking LCD cursor.
- `display()` - Turns on the LCD display, after it's been turned off with `noDisplay()`. This will restore the text (and cursor) that was on the display.
- `noDisplay()` - Turns off the LCD display, without losing the text currently shown on it.
- `scrollDisplayLeft()` - Scrolls the contents of the display (text and cursor) one space to the left.
- `scrollDisplayRight()` - Scrolls the contents of the display (text and cursor) one space to the right.
- `autoscroll()` - Turns on automatic scrolling of the LCD. This causes each character output to the display to push previous characters over by one space. If the current text direction is left-to-right (the default), the display scrolls to the left; if the current direction is right-to-left, the display scrolls to the right. This has the effect of outputting each new character to the same location on the LCD.
- `noAutoscroll()` - Turns off automatic scrolling of the LCD.
- `leftToRight()` - Set the direction for text written to the LCD to left-to-right, the default. This means that subsequent characters written to the display will go from left to right, but does not affect previously-output text.

- `rightToLeft()` - Set the direction for text written to the LCD to right-to-left (the default is left-to-right). This means that subsequent characters written to the display will go from right to left, but does not affect previously-output text.
- `createChar(num,data)` - Create a custom character (glyph) for use on the LCD. Up to eight characters of 5x8 pixels are supported (numbered 0 to 7). The appearance of each custom character is specified by an array of eight bytes, one for each row. The five least significant bits of each byte determine the pixels in that row. To display a custom character on the screen, `write()` its number.

Trainer Board Schematic Diagram

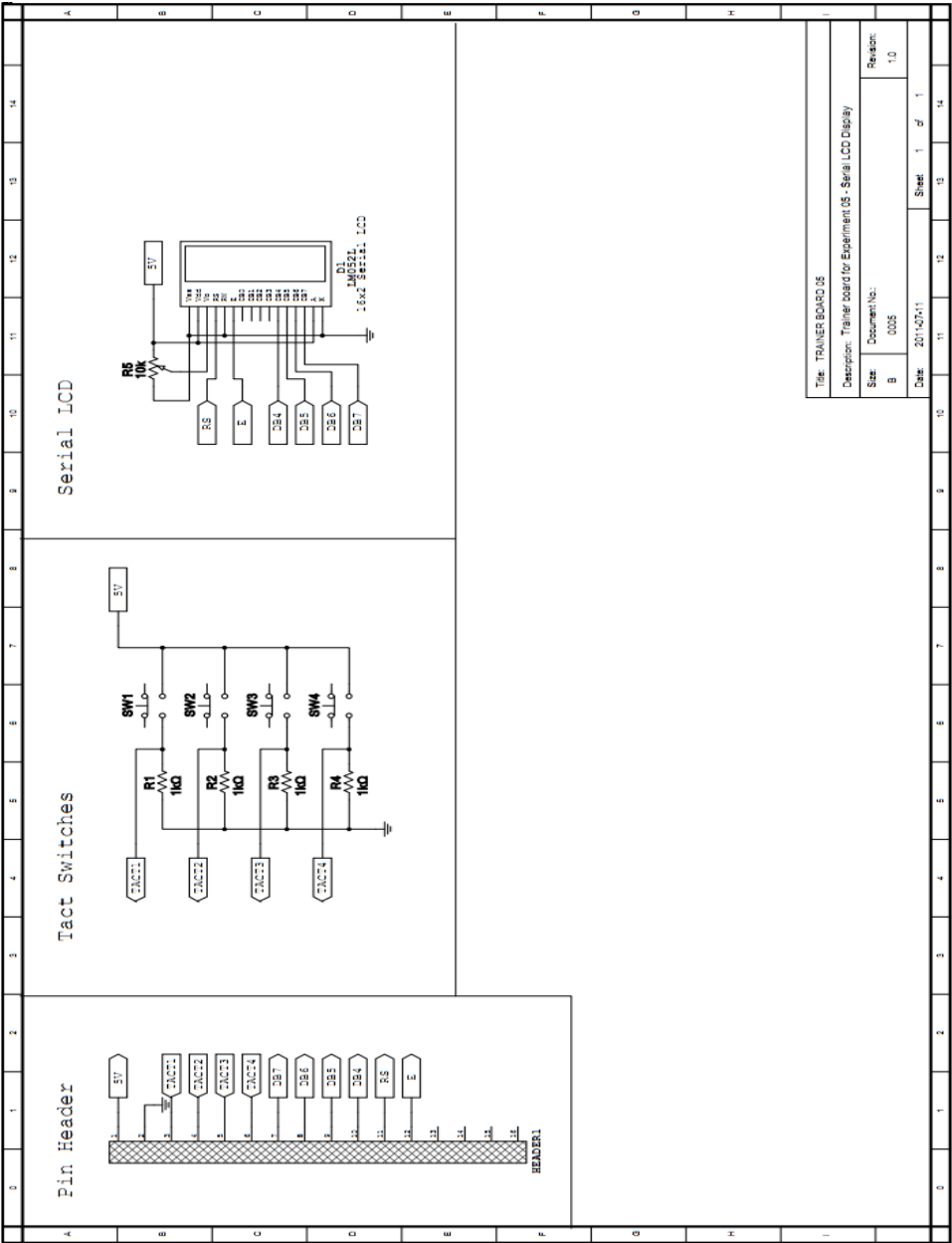


Figure 5.1. Trainer Board 05

Procedures

Part I. Basic LCD Commands

A. Display the message “Hello World” and a blinking cursor

1. Encode the program below and upload it to the Arduino board:

```
//include the library code
#include <LiquidCrystal.h>

//initialize the library with the numbers of the
//interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

void setup()
{
    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
    // Print a message to the LCD.
    lcd.print("HELLO WORLD!");
}

void loop()
{
    //Turn off the cursor
    lcd.noBlink();
    delay(3000);
    //Turn on the cursor
    lcd.blink();
    delay(3000);
}
```

2. Connect pins 7-12 of the Gizduino to pins DB7, DB6, DB5, DB4, RS, and EN of the trainer board respectively (refer to Figure 5.1).
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Adjust the potentiometer until you see the output on the LCD. Observe the message displayed in the display.

B.

1. Encode the program below and upload it to the Arduino board:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
char thisChar;
int col, row;

void setup()
{
    lcd.begin(16, 2);
}

void loop()
{
    lcd.clear();
    col=0; row=0; thisChar = 'a';
    for ( ; thisChar <= 'z'; thisChar++)
    {
        lcd.setCursor(col, row);
        lcd.cursor();
        delay(1000);

        lcd.write(thisChar);
        lcd.noCursor();
        delay(1000);

        if (row != 1)
        {
            row = 1;
        }
        else
        {
            col++; row=0;
        }
    }
}
```

2. Connect pins 7-12 of the Gizduino to pins DB7, DB6, DB5, DB4, RS, and EN of the trainer board respectively.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Observe the LCD output.

C.

1. Encode the program below and upload it to the Arduino board:

```
#include <LiquidCrystal.h>

//initialize the library with the numbers of the
//interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

void setup()
{
    //set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
    //print a message to the LCD.
    lcd.setCursor(0,1);
    lcd.print("* HELLO WORLD! *");
    lcd.home();
    lcd.print("*****");
}

void loop()
{
    //turn off the display
    lcd.noDisplay();
    delay(500);
    //turn on the display
    lcd.display();
    delay(500);
}
```

2. Connect pins 7-12 of the Gizduino to pins DB7, DB6, DB5, DB4, RS, and EN of the trainer board respectively.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Observe the LCD output.

Part II. Advanced LCD Commands

A.

1. Encode the program below and upload it to the Arduino board:

```
// include the library code:
```

```

#include <LiquidCrystal.h>

// initialize the library with the numbers of the
// interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
int posctr;

void setup()
{
    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
    // Print a message to the LCD.
    lcd.setCursor(4, 0);
    lcd.print("ARDUINO");
    lcd.setCursor(3, 1);
    lcd.print("DIECIMILA");
}

void loop() {
    delay(1000);
    for(posctr=0; posctr<14; posctr++)
    {
        lcd.scrollDisplayRight();
        delay(150);
    }
    for(posctr=0; posctr<30; posctr++)
    {
        lcd.scrollDisplayLeft();
        delay(150);
    }
    for(posctr=0; posctr<16; posctr++)
    {
        lcd.scrollDisplayRight();
        delay(150);
    }
}

```

2. Connect pins 7-12 of the Gizduino to pins DB7, DB6, DB5, DB4, RS, and EN of the trainer board respectively.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Observe the LCD output.

B.

1. Encode the program below and upload it to the Arduino board:

```
// include the library code:
#include <LiquidCrystal.h>
char letter;
// initialize the library with the numbers of the
interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
int dir=HIGH;

void setup() {
    // set up the LCD's number of columns and rows:
    lcd.begin(16,2);
    lcd.cursor();
}

void loop() {
    lcd.clear();

    if (dir == HIGH)
    {
        lcd.rightToLeft();
        lcd.setCursor(15, 0);
    }
    else
    {
        lcd.leftToRight();
        lcd.home();
    }

    // print from 0 to 9:
    for (letter = 'a'; letter <= 'z' ; letter++)
    {
        if (letter == 'q')
        {
            lcd.autoscroll();
            lcd.cursor();
        }
        lcd.print(letter);
        delay(500);
    }

    lcd.noAutoscroll();
```

```

    // dir = !dir;
    // clear screen for the next loop:

}

```

2. Connect pins 7-12 of the Gizduino to pins DB7, DB6, DB5, DB4, RS, and EN of the trainer board respectively.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Observe the LCD output.

C.

1. Encode the program below and upload it to the Arduino board:

```

#include <LiquidCrystal.h>

LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
int col=0,row=0,col1=0,row1=0, num;

byte ghost[8] = {
    B00100,
    B01110,
    B10101,
    B11111,
    B11111,
    B10101,
    B10101,
};

byte pacman_c[8] = {
    B00100,
    B01010,
    B11111,
    B11111,
    B11111,
    B11110,
    B01100,
};

byte pacman_or[8] = {
    B00110,
    B01011,
    B11100,
    B11000,
};

```

```

        B11100,
        B11111,
        B01110,
    };

    byte pacman_ol[8] = {
        B01100,
        B11010,
        B01111,
        B00111,
        B00011,
        B11111,
        B01110,
    };

    void setup() {
        lcd.createChar(0, ghost);
        lcd.createChar(1, pacman_c);
        lcd.createChar(2, pacman_or);
        lcd.createChar(3, pacman_ol);
        pinMode(2, OUTPUT);
        pinMode(3, OUTPUT);
        lcd.begin(16, 2);
        // lcd.write(0);
        // lcd.write(1);
        // lcd.write(2);
    }

    void loop()
    {
        lcd.write(
            attachInterrupt(0, left, RISING);
            attachInterrupt(1, right, RISING);
        )

    void left()
    {
        if (col < 16)
        {
            setCursor(col++, row);
            num
        }
    }

    void right()
    {

```

```

        if (col < 16)
        {
            setCursor(col++, row);
            num = 2;
        }
    }
}

```

2. Connect pins 7-12 of the Gizduino to pins DB7, DB6, DB5, DB4, RS, and EN of the trainer board respectively.
3. Connect the supply pins of the trainer board to +5V and GND pins of Gizduino.
4. Observe the LCD output.

Activities

- a. Create a sketch that displays “HELLO WORLD” when a tact switch is pressed and “WORLD HELLO” when another tact switch is pressed. The initial display for the LCD should be blank.
- b. Create a sketch that displays the numbers 1 – 10 on the LCD. The numbers should be displayed one at a time and should be one space apart from each other. Use only the first row of the LCD. Scroll the contents of the display if necessary.
- c. Create
a sketch that displays your own character. Animate your character by letting it move throughout the display. Define your own animation sequences.
- d. Create a sketch that displays the number “1” on the bottom right of the LCD display. Use two tact switches for incrementing and decrementing the number displayed. Use another two tact switches for setting the position of the displayed number at the bottom right or the top left of the LCD respectively.

References

http://www.deepakshionline.com/technical_lcd.html
http://en.wikipedia.org/wiki/HD44780_Character_LCD

Experiment No. 6

WIRELESS COMMUNICATION

Objectives

At the end of the experiment the student should be able to:

1. Program two XBee modules for point-to-point communication.
2. Interface an XBee module to an Arduino board.
3. Read input and control output devices wirelessly using XBee modules.

Equipment and Materials

2 pcs	Gizduino
1 pc	Trainer Board 01
1 pc	Trainer Board 03
2 pcs	Trainer Board 06
1 pc	USB Standard A-B Cable
1 pc	RS232 cable
1 set	Connecting Wires
1 set	Desktop computer

Discussion

ZigBee® Wireless Standard

ZigBee is a wireless technology developed as an open global standard to address the unique needs of low-cost, low-power wireless machine-to-machine (M2M) networks. The ZigBee standard operates on the IEEE 802.15.4-2003 standard for Low-Rate Wireless Personal Area Networks (LR-WPANs), such as wireless light switches with lamps, electrical meters with in-home-displays, consumer electronics equipment via short-range radio needing low rates of data transfer. The technology defined by the ZigBee specification is intended to be simpler and less expensive than other WPANs, such as Bluetooth. ZigBee is targeted at radio-frequency (RF) applications that require a low data rate, long battery life, and secure networking.

XBee®

XBee is the brand name from Digi International for a family of form factor compatible radio modules. XBee modules are designed to operate within the ZigBee protocol and support the unique needs of low-cost, low-power wireless sensor networks. The

modules require minimal power and provide reliable delivery of data between remote devices.



Figure 6.1. XBee and XBee-PRO ZB RF modules.
(Photo courtesy: Digi International)

Programming XBee Modules Using X-CTU

X-CTU is a Windows-based application provided by Digi. This program was designed to interact with the firmware files found on Digi's RF products and to provide a simple-to-use graphical user interface to them.

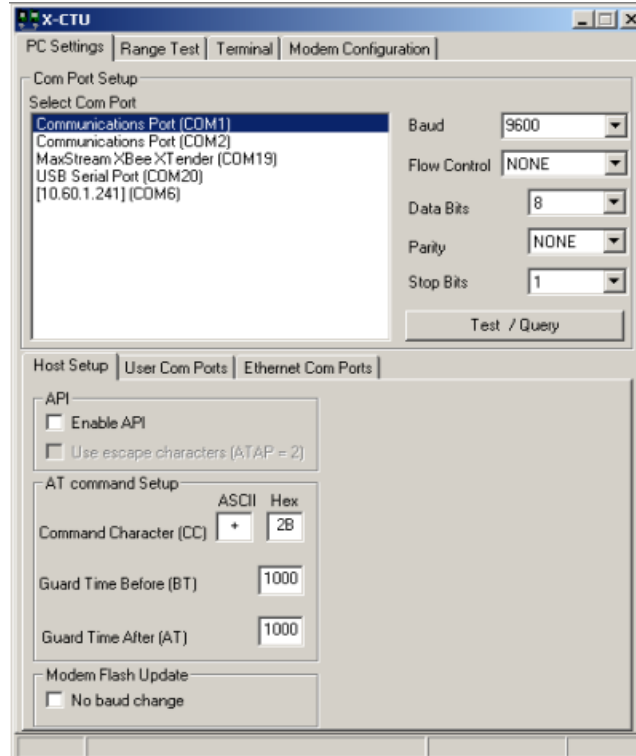


Figure 6.2. X-CTU.

When launched, you will see four tabs across the top of the program (see Figure 6.2). Each of these tabs has a different function. The four tabs are:

PC Settings. Allows a user to select the desired COM port and configure that port to fit the radios settings.

Range Test. Allows a user to perform a range test between two radios.

Terminal. Allows access to the computers COM port with a terminal emulation program. This tab also allows the ability to access the radios' firmware using AT commands (for a complete listing of the radios' AT commands, please see the product manuals available online).

Modem Configuration. Allows the user to program the radios' firmware settings via a graphical user interface. This tab also allows customers the ability to change firmware versions.

Trainer Board Schematic Diagram

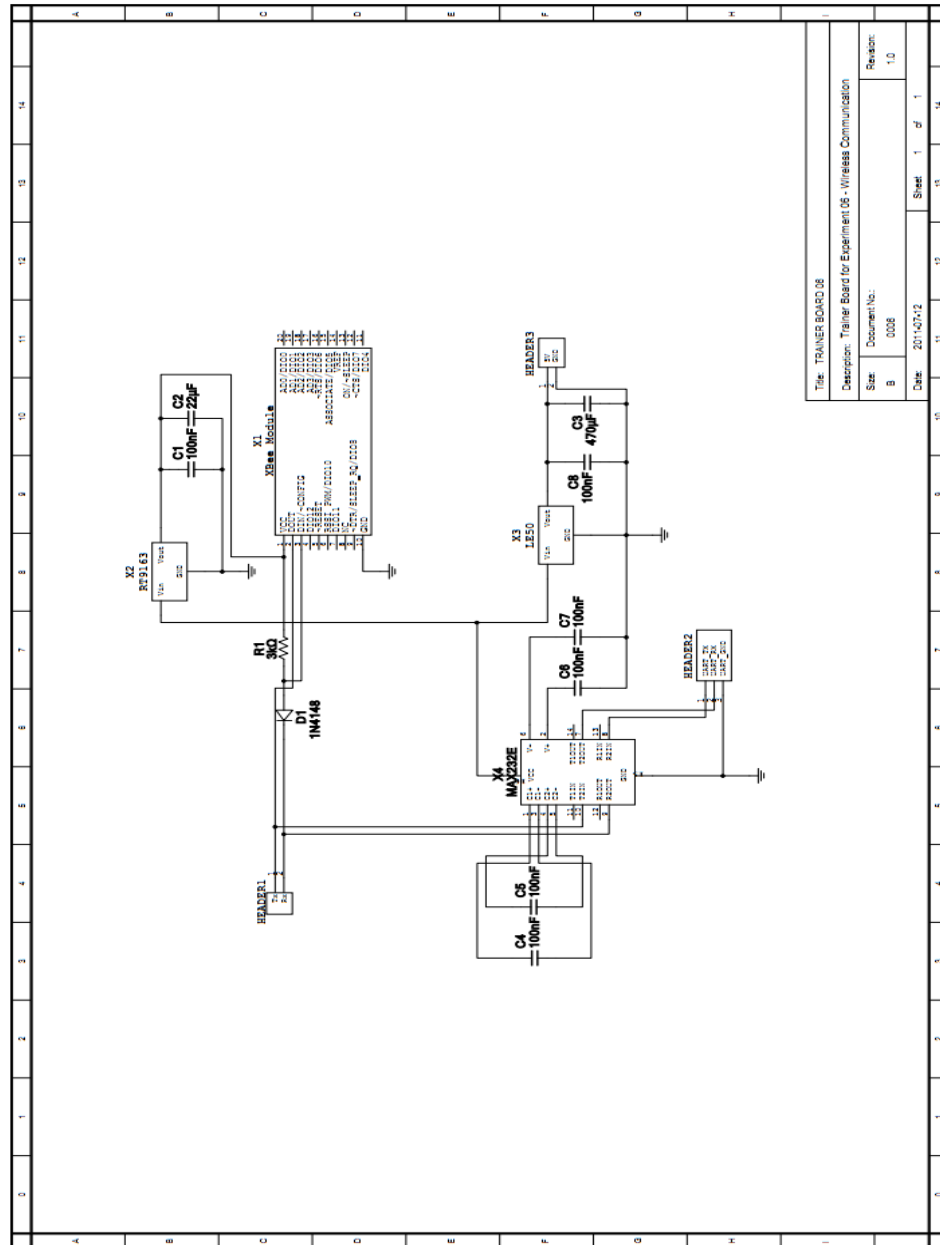


Figure 6.3. Trainer Board 06

Procedures

Part I. Configure XBee point-to-point communication

1. Connect one Trainer Board 06 to your PC using a RS232 cable. This board shall serve as the interface for the Gizduino on the transmitter side.
2. Open X-CTU. Select the COM port where the Trainer Board 06 is connected. Set the baud, flow control, data bits, parity, and stop bits to the values shown in Figure 6.4.

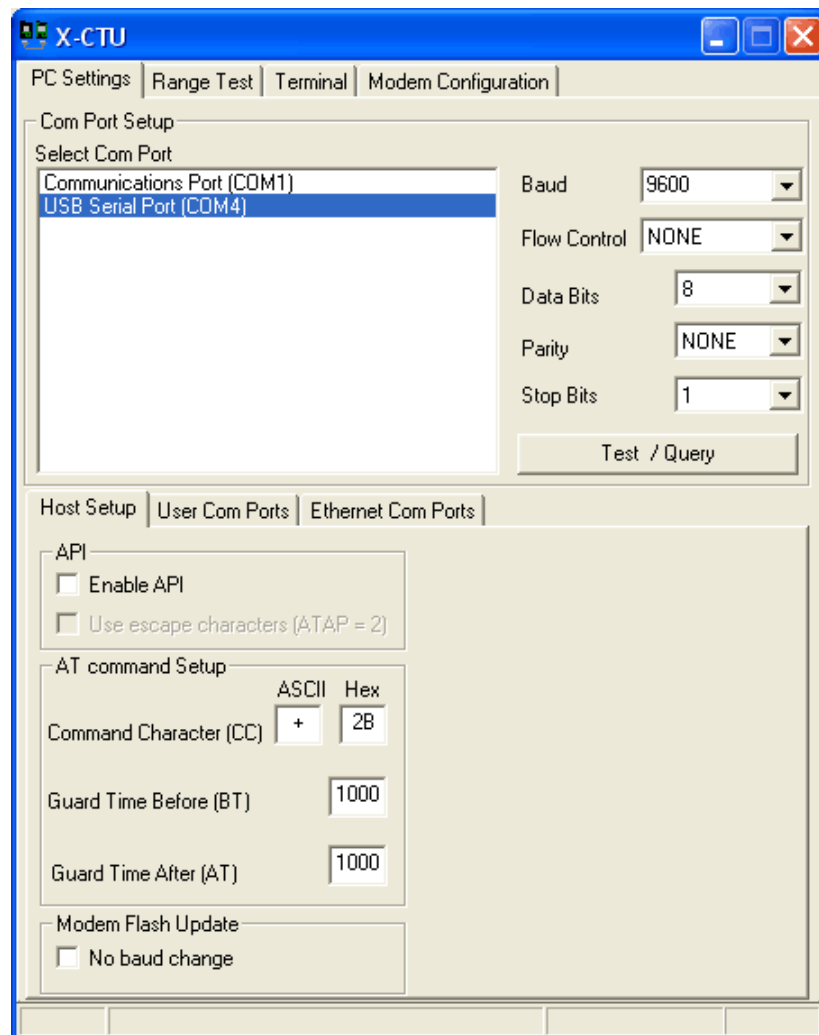


Figure 6.4. X-CTU PC Settings Tab.

3. Click **Test / Query**. This tests the modem for communication and returns the modem's type and firmware version if the test is successful (see Figure 6.5). If the test is unsuccessful, check that the XBee is powered, wired correctly, and the right communication settings are set.

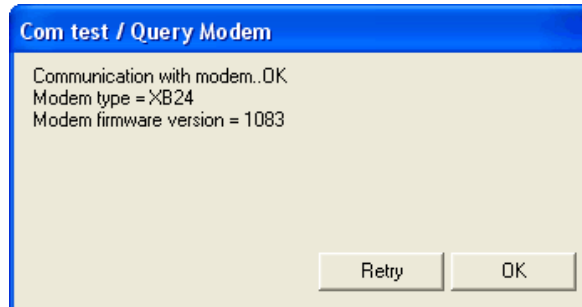


Figure 6.5. COM test / Query Modem dialog box.

4. Go to the Modem Configuration tab. Click **Read** under **Modem Parameters and Firmware** to read in the current version and settings of the XBee.
5. Set the 16-bit source address (**MY**) and the destination address low address (**DL**) to the values assigned to you by your instructor (see Figure 6.6). Then click **Write** under **Modem Parameters and Firmware** to upload the firmware to the XBee.

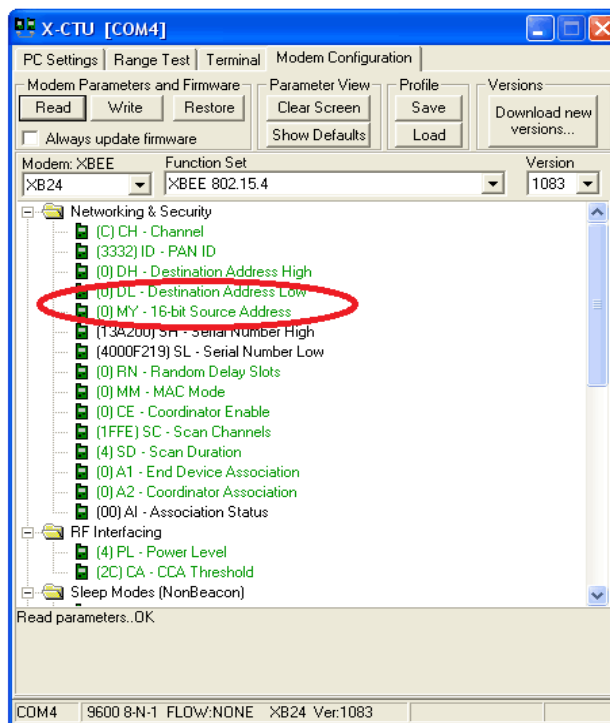


Figure 6.6. X-CTU Modem Configuration Tab.

6. Repeat the steps 1 to 5 for the other Trainer Board 06. This time, set **MY** and **DL** of the receiver side Trainer Board 06 to the values of **DL** and **MY** of the transmitter side Trainer Board 06 respectively.

Part II. Wireless data transmission and reception

A. Read tact switch input and control LED output

1. Encode the sketch below and upload it to the transmitter side Arduino board:

```
//transmitter side
int tackswitch = 9;
int tackstate = LOW;
int Ledpin = 13;
void setup()
{
    Serial.begin(9600);
    pinMode(tackswitch, INPUT);
    pinMode(Ledpin, OUTPUT);
    digitalWrite(Ledpin, LOW);
}

void loop()
{
    tackstate=digitalRead(tackswitch);
    if(tackstate==HIGH)
    {
        Serial.write('1');
        digitalWrite(Ledpin,HIGH);

        delay(500);
    }
    digitalWrite(Ledpin,LOW);
}
```

2. Encode the sketch below and upload it to the receiver side Arduino board:

```
//receiver side
int Ledpin = 13;
char data;

void setup()
{
    Serial.begin(9600);
    pinMode(Ledpin, OUTPUT);
    digitalWrite(Ledpin, LOW);
}
```

```

}

void loop()
{
    if(Serial.available() > 0)
    {
        data = Serial.read();
        Serial.print(data,BYTE);

        if(data == '1')
        {
            digitalWrite(Ledpin,HIGH);
            delay(500);
        }
        else
        {
            digitalWrite(Ledpin,LOW);
            delay(500);
        }
    }
    digitalWrite(Ledpin,LOW);
    delay(1000);
}

```

3. Connect pin 9 of the transmitter side Gizduino to a tact switch on Trainer Board 01. Also, connect pin 13 of the receiver side Gizduino to pin 4 of the Trainer Board 03.
4. Connect the Tx and Rx pins of the transmitter side Gizduino to the Rx and Tx pins of the transmitter side Trainer Board 06 respectively. Do the same for the receiver side Gizduino and receiver side trainer Board 06.
5. Connect the supply pins of the trainer boards to +5V and GND pins of the Gizduinos.
6. Press the tact switch that you connected on the transmitter side Gizduino. Observe the LED that you connected to the receiver side Gizduino as the tact switch is pressed.

B. Detect ambient light intensity using LDR

1. Encode the program below and upload it to the transmitter side Arduino board:

```

//transmitter side
const int sensorPin = A0;
int senMin;
int senMax;

```

```

int senReading;

void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);

  while (millis() < 5000)
  {
    senReading = analogRead(sensorPin);
    Serial.println("Calibrating sensor...");
    if (senReading > senMax)
    {
      senMax = senReading;
    }

    if (senReading < senMin)
    {
      senMin = senReading;
    }
  }

  //signal the end of the calibration period
  digitalWrite(13, LOW);
}

void loop()
{
  //read the sensor:
  senReading = analogRead(A0);
  //map the sensor range to a range of four options:
  int range = map(senReading, senMin, senMax, 0, 3);
  //do something different depending on the
  //range value:
  switch (range)
  {
    case 0: //your hand is on the sensor
      Serial.write('1'); //dark
      break;
    case 1: //your hand is close to the sensor
      Serial.write('2');//dim
      break;
    case 2: //your hand is a few inches from the
    sensor
      Serial.write('3'); //medium
      break;
  }
}

```

```

        case 3: //your hand is nowhere near the sensor
            Serial.write('4'); //bright
            break;
    }
}

```

2. Encode the program below and upload it to the receiver side Arduino board:

```

//receiver side
int Ledpin = 13;
char data;

void setup()
{
    Serial.begin(9600);
    pinMode(Ledpin, OUTPUT);
    digitalWrite(Ledpin, LOW);
}

void loop()
{
    if(Serial.available() > 0)
    {
        data = Serial.read();
        //Serial.print(data, BYTE);

        if(data == '1')
        {
            Serial.println("dark");
        }
        else if(data == '2')
        {
            Serial.println("dim");
        }
        else if(data == '3')
        {
            Serial.println("medium");
        }
        else if(data == '4')
        {
            Serial.println("bright");
        }
    }
}

```


3. Connect pin A0 of the transmitter side Gizduino to the LDR pin in the Trainer Board 03.
4. Connect the Tx and Rx pins of the transmitter side Gizduino to the Rx and Tx pins of the transmitter side Trainer Board 06 respectively. Do the same for the receiver side Gizduino and receiver side trainer Board 06.
5. Connect the supply pins of the trainer boards to +5V and GND pins of the Gizduinos.
6. Open Serial Monitor of the receiver side Gizduino. Try varying the light received by the LDR in Trainer Board 03 by covering it or shining a flashlight on it. Observe the messages displayed in the Serial Monitor.

C. Control LED output using a potentiometer

1. Encode the sketch below and upload it to the transmitter side Arduino board:

```
//transmitter side
int sensorPin = A0;
int sensorValue = 0;
int senMin;
int senMax;
void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);

  while (millis() < 5000)
  {
    sensorValue = analogRead(sensorPin);
    Serial.println("Calibrating sensor...");
    if (sensorValue > senMax)
    {
      senMax = sensorValue;
    }

    if (sensorValue < senMin)
    {
      senMin = sensorValue;
    }
  }

  //signal the end of the calibration period
  digitalWrite(13, LOW);
}
```

```

void loop()
{
  sensorValue=analogRead(sensorPin);
  int range = map(sensorValue, senMin, senMax, 0 , 3);
  switch(range)
  {
    case 0: //your hand is on the sensor
      Serial.write('0'); //dark
      break;
    case 1: //your hand is close to the sensor
      Serial.write('1');//dim
      break;
    case 2: //your hand is a few inches from the sensor
      Serial.write('2');//medium
      break;
    case 3: //your hand is nowhere near the sensor
      Serial.write('3');//bright
      break;

  }

}

```

2. Encode the sketch below and upload it to the receiver side Arduino board:

```

//receiver side
int ledPin1 = 13;
int ledPin2 = 12;
int ledPin3 = 11;
int ledPin4 = 10;
char data;
int ledState = LOW;
unsigned long currentTime = 0;
unsigned long previousTime = 0;
long interval = 250;

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(ledPin4, OUTPUT);
}

void loop()
{
  if( Serial.available() > 0)

```

```

{
    data = Serial.read();
    Serial.print(data, BYTE);

    if(data == '0')
    {
        digitalWrite(ledPin1, LOW);
        digitalWrite(ledPin2, LOW);
        digitalWrite(ledPin3, LOW);
        digitalWrite(ledPin4, HIGH);
        delay(5);
    }
    else if(data == '1')
    {
        digitalWrite(ledPin1, LOW);
        digitalWrite(ledPin2, LOW);
        digitalWrite(ledPin3, HIGH);
        digitalWrite(ledPin4, LOW);
        delay(5);
    }
    else if(data == '2')
    {
        digitalWrite(ledPin1, LOW);
        digitalWrite(ledPin2, HIGH);
        digitalWrite(ledPin3, LOW);
        digitalWrite(ledPin4, LOW);
        delay(5);
    }
    else if(data == '3')
    {
        digitalWrite(ledPin1, HIGH);
        digitalWrite(ledPin2, LOW);
        digitalWrite(ledPin3, LOW);
        digitalWrite(ledPin4, LOW);
        delay(5);
    }
}
}

```

3. Connect pin A0 of the transmitter side Gizduino to any potentiometer pin in the Trainer Board 03. Also, connect pin 13 of the receiver side Gizduino to a LED in Trainer Board 01.
4. Connect the Tx and Rx pins of the transmitter side Gizduino to the Rx and Tx pins of the transmitter side Trainer Board 06 respectively. Do the same for the receiver side Gizduino and receiver side trainer Board 06.

5. Rotate the potentiometer knob that you connected on the transmitter side Gizduino. Observe the LED output in Trainer Board 01 as the potentiometer knob is rotated.

Activities

- a. Create a sketch that controls an RGB LED wirelessly using three tact switches. The three tact switches should set three different colors of your choice for the RGB LED.
- b. Create a sketch which simulates a three-way handshake operation using a tact switch and LEDs. The tact switch on the transmitter side should toggle a LED on or off on the receiver side. Whenever the receiver side receives the signal sent by the tact switch from the transmitter side, it should reply back with a signal which causes a LED on the transmitter side to blink once for one second.
- c. Create a sketch that controls a 7-segment display wirelessly using a potentiometer. The analog input read from the potentiometer should be scaled from 0 to 9 and then displayed to the 7-segment display.

References

<http://en.wikipedia.org/wiki/XBee>

<http://www.digi.com/technology/rf-articles/wireless-zigbee.jsp>

APPENDIX B

Answer Key for Arduino Laboratory Manual Experiment 1

Guide Questions I.A

1. Describe the behavior of the LED that you connected to the Gizduino. What functions cause the LED to exhibit this kind of behavior?

The LED turns on and off every one second. The functions `digitalWrite()` and `delay()` cause the LED to exhibit this kind of behavior.

2. Disconnect the trainer board from the Gizduino then observe the onboard LED on the Gizduino. What can you conclude from this?

The onboard LED is directly connected to pin 13 of the Gizduino.

Guide Questions I.B

1. What does the `millis()` function do?

It returns the number of milliseconds since the Arduino board began running the current program.

2. How does the blinking LED sketch in this part differ in the blinking LED sketch in Part A?

The blinking LED sketch in this part does not use the `delay()` function in blinking the LED on and off.

Guide Questions II.A

1. Describe the behavior of the LEDs as you push the tact switches.

The LEDs turn on when the tact switches are pressed and turn off when the tact switches are not pressed.

2. What function detects the state of the tact switch when pressed?

The `digitalRead()` function detects the state of the tact switch when pressed.

Guide Questions II.B

1. Describe the behavior of the LED as you push the tact switch. How does the behavior of the LED in this part differ from the behavior of the LEDs in Part A when the tact switches are pressed?

The LED toggles on or off as the tact switch is pressed. The LED in this part turns on or off whenever the tact switch is pressed as opposed to the LEDs in the previous part which turns on or off depending on whether the tact switches are pressed or not.

Guide Questions II.C

2. Describe the behavior of the 7-segment display as the switches in the DIP switch are pushed.
The 7-segment display shows digits from 0 to 9 as switches on the DIP switch are pushed.

Review Questions

1. What is Arduino? What are its advantages compared to other microcontroller boards?
Arduino is an electronic prototyping platform which utilizes Atmel microcontrollers. It is an open-source prototyping platform with flexible and easy-to-use hardware and software which makes it a great tool for beginners.
2. What is Gizduino? What is its advantage compared to original Arduino boards?
Gizduino is an Arduino board based on the model Arduino Diecimila. It is much cheaper and more accessible than the original Arduino boards since Gizduino is available locally.
3. What is a sketch? What are the settings that must be checked in order to successfully upload a sketch to an Arduino board?
A sketch is a program written in the Arduino development environment. The board and the serial port must be initially configured in order to successfully upload a sketch to an Arduino board.
4. List and briefly describe the toolbar buttons commonly used in the Arduino development environment.
Verify/Compile – checks the code for errors; Save – saves the sketch; New – creates a new sketch; Upload to I/O board – compiles the code and uploads it to the Arduino board; Open – presents a menu of all the sketches in the sketchbook.
5. What is the difference between the `void setup()` and `void loop()` sections in an Arduino sketch?
The void setup() section runs only once when the sketch starts. The void loop() section runs indefinitely as long as the Arduino board is powered on.
6. What does the function `pinMode()` do? Briefly explain the parameters that are passed to this function.
The function `pinMode()` configures the specified pin to behave as either an input or output. The parameters passed to this function are the pin number to set (0 to 13) and the mode of the pin (either INPUT or OUTPUT)

7. Explain the advantage and disadvantage of not using the `delay()` function in a sketch which requires the use of time delays
Its advantage is that it does not affect the execution of the other sections of the sketch. Its disadvantage is that it is much more complicated to implement than the `delay()` function.
8. What does the functions `digitalRead()` and `digitalWrite()` do?
`digitalRead()` reads the value from a specified digital pin as either HIGH or LOW. `digitalWrite()` writes a HIGH or LOW value to a digital pin.
9. How does the detection of the state of inputs read to Arduino work? When is state change detection performed?
State change detection is done by detecting the transition of the input signal from LOW to HIGH or from HIGH to LOW. State change detection is used when a certain action needs to be done upon the transition of the input signal.
10. Refer to the trainer board schematic diagram in Figure 1.4. How does the flow or direction of the signal depicted in this schematic diagram?
The flow/direction of the signal is depicted by the arrow-pointed signal labels in the schematic diagram.

Activities

a.

```
int tact1State = LOW;
int tact2State = LOW;
int tact3State = LOW;
int prevTact1State = LOW;
int prevTact2State = LOW;
int prevTact3State = LOW;
int ledState = LOW;
unsigned long currentTime = 0;
unsigned long previousTime = 0;
long interval = 500;

void setup()
{
    pinMode(0, INPUT);
    pinMode(1, INPUT);
    pinMode(2, INPUT);
    pinMode(3, OUTPUT);
}

void loop()
{
```

```

    currentTime = millis();

    if (currentTime - previousTime == interval)
    {
        previousTime = currentTime;
        ledState = !ledState;
        digitalWrite(3, ledState);
    }

    tact1State = digitalRead(0);
    tact2State = digitalRead(1);
    tact3State = digitalRead(2);

    if (tact1State != prevTact1State)
    {
        if (tact1State == HIGH)
        {
            interval = 500;
        }
    }

    if (tact2State != prevTact2State)
    {
        if (tact2State == HIGH)
        {
            interval = 1000;
        }
    }
    if (tact3State != prevTact3State)
    {
        if (tack3State == HIGH)
        {
            interval = 2000;
        }
    }
    prevTact1State = tact1State;
    prevTact2State = tact2State;
    prevTact3State = tact3State;
}

```

b. `int clockState = LOW;`
`int prevClockState = LOW;`
`int Q = LOW;`

```

void setup()
{
    pinMode(0, INPUT);
    pinMode(1, INPUT);
    pinMode(2, INPUT);
    pinMode(3, OUTPUT);
    pinMode(4, OUTPUT);
}

```

```

}

void loop()
{
    clockState = digitalRead(0);

    if (clockState != prevClockState)
    {
        if (clockState == LOW)
        {
            JK();
        }
    }
    prevClockState = clockState;
}

void JK()
{
    if (digitalRead(1) == LOW && digitalRead(2) == LOW)
    {
        digitalWrite(3, Q);
        digitalWrite(4, !Q);
    }
    else if (digitalRead(1) == LOW && digitalRead(2) == HIGH)
    {
        Q = LOW;
        digitalWrite(3, Q);
        digitalWrite(4, !Q);
    }
    else if (digitalRead(1) == HIGH && digitalRead(2) == LOW)
    {
        Q = HIGH;
        digitalWrite(3, Q);
        digitalWrite(4, !Q);
    }
    else
    {
        Q = !Q;
        digitalWrite(3, Q);
        digitalWrite(4, !Q);
    }
}
}

```

```

C. void setup()
{
    pinMode(0, INPUT);
    pinMode(1, INPUT);
    pinMode(2, INPUT);
    pinMode(3, INPUT);
    pinMode(4, INPUT);
    pinMode(5, INPUT);
}

```

```

    pinMode(6, INPUT);
    pinMode(7, INPUT);

    pinMode(8, OUTPUT);
    pinMode(9, OUTPUT);
    pinMode(10, OUTPUT);
    pinMode(11, OUTPUT);
}

void loop()
{
    int addend, augend, sum;
    addend = (digitalRead(7)<<3)+ (digitalRead(6)<<2)+
        (digitalRead(5)<<1)+ digitalRead(4);
    augend = (digitalRead(3)<<3)+ (digitalRead(2)<<2)+
        (digitalRead(1)<<1)+ digitalRead(0);
    sum = addend + augend;
    digitalWrite(11, sum&8);
    digitalWrite(10, sum&4);
    digitalWrite(9, sum&2);
    digitalWrite(8, sum&1);
}

```

d. `int tact1State = LOW;`
`int tact2State = LOW;`
`int prevTact1State = LOW;`
`int prevTact2State = LOW;`
`int num = 0;`

```

void setup()
{
    pinMode(0, INPUT);
    pinMode(1, INPUT);
    pinMode(2, INPUT);
    pinMode(3, INPUT);
    pinMode(4, INPUT);
    pinMode(5, INPUT);
    pinMode(6, INPUT);
    pinMode(7, INPUT);

    pinMode(8, INPUT);
    pinMode(9, INPUT);

    pinMode(10, OUTPUT);
    pinMode(11, OUTPUT);
    pinMode(12, OUTPUT);
    pinMode(13, OUTPUT);
    display7seg(num);
}

void loop()

```

```

{
    tact1State = digitalRead(8);
    tact2State = digitalRead(9);

    if (tact1State != prevTact1State)
    {
        if (tact1State == HIGH)
        {
            readDIP();
        }
    }

    if (tact2State != prevTact2State)
    {
        if (tact2State == HIGH)
        {
            if (num == 9)
            {
                num = 0;
            }
            else
            {
                num++;
            }
            display7seg(num);
        }
    }
    prevTact1State = tact1State;
    prevTact2State = tact2State;
}

void readDIP()
{
    if (digitalRead(7) == HIGH)
    {
        num = 8;
    }
    else if (digitalRead(6) == HIGH)
    {
        num = 7;
    }
    else if (digitalRead(5) == HIGH)
    {
        num = 6;
    }
    else if (digitalRead(4) == HIGH)
    {
        num = 5;
    }
    else if (digitalRead(3) == HIGH)
    {
        num = 4;
    }
}

```

```

    }
    else if (digitalRead(2) == HIGH)
    {
        num = 3;
    }
    else if (digitalRead(1) == HIGH)
    {
        num = 2;
    }
    else if (digitalRead(0) == HIGH)
    {
        num = 1;
    }
    else
    {
        num = 0;
    }
    display7seg(num);
}

void display7seg(int x)
{
    digitalWrite(13, x&8);
    digitalWrite(12, x&4);
    digitalWrite(11, x&2);
    digitalWrite(10, x&1);
}

```

APPENDIX C

Arduino Laboratory Assessment Form

Experiment No. _____

ARDUINO LABORATORY ASSESSMENT QUESTIONNAIRE

Instructions: Please read each statement and indicate how much you agree or disagree by putting a ✓ or an X on the box that best reflects your opinion. Mark only one box per statement.

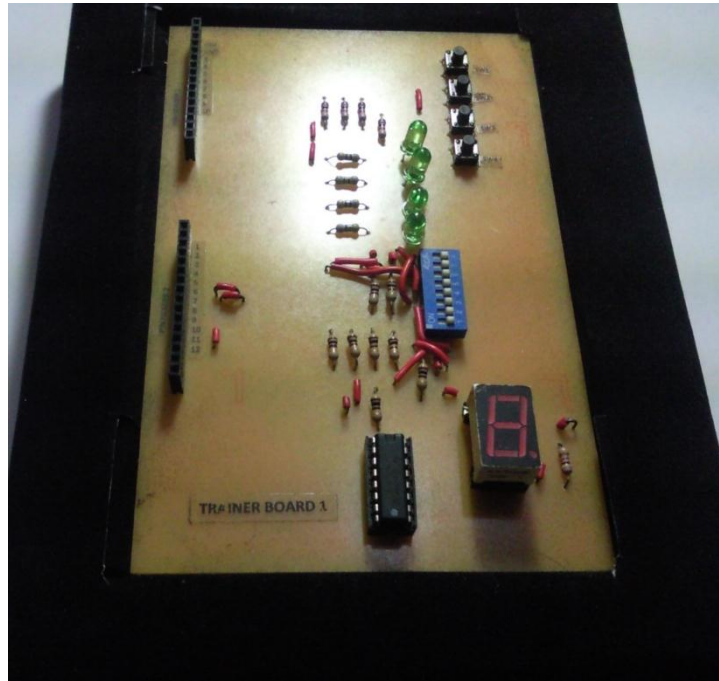
	Strongly Agree	Agree	Neither	Disagree	Strongly Disagree
A. Laboratory Experiment					
1. The discussion is clear in introducing the basic concepts necessary to perform the experiment.					
2. The schematic diagram is readable and understandable.					
3. The procedures are clear and easy to follow.					
4. The activities are easy to perform.					
5. The objectives of the experiment have been met.					
B. Trainer Board					
1. The trainer board is suited to the experiment to be performed.					
2. The trainer board is helpful in understanding and performing the experiment.					
C. Self-Assessment					
1. I am confident that I could apply the concepts I learned in the experiment using an Arduino board.					
2. I am confident that the knowledge I obtained will be helpful in learning and using other kinds of microcontrollers.					

Name: _____ Program/Yr: _____ Signature: _____

APPENDIX D

Pictures of Prototype

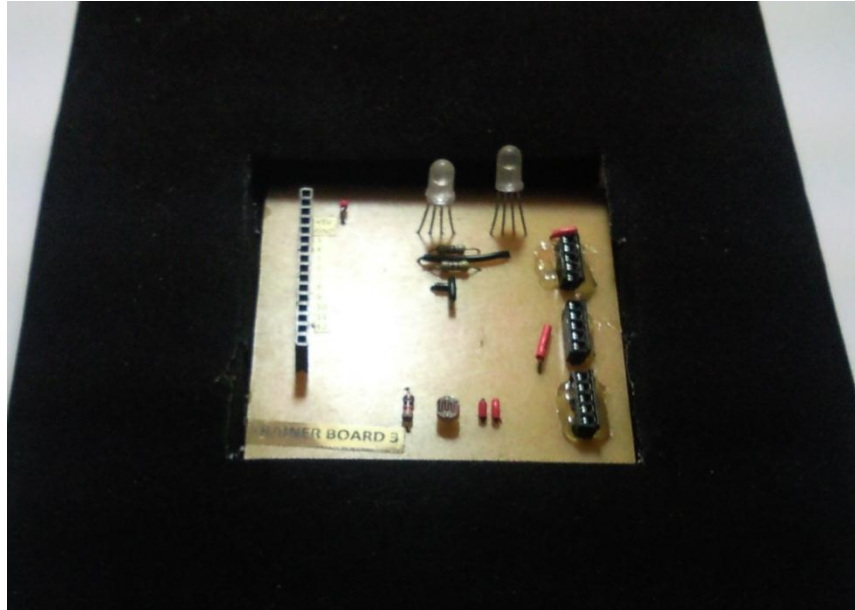
PICTURES OF PROTOTYPE



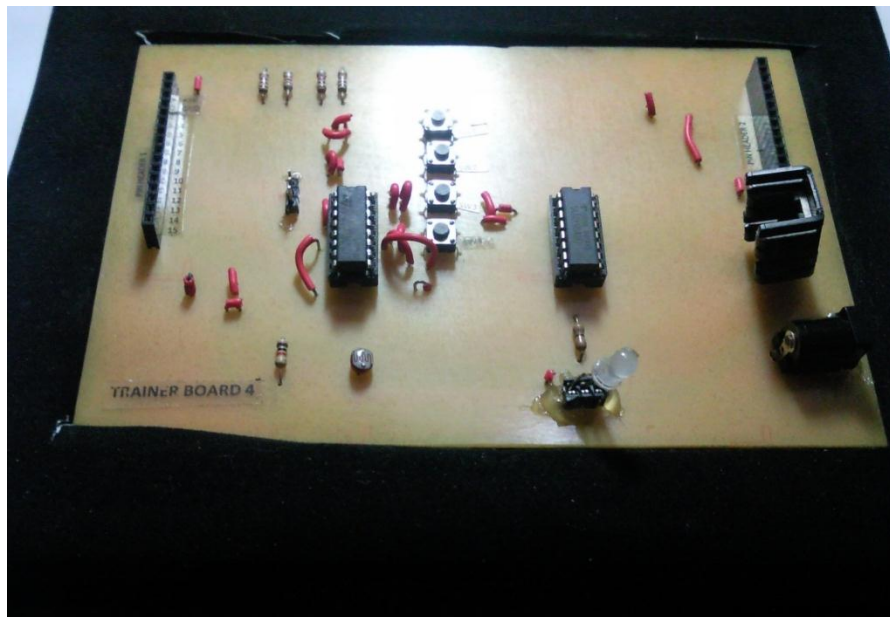
Trainer Board 01



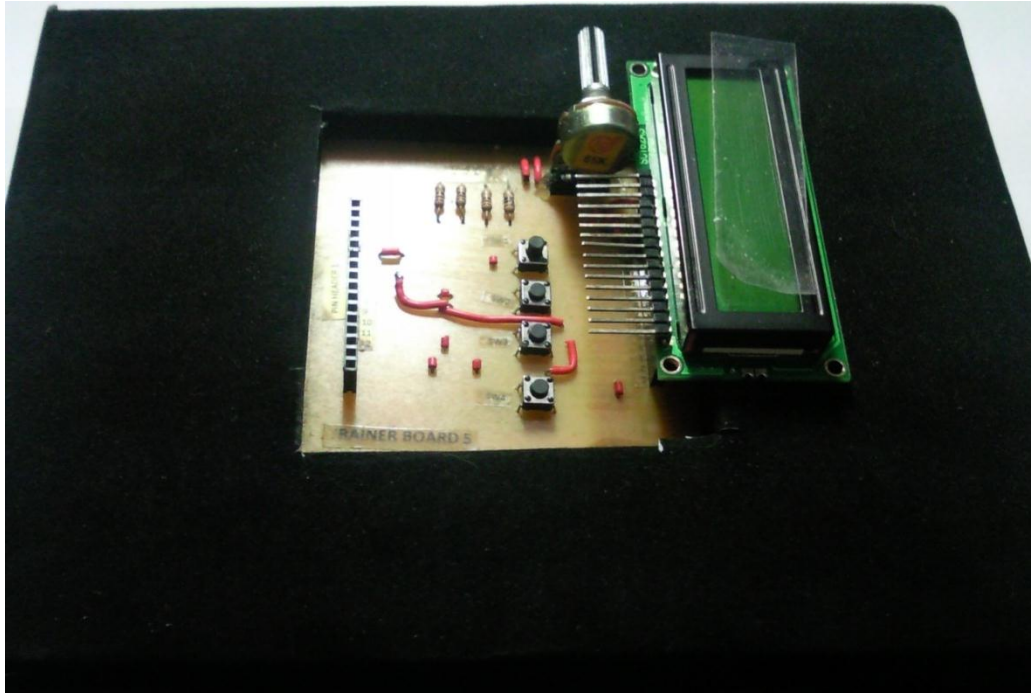
Trainer Board 02



Trainer Board 03



Trainer Board 04



Trainer Board 05