THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MECHANICAL AND NUCLEAR ENGINEERING

DEVELOPMENT OF A MECHATRONICS EDUCATION SYSTEM FOR MECHANICAL ENGINEERS AT PENN STATE UNIVERSITY

KEVIN SWANSON
SPRING 2011

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Mechanical Engineering
with honors in Mechanical Engineering

Reviewed and approved* by the following:

Dr. Sean Brennan
Professor of Mechanical Engineering
Thesis Supervisor/Honors Adviser

Dr. Eric Marsh
Professor of Mechanical Engineering
Faculty Reader

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

This thesis seeks to develop an improved program interface for teaching Mechatronics to Mechanical Engineers at Penn State. Mechanical Engineers typically have a weak background in both circuitry and programming, giving Electrical Engineers and Computer Scientists an advantage in Mechatronics that is heavily reliant on either approach. However, hardware and software exists that, if used appropriately, can help Mechanical Engineers improve their ability to learn and develop Mechatronics solutions. This thesis explores the use of an Arduino microcontroller, MATLAB, and Simulink to effectively implement data acquisition systems that are an integral part of Mechatronics.

These labs require the student to communicate between the microcontroller and PC in two ways. First, the student will connect an Arduino, Ethernet Shield, and WIZnet to a PC running Simulink with an Ethernet connection. This method has the advantage of giving the student an easier programming platform through Simulink's visual programming structure, along with allowing the student to monitor the data in real time. Second, the student will connect an Arduino to MATLAB via a serial connection. While this method does not contain the benefit of Simulink's visual programming language, it does provide a simple platform on which the student can plot data.

The Arduino to Simulink connection proves to be a very beneficial system for the student. The labs give a base for which the student can do basic Mechatronic functions such as read and write data, or implement advanced control system utilizing the data acquisition system to allow interaction with a real-plant. The Arduino to MATLAB connection will be beneficial to students for the plotting ability. While MATLAB is not a visual language like Simulink, it does provide a relatively simple medium for Mechanical Engineers to collect and plot data. Currently the MATLAB plots are not in "real time" because of slow plotting speeds. Further investigation

could improve the MATLAB plotting techniques and increase the benefits of using MATLAB for data acquisition.

Table of Contents

ABSTRACT i

LIST OF FIGURES v

ACKNOWLEDGEMENTS vii

Chapter 1 1

Introduction 1

Chapter 2 3

Data Acquisition in Mechatronics 3
Possible Microprocessors 5
MATLAB and Simulink 7
Required Equipment and Affordability 8

Chapter 3 12

Lab Design 12
Digital I/O Labs 13
Analog I/O Labs 23
Digital I/O Using Serial Communication 29
Analog I/O Using Serial Communication 33

Chapter 4 37

Results 37
Conclusions 46
Further Recommendations 46

Bibliography 48

Appendix A – Starter Arduino Code and Simulink Diagram 49

Appendix B – Lab 1: Arduino Digital Output 57

Appendix C – Lab 2: Arduino Digital I/O 60

Appendix D – Lab 3: Arduino Analog Output 63

Appendix E – Lab 4: Arduino Analog I/O 67

Appendix F – Lab 5: Arduino Analog Control 71

Appendix G – Lab 6: MATLAB Serial Digital Read 76

Appendix H – Lab 7: MATLAB Serial Digital I/O 78

Appendix I – Lab 8: MATLAB Serial Analog I/O 80

LIST OF FIGURES

Figure 1: Various Interfaces of an Ideal Microprocessor 4

Figure 2: Microprocessor Comparison Chart.....7

Figure 3: PIC 16F690 - http://parts.digikey.com/1/parts/560178-ic-pic-mcu-flash-4kx14-20dip-pic16f690-i-p.html.....9

Figure 4: Arduino MEGA - http://www.davidorlo.com/articles/arduino/arduino-getting-started.....10

Figure 5: Hardware Setup for TCP/IP Communication with no Additional Peripherals.....13

Figure 6: Starter Simulink Diagram.....16

Figure 7: Arduino Digital Output Lab Simulink Diagram.....19

Figure 8: Arduino Digital Output Lab Simulink Response 20

Figure 9: Arduino Digital I/O Lab Simulink Diagram 22

Figure 10: Hardware Setup for TCP/IP Communication with Potentiometer and Servo 24

Figure 11: Arduino Analog Control Lab Simulink Diagram.....28

Figure 12: Data Conversion Subsystem.....29

Figure 13: Hardware Setup for Serial Communication with no Additional Peripherals 30

Figure 14: Hardware Setup for Serial Communication with Potentiometer and Servo 33

Figure 15: Communication Delay without Flush.....38

Figure 16: Communication Delay with Flush.....39

Figure 17: Communication Delay with Timed Flush 40

Figure 18: Plotting Delay in MATLAB Serial Analog I/O Lab 42

Figure 19: MATLAB Serial Analog I/O Lab Delay - Time as Double 44

Figure 20: MATLAB Serial Analog I/O Lab Delay – Time as Integer.....45

Figure 22: Starter Simulink Diagram.....56

Figure 23: Lab 1 Simulink Diagram 59

Figure 24: Lab 2 Simulink Diagram 63

Figure 25: Lab 3 Simulink Diagram 66

Figure 26: Lab 4 Simulink Diagram 70

Figure 27: Lab 5 Simulink Diagram 74

Figure 28: Lab 5 Control Block 75

ACKNOWLEDGEMENTS

I would like to thank Dr. Brennan for all his help and support in completing my Undergraduate Honors Thesis. Not only has he taken time out of his hectic schedule to help me find multiple project options, but he has been more than willing to provide aid along the way.

I would also like to thank Alex Brown for his support throughout this project. Alex provided extremely valuable knowledge about Simulink along with example programs to get the project started. This thesis would not have been successful without his instruction.

Chapter 1

Introduction

Mechatronics is a growing field in engineering and computer science that has many applications to recent technology. Nearly all robotics and automation projects require knowledge of mechatronics and programming. The main users of Mechatronics are Electrical Engineers and Computer Scientists because of their electrical design and programming backgrounds. Recently, however, Mechatronics has overflowed into all fields of engineering because of its use in controlling dynamic systems, many of which are mechanical in nature. Unfortunately the teaching methods of Mechatronics that are typically used in Electrical Engineering and Computer Science domains are not always suited to Mechanical Engineers' level of electrical and programming knowledge.

Others have recognized the need to improve the electrical focus of mechatronics education to level the playing field between electrical and all other types of engineers. Professor Victor Guirgiutiu of The University of South Carolina notes that many non-Electrical Engineers are attempting to break into the field of Mechatronics by enrolling in high-level electrical engineering courses, but the lack of a strong electrical background leaves them behind in the class and results in a sub-par understanding of the material and possibly a lower GPA [1] [2]. Guirgiutiu's focus in Mechatronics education is to create a series of modules that students can use to quickly gain experience in the field of electronics. There exists, however, another aspect of Mechatronics that is largely overlooked in these modules: programming. Mechatronics programming currently depends heavily on writing code in C which requires training and experience. Unfortunately, other software - MATLAB especially - is often taught to Mechanical Engineers for programming, leaving a gap in ME's ability to effectively program their circuits

and utilize their training in electronics. A goal of this thesis is to explore the combinations of C with software more typically used by MEs, especially MATLAB and Simulink, for learning Mechatronics.

According to Onur Erdener's thesis on Mechatronics education, many universities with strong engineering programs such as Georgia Tech, Rensselaer Polytechnic Institute, and Northwestern University create labs that are completely devoted to Mechatronics and can build a strong base for electrical and programming education [3]. The purpose of these labs is to immerse the students in Mechatronics studies to gain experience in all areas. There are two problems with this. First, students who want just a basic knowledge of Mechatronics may not be willing to commit themselves to a series of labs and may therefore be dissuaded from pursuing Mechatronics. Second, universities without established labs may not have the funding to build a lab solely devoted to Mechatronics. Penn State, for example, is currently under a proposed 52% state budget cut [4], so it is unlikely that much funding would soon exist for a dedicated Mechatronics lab. For this reason, a method of improving Mechatronics education must be used that does not require the development of an entirely new program or set of equipment, but instead utilizes existing base skills.

This thesis will outline a series of labs that seek to improve the Mechatronics skills of Mechanical Engineers at Penn State. The focus of these labs will be the creation of a digital acquisition system that satisfies all of the core needs of an effective digital acquisition system (defined in the following chapter) while using techniques appropriate to the skill level of mechanical engineers and without incurring large costs on the Mechanical Engineering Department.

Chapter 2

Data Acquisition in Mechatronics

Data acquisition involves the collection and processing of data for use in automated control, and programming is an integral part of Mechatronics when designing data acquisition (DAQ) systems. It is impossible to use Mechatronics without using data acquisition. Even the most basic system requires some feedback. For example, a simple IR distance sensor must use a DAQ system to first get raw data (in voltage from the IR sensor), process this data, and calculate a real-world distance from the voltage, and send the data back out in the form of a display for the user. Most of this is accomplished through programming. As the requirements of the system become more complex, more sensors will be added and the DAQ system will become more complex.

Most DAQ systems involve both a microprocessor and a personal computer (PC). A key premise of the work of this thesis is that the core needs of a fully equipped DAQ system can be split between the microprocessor requirements, and the PC requirements. For microprocessors, all have the ability to read digital inputs and change digital outputs (digital I/O). Some other abilities include analog I/O; power management; and interfacing with users via LCD screens, reading advanced signals (serial, I2C, etc), and communicating with other DAQ systems. The possible abilities of a microprocessor can be seen in Figure 1.

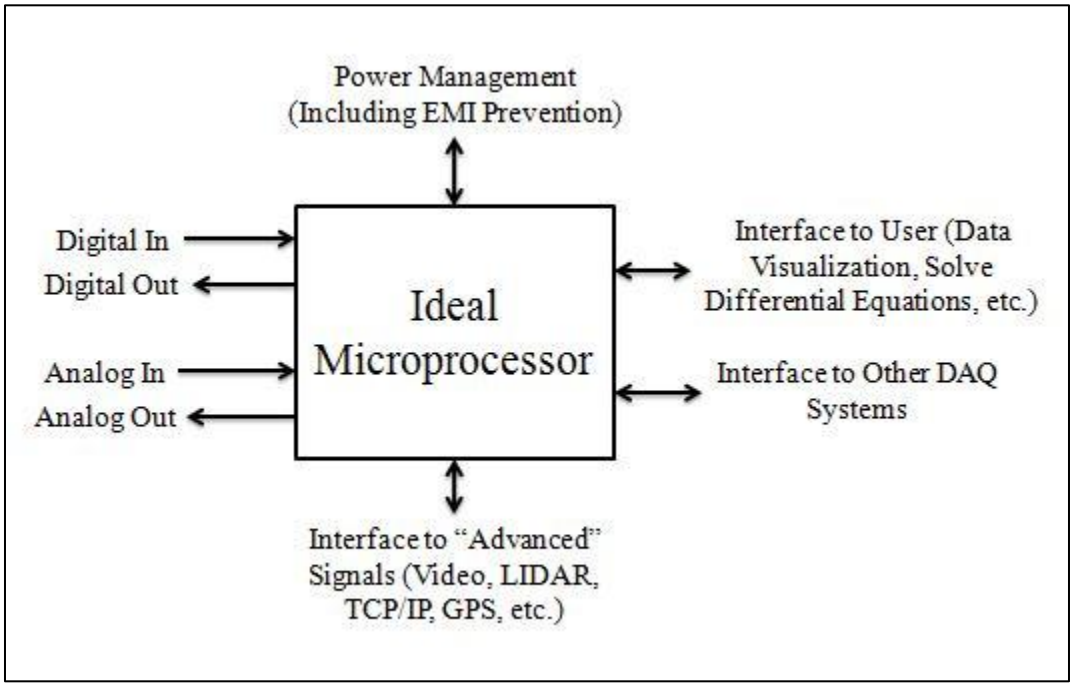

Figure 1: Various Interfaces of an Ideal Microprocessor

Most microprocessors do not have all of these capabilities, but all microprocessors have some. Similarly, the personal computer often does have all of these capabilities when suitable DAQ cards are installed, but this configuration is both very expensive and non-portable.

The core needs of the PC are as follows:

- Ability to interface with the microprocessor.
- Ability to communicate with multiple pieces of hardware.
- Ability to solve differential equations.
- Ability to process advanced peripherals (cameras, for example).

Most PC's have these capabilities depending on the software they are running.

Because both PCs and microprocessors have relative strengths and weaknesses, it is useful to consider a microprocessor that has a good balance of these capabilities and then develop some interface to work with a PC. More importantly, however, is the selection of a microprocessor that satisfies its core needs the best.

Possible Microprocessors

The original microprocessor that was used in Penn State's Mechatronics class was the digital signal processor (DSP). DSPs have multiple benefits and problems that can be seen below [5].

Advantages:

- Very fast.

Disadvantages:

- Very expensive. Each DSP costs around $600.
- It cannot handle analog I/O without custom hardware.
- It cannot handle peripherals like encoders without additional hardware.
- Power is not managed well and it has weak EMI protection.

Another possible system is to use a PIC microprocessor, which is the system used at PSU's Advanced Mechatronics course after DSPs were phased out. The PIC microprocessor comes in many shapes and sizes that provide a different number of I/O ports. All PICs have the following advantages and problems [5].

Advantages:

- Moderately fast.
- Able to handle analog I/O.
- Very cheap. Each costs around $2.

Disadvantages:

- Requires custom hardware to use peripherals.
- Cannot handle advanced peripherals like GPS.
- No power management.
- The compiler software is expensive. (at least $500)

- Poor interface. It relies on C programming with little documentation on microprocessor-specific function commands.

The final microprocessor that is used currently in the Advanced Mechatronics course is the Arduino MEGA. It has many advantages that make it ideal for mechatronics applications.

Advantages:

- Able to use analog I/O.
- Requires custom hardware for peripherals, but is able to process advanced signals.
- Decent power management.

Disadvantages:

- Expensive compared to PICs. Each MEGA costs $65 [7] plus more for additional hardware.
- Poor interface without additional hardware/software.
- Larger than other microprocessors.

A summary of the benefits of each microprocessor can be seen below.

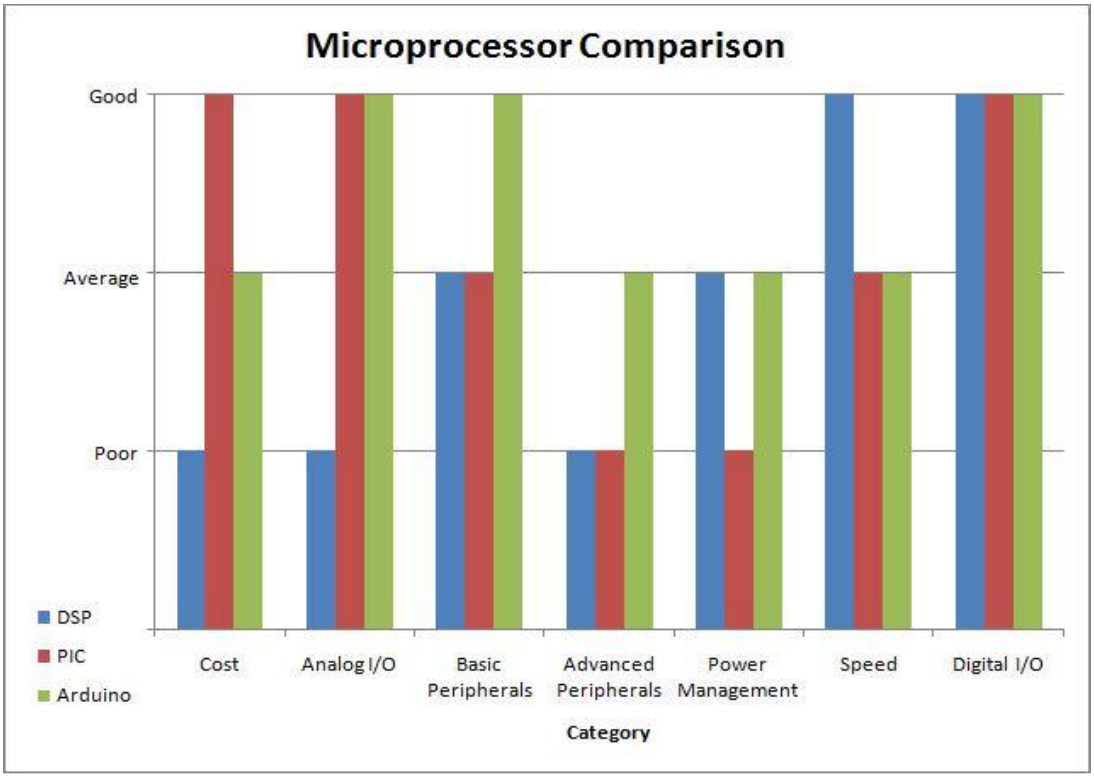

Figure 2: Microprocessor Comparison Chart

Even with the selection of a microprocessor, it is still necessary to select software that enables the user to interface the microprocessor with a personal computer. The software selection is important because it defines the user interface of your DAQ system. While most DAQ systems use C code, this method is not very user friendly. Many improvements can be made utilizing MATLAB and Simulink as the microprocessor to PC interface.

MATLAB and Simulink

Mechanical Engineers at Penn State are required to take at least one programming class, which for MEs is usually taught in MATLAB. One of the prerequisite classes for Mechanical Engineers is CMPSC 200 (Programming for Engineers with MATLAB). While this class gives only an introductory look to MATLAB, it builds a great base from which to expand on in future

classes. Once students progress into 400 level classes they will be required to use MATLAB in many cases. Classes such as ME 450 (Modeling Dynamic Systems) and ME 452 (Automotive Vehicle Dynamics) utilize MATLAB and expand on it by implementing Simulink, which is an add-on graphical differential equation solver to MATLAB, for solving differential equations and modeling systems.

There are multiple reasons that MATLAB and Simulink are the preferred programs for mechanical engineers. The reason that Simulink is easier for mechanical engineers than C programming is that it is a visual programming language. Instead of writing lines of code, the user simply grabs blocks from a library, places them into the workspace, and connects them with arrows. Settings for these blocks can be changed easily by double clicking on the block and editing the available options. By using Simulink, users can easily set up, analyze, and debug programs without reading through multiple lines of code to find errors. This visual system is very beneficial to mechanical engineers who do not have extensive training in C programming. The ability to use this knowledge of Simulink would give Mechanical Engineers an advantage when entering the field of Mechatronics.

Required Equipment and Affordability

One concern that arises with the implementation of a new system is feasibility. To effectively analyze the viability of a new system it is necessary to determine what supplies and programs are needed for each. Many methods are available for programming with C. One method that is implemented in Penn State classes is using the PIC microcontroller and compiler. Another is using the Arduino microcontroller and compiler.

The PIC is popular because it is very small and inexpensive. The PIC 16F690 that is used

in ME 445 is only 1cm by 3cm in size and can be purchased from multiple vendors for around $2 each [6]. The compiler software for the PIC must be purchased, however, for $500 [5].

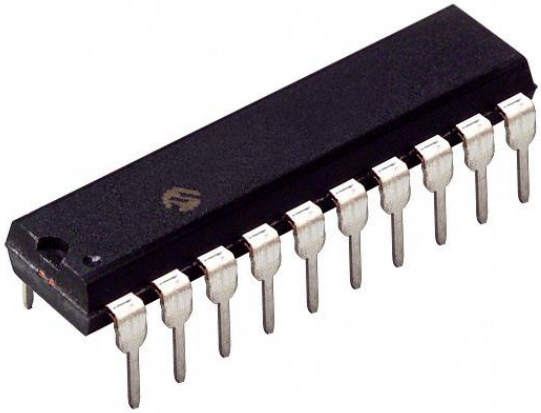

Figure 3: PIC 16F690 - http://parts.digikey.com/1/parts/560178-ic-pic-mcu-flash-4kx14-20dip-pic16f690-i-p.html

The Arduino is larger and more expensive than the PIC. The Arduino MEGA that is used in ME 597D is 16cm by 6cm in size and costs $65 each [7]. While the board itself is expensive the software is a free download from the Arduino website, http://www.arduino.cc/. Luckily, with the exception of the Arduino MEGA boards, all of this equipment is readily available in Penn State's Mechanical Engineering labs on campus.

Figure 4: Arduino MEGA - http://www.davidorlo.com/articles/arduino/arduino-getting-started

In order to use Simulink to read the data and write code, the user must be able communicate between the microprocessor and the computer. The method that the following labs implement is to use an Arduino with a WIZnet and Ethernet Shield to collect raw data and send it to the computer. The WIZnet can be purchased from SparkFun for $25 [8] and the Ethernet Shield can be purchased from Adafruit for $15 [9]. A simple code must be written using the Arduino program to communicate with Simulink. On the PC, the Arduino communicates with Simulink through QuaRC, which is a third-party software created by Quanser to supplement Simulink. It enables microcontrollers to communicate with Simulink via Ethernet connection through a TCP/IP interface. These programs are also all available in Penn State's Mechanical Engineering laboratory. The Arduino can also communicate with MATLAB through a serial connection, which would make QuaRC unnecessary. The viability of using this technique will be explored later in this thesis.

Since the only additional costs that exist for teaching Mechatronics using Simulink are the Arduino MEGA, WIZnet, and Ethernet Shield, the use of this method is feasible. Even if the

students are required to purchase their own supplies the cost would still be less than a typical textbook.

Chapter 3

Lab Design

This thesis aims to implements a series of labs which will determine whether or not Simulink can effectively be used as the bridge between the microprocessor and computer for automated control. Each lab will expand on the last to improve the student's understanding of data acquisition. Ultimately, the student should be comfortable with the process of collecting both analog and digital data on the microprocessor and sending that data to and from Simulink. Some labs will also require the student to implement controls on the system which will illustrate the usefulness of this programming method.

These labs will be created to teach the basic communication between the Arduino and Simulink. The series of labs will be used to gain experience in different elements of data acquisition. Initially, this thesis considers how to use the digital I/O of the Arduino and open basic communication with Simulink. Next, the subsequent lab will expand on the digital I/O by requiring the student to control the digital output of the Arduino through Simulink. Once digital communication is established, the next two labs use analog inputs and outputs, respectively. Finally, after digital and analog I/O is established, this chapter considers a basic controller implementation using Simulink's visual programming, communicating with Arduino hardware over TCP/IP.

After the TCP/IP communication is established, it is necessary to look into the possibility of using this system without the Ethernet connection and QuaRC, both of which add expense and complexity. The thesis will explore the usefulness of direct communication via the serial port. While this does not provide the benefits of the Simulink's visual programming, it does provide an interface over which the user can view data. To explore the use of the serial port, the previous

labs conducted with the Ethernet connection will be repeated with a serial connection instead.

Digital I/O Labs

The first two labs designed as part of this thesis seek to give the student experience with opening communication between the Arduino and PC via Ethernet by using Simulink and QuaRC's TCP/IP interface. The hardware setups for the digital I/O labs are simply an Arduino MEGA with an Ethernet shield and WIZnet connected to the PC via USB for power and via Ethernet for communication. This setup requires no additional peripherals

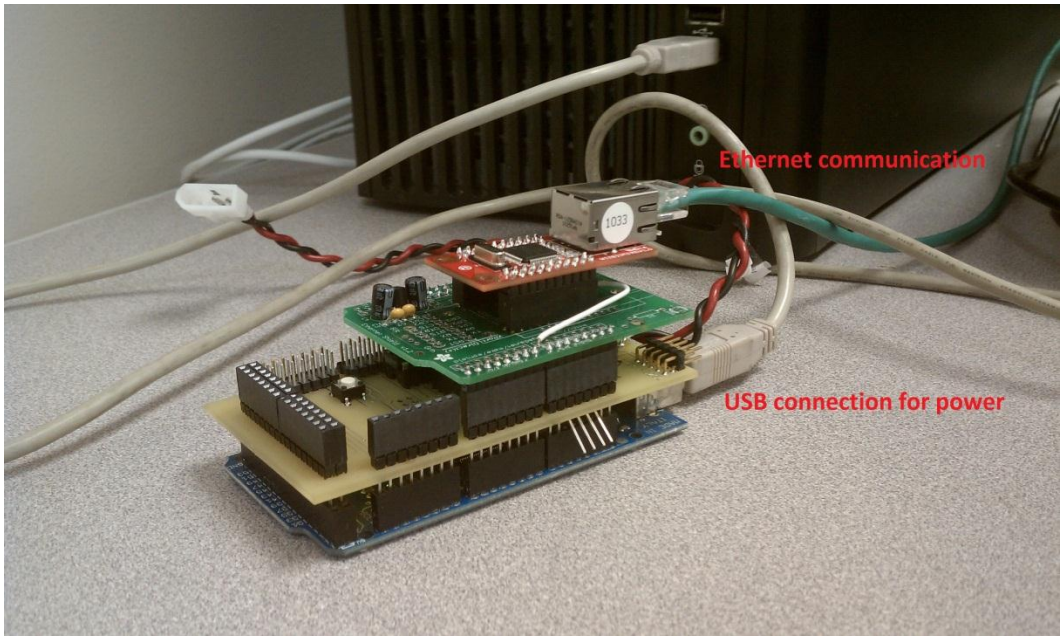

Figure 5: Hardware Setup for TCP/IP Communication with no Additional Peripherals

Students will initially be given a starter code for both the Arduino and Simulink that serves as an example of basic communication between the Arduino and Simulink through a TCP/IP interface. The Arduino starter code contains a code titled "Hardware" that sets up the ability for communication through QuaRC. The "Hardware" code will open automatically with the starter code and should never be altered in these labs. The code titled "arduino_loopback"

opens the Ethernet communication with Simulink, reads data from Simulink, and sends the same data back. This is the portion of the code that should be altered in each lab. It can be seen below.

```
#include <Wire.h>
#include "Messenger.h"
#include <Ethernet.h>
//ETHERNET CONFIG
byte mac[] = { 0x1C, 0xBD, 0xB9, 0x1C, 0x1C, 0x06 };
byte ip[] = { 172, 16, 2, 106 };
byte gateway[] = { 172, 16, 2, 1 };
byte subnet[] = { 255, 255, 0, 0 };
byte server[] = {172,16,1,106};
//Server dataserver(5100);
int port = 5106;
Client client(server,port);

//now we need to set up a messenger 'object'
//default delimiter is the space character.
Messenger message = Messenger();

double value[3] = {0,0,0};
int messageflag = 0;//this tells the program whether we've gotten something

//now we need the callback funtion for messenger
void messageReady(){
  //loop through all the available elements of our message,
  //assigning our read values as we go.

  for(int i=0;i<3;i++){
  if (message.available()){
    messageflag = 1;
    value[i] = message.readDouble();
    }
  }
}

void setup()
{
  Serial.begin(115200);
  Serial.println("hello!");
  Serial.println("My IP address is:");
  Serial.print(ip[0],DEC); Serial.print("."); Serial.print(ip[1],DEC); Serial.print(".");
Serial.print(ip[2],DEC); Serial.print("."); Serial.println(ip[3],DEC);
  Serial.println("I would like to connect to Simulink on computer:");

Serial.print(server[0],DEC);Serial.print(".");Serial.print(server[1],DEC);Serial.print(".");Serial.pri
nt(server[2],DEC);Serial.print(".");Serial.println(server[3],DEC);
  Serial.print("on port number: ");Serial.println(port);
  // start Ethernet
  Ethernet.begin(mac,ip,gateway,subnet);
  //server.begin();
```

```
  //now we must attach that callback funtion to the messenger object
  message.attach(messageReady);
if (client.connect()){
  Serial.println("connected");
}
else{
  Serial.println("connection failed");
}
}

void loop()
{

  //first let's make sure we are connected
    // Connect to server, check for new data
if (!client.connected()){
  client.flush();
  Serial.println("problem!");
  client.stop();
  Serial.println("connecting...");
if (client.connect()){
  Serial.println("back online!");
}
}
  messageflag = 0;
  //first thing's first: read our message. this fills value into a 3x1 vector

  while(client.available()){
    message.process(client.read());
  }

 //now let's send our data back to QuaRC
 if (messageflag ==1){
    for( int i=0;i<3;i++){
    //print each value[] element
    client.print(value[i]);
    //delimit with a space
    client.print(" ");
  }
  delayMicroseconds(10);
  //end the message with a carriage return
  client.println();
  delayMicroseconds(10);
  messageflag = 0;
  client.flush();
  }

}
```

The student will be required to alter this portion of the code in each lab to read and write the appropriate data. The starter code for Simulink contains four major sections that can be seen

below.

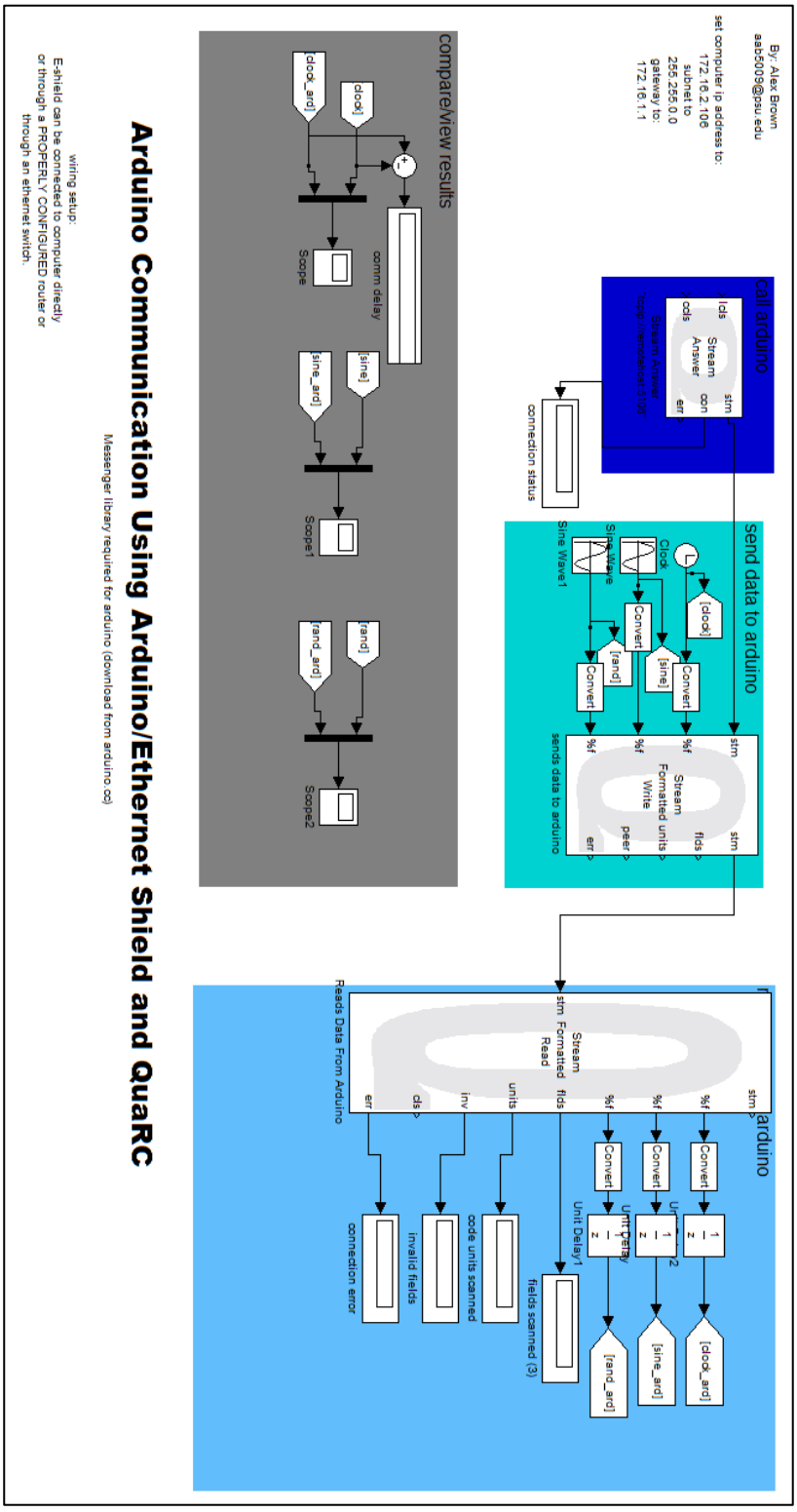

Figure 6: Starter Simulink Diagram

“Call arduino” opens communication with the Arduino, “send data to arduino” sends a clock signal and two different sine waves to the Arduino, “read data from arduino” reads the data sent back from the Arduino and routes this data to “Goto” blocks, and “compare/view results” sends the data to scopes or displays to be monitored. In each lab the student will alter this diagram to read, write, and display the appropriate data. The full starter code for both the Arduino and Simulink can be seen in Appendix A

The student’s first assignment will be to recreate the basic Arduino blink example and monitor the “blinking” as a Simulink plot instead of with an LED. The blink program is performed by setting a digital output pin high for a set time, then low for a set time, and looping this process. To set up this program the student must first designate a pin as the output pin and initiate it as an output in the setup loop. The beginning of the code starting after the Ethernet configuration can be seen below.

```
int ledPin = 13; //designate the pin for your LED

//now we need to set up a messenger 'object'
//default delimiter is the space character.
Messenger message = Messenger();

int output = 0; //initialize a variable for the output state of the LED

//this loop prints the connection status to the serial port
void setup()
{
  pinMode(ledPin, OUTPUT);  //initialize the state of the LED pin
  Serial.begin(115200);
  Serial.println("hello!");
```

Programming the pin to blink is more difficult than in the example program. In the blink example one can simply set the pin high and wait for a second before setting it low again. Since the program must continuously send values to Simulink there can be no one second delays, so instead the program is written to repeatedly send the high value for about one second before repeatedly sending low. The loop that performs this action can be seen below.

```
for(int i=0;i<200;i++){  //this for loop sets the period of the
  //square wave. each period will be approximately 200 times the
  //10 ms delay, but there will still be a delay
  if(i<100){  //this sets the time that the signal will be held HIGH
    output = 1;  //pulls the signal HIGH
  }
  else{  //outside of the previous time
    output = 0;  //pull the signal LOW
  }
    digitalWrite(ledPin, output);  //write the signal to the LED
    client.println(output);  //write the signal to Simulink
    delay(10);
}
```

These are the only additions that must be made to the example program. Since the Arduino is not required to read and data in this lab, much of the Arduino starter code can be erased. The only necessary parts are those that send data to Simulink. As seen below, much of the starter code for Simulink can also be erased.

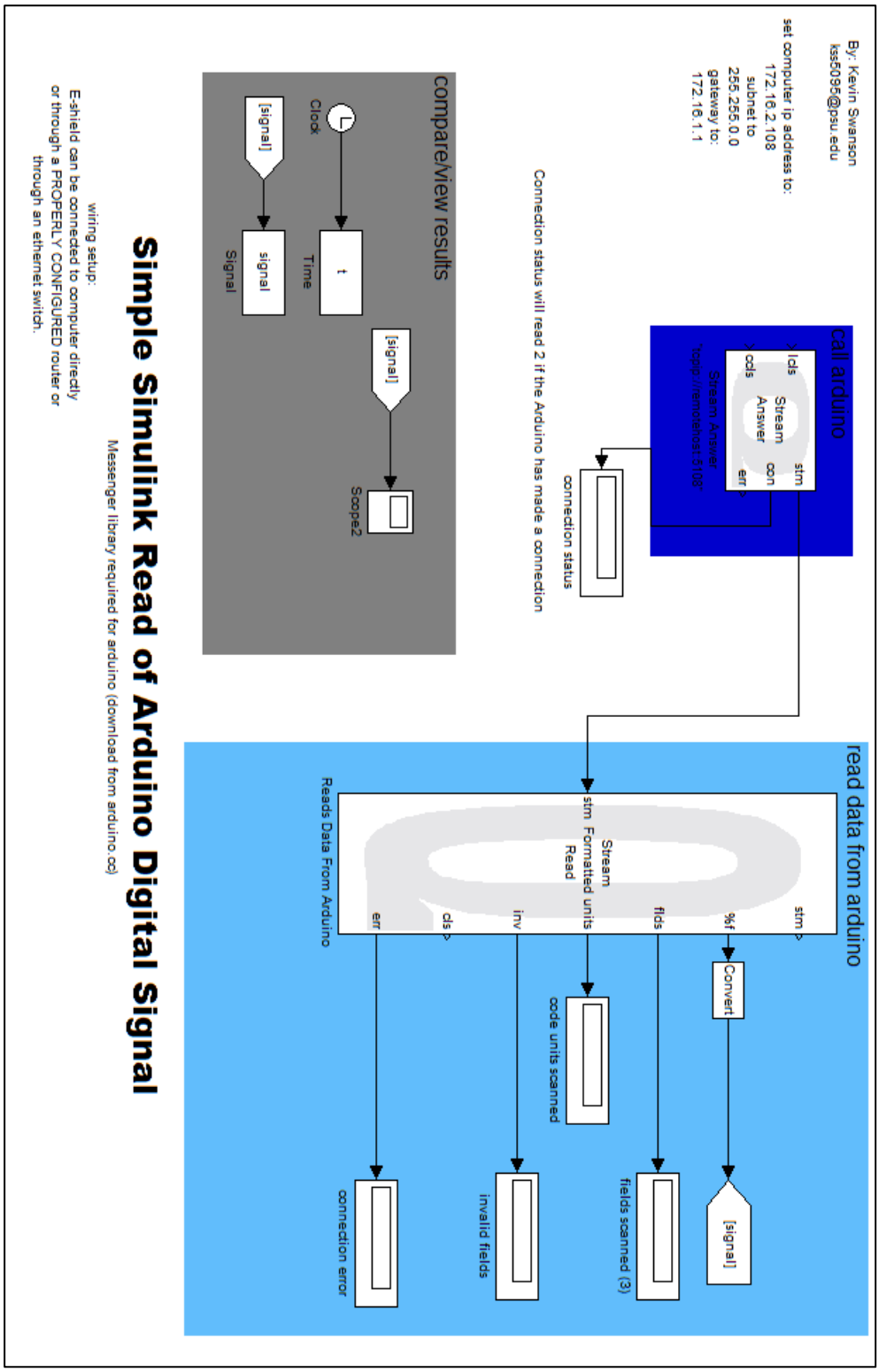

Figure 7: Arduino Digital Output Lab Simulink Diagram

The entire “send data to arduino” section can be erased and the only one piece of data must be read, so the “Stream Formatted Read” block will only have one “%f” in its settings.

The final versions of the Arduino code and Simulink diagram for the Arduino Digital Output Lab can be seen in Appendix B.

When generating a blink with a “for loop” it is very difficult to control the duration of the blink. Even though it is possible to calculate a period with the “delay()” function, there is still a delay inherent in data transfer that will increase this time. The last code, for example, set the period to be two second by setting 200 repetitions with a 10ms delay. As can be seen below, this was not the true period. The true period ended up being about 2.2 seconds.

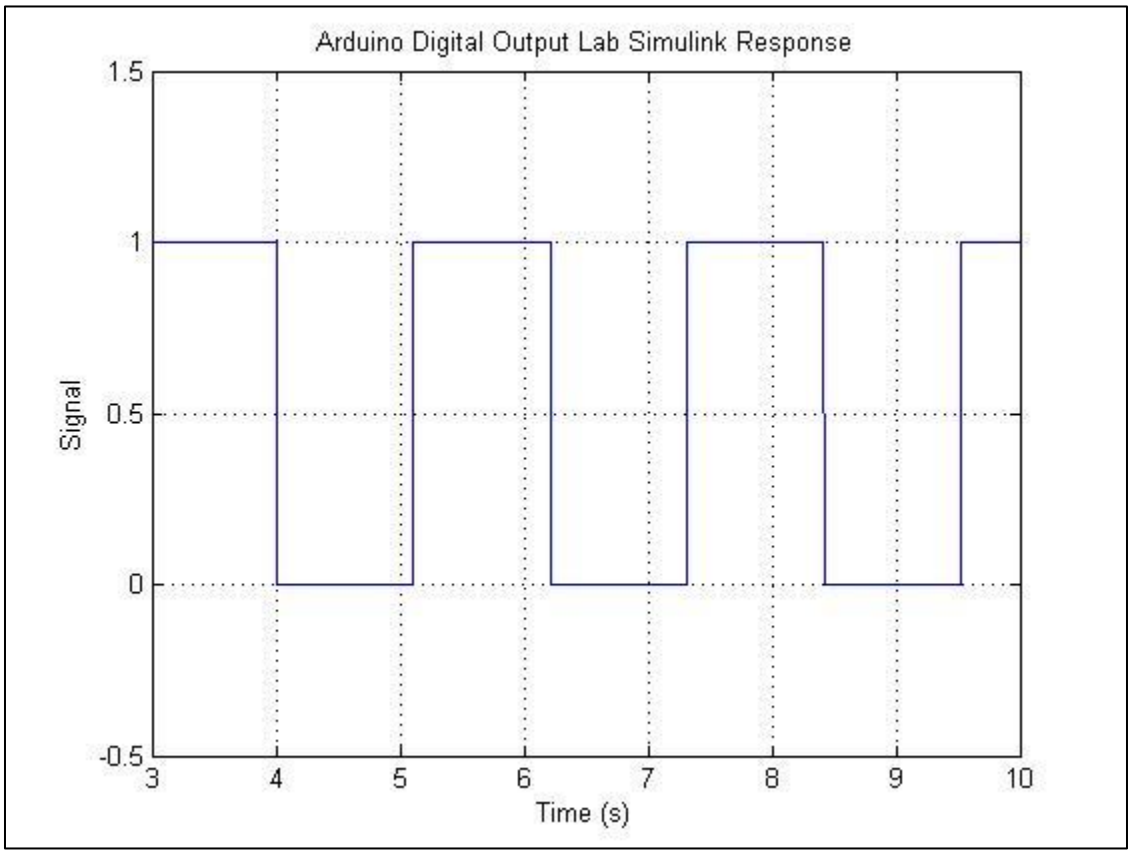

Figure 8: Arduino Digital Output Lab Simulink Response

Since it is difficult to control the blink duration by using a “for loop,” the student will learn how to use a Simulink source to command the blink. To accomplish this, the student will

use similar initializations as the Arduino Digital Output Lab, only they will not use a “for loop” to control the blink. Instead the user will send a message from Simulink to the Arduino in the form of a pulse with amplitude of one. This pulse can be easily created in Simulink by inserting the pulse generator block as the only input to the “Formatted Stream Write” block in the Simulink starter code, setting the amplitude to one, and customizing the period and duty cycle to create any blink pattern. These additions to the Simulink Diagram can be seen below.

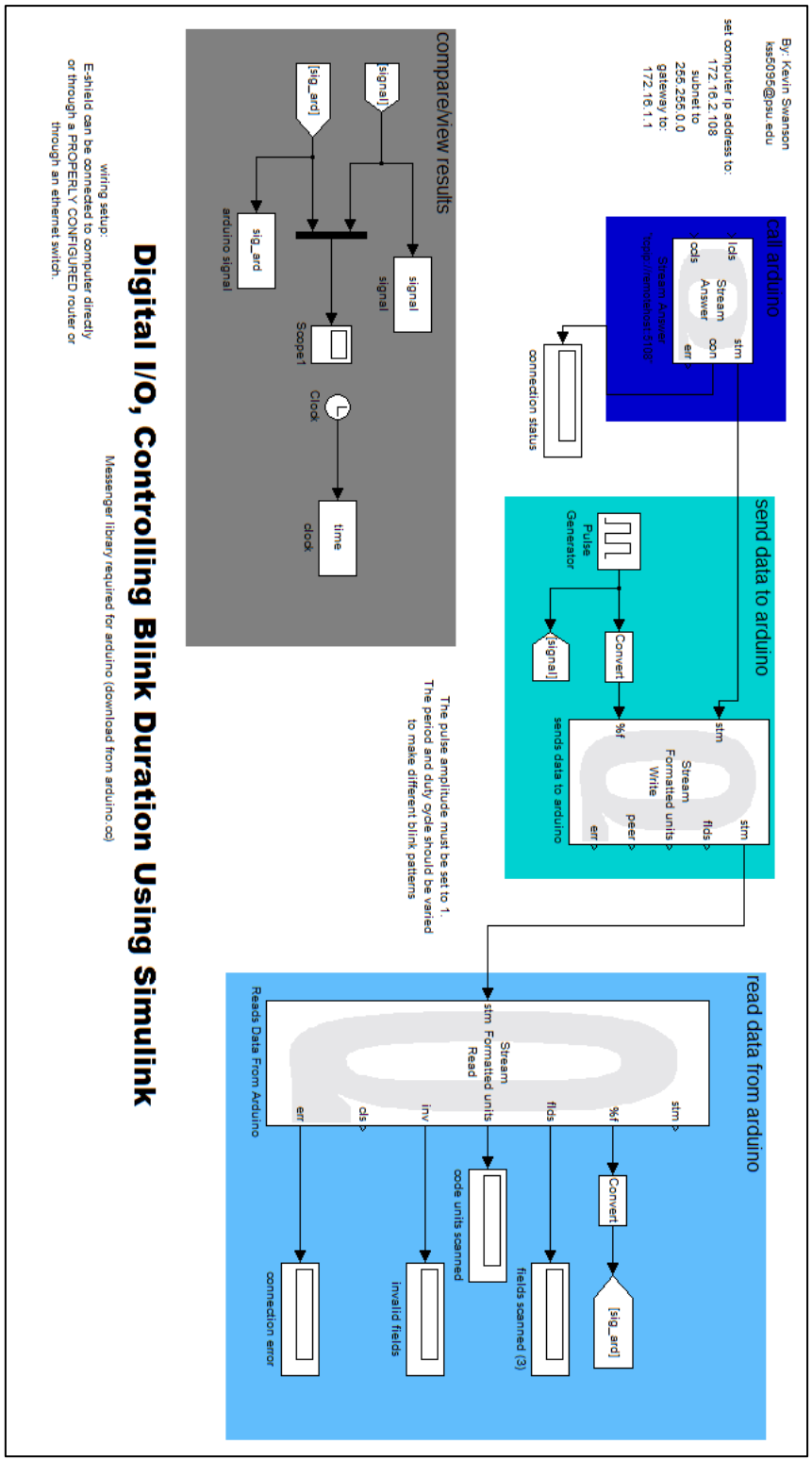

Figure 9: Arduino Digital I/O Lab Simulink Diagram

The Arduino starter code must be altered by inserting the “digital.Write()” function after the pulse value is read by Simulink. The code below shows how the Simulink data can be written to the LED pin after the Arduino reads it.

```
messageflag = 0;
  //first thing's first: read our message. this fills value into a 1x1 vector
  //the pulse signal is the only element in this vector

  while(client.available()){
    message.process(client.read());
  }

//this point needs to take the only element of the read data and
//output it to a pin on the Arduino.
  digitalWrite(blinkPin, value[0]);  //this writes the pulse value
  //to the LED pin
  Serial.println(value[0]);  //this writes the pulse value to the
  //serial port

 //now let's send our data back to QuaRC
```

The full Arduino code and Simulink diagram for the Arduino Digital I/O Lab can be seen in Appendix C.

Analog I/O Labs

There are three labs to teach analog I/O that involve reading a potentiometer and using that reading to control a servo. The first lab, much like the first digital lab, requires simply reading the analog signal from the potentiometer and displaying it graphically using Simulink. The second lab will use a sine wave signal sent from Simulink to control the motion of the servo. Finally, the third lab requires the student to read an analog signal from the potentiometer and use it to control the motion of the servo. All three of these labs require the hardware setup used in the digital I/O labs with the addition of a servo and potentiometer.

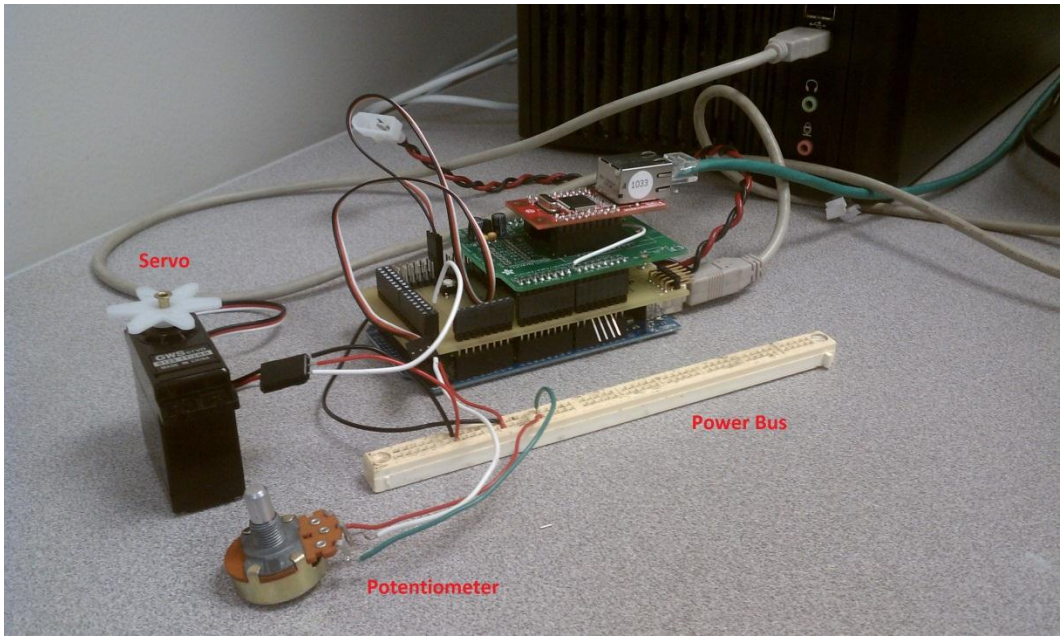

Figure 10: Hardware Setup for TCP/IP Communication with Potentiometer and Servo

The first analog lab is simpler than the first digital lab. All that is required for this is to initiate command the Arduino to read the analog signal from a potentiometer and send this signal to Simulink. The portion of the code for reading the potentiometer and sending it to Simulink is placed in the program's "void loop()" and can be seen below.

```
void loop()
{

  //first let's make sure we are connected
  // Connect to server, check for new data
if (!client.connected()){
  client.flush();
  Serial.println("problem!");
  client.stop();
  Serial.println("connecting...");
if (client.connect()){
  Serial.println("back online!");
}
}

  output = analogRead(potPin);  //read the analog value from the potentiometer
  client.println(output);  //write the signal to Simulink
  Serial.println(output);  //print the value to the serial monitor
  delay(10);

}
```

The Simulink diagram for this lab is almost exactly the same as for the Arduino Digital Output Lab. The final Arduino code and Simulink diagram for the Arduino Analog Output Lab can be seen in Appendix D.

The Arduino Analog I/O Lab not involve the potentiometer at all, but focuses on reading a signal from Simulink and using it to control the motion of a servo. The purpose of this lab is to expand on the Arduino Digital I/O Lab and show how more advanced peripherals than an LED can be controlled by Simulink. To begin the lab, first the servo must be initialized. A unique benefit of using an Arduino is the ability to utilize the Arduino's servo library [10]. Most microcontrollers require the user to control the servo with pulse width modulation (PWM), but the servo library enables the user to simply write a value from 0-179 to command the servo to turn to that angle. The servo library is opened with the "#include <Servo.h>" command. After the library is opened, the user must name their servo with the command "Servo myservo" and attaching it to a pin in the setup loop with "myservo.attach(22)". Attaching the servo to a pin enables the user to write 0-180 to that servo to command it to that position. On most Arduino boards the attach command removes PWM ability from pins 9 and 10. On the MEGA, however, using 12 or less servos does not disturb PWM functionality. The servo setup process can be seen in the code below, with the servo commands highlighted in red.

```
#include <Wire.h>
#include "Messenger.h"
#include <Ethernet.h>
#include <Servo.h>  //include the servo library
//ETHERNET CONFIG - set these values to correspond to your
//computer and Simulink
byte mac[] = { 0x1C, 0xBD, 0xB9, 0x1C, 0x1C, 0x01 };
byte ip[] = { 172, 16, 2, 108 };  //your Arduino
byte gateway[] = { 172, 16, 2, 1 };
byte subnet[] = { 255, 255, 0, 0 };
byte server[] = {172,16,1,108}; //your computer
//Server dataserver(5100);
int port = 5108; //should match the Simulink port
```

```
Client client(server,port);

Servo myservo;  //initialize the servo
int output = 0;  //initalize a value to hold the servo angle
int a = 0;  //random value to use flush function later

//now we need to set up a messenger 'object'
//default delimiter is the space character.
Messenger message = Messenger();

double value[2] = {0,0};  //this holds that values read by the Arduino
//in this case, it will only be the clock and sine signal
int messageflag = 0;//this tells the program whether we've gotten something

//now we need the callback funtion for messenger
void messageReady(){
 //loop through all the available elements of our message,
 //assigning our read values as we go.
 //in this case, there is only one value, so the loop is removed
 //and only value[0] is left

 for(int i=0;i<2;i++){
 if (message.available()){
  messageflag = 1;
  value[i] = message.readDouble();
  }
 }
}



void setup()
{
 myservo.attach(22);  //attach the servo to pin 22 (digital)
```

Once this initialization is complete, one needs to write the Simulink data – the servo position command - to the Arduino. The Simulink command is received by the Arduino and implemented as a change in the servo output by the following code.

```
 //we need to write the signal from Simulink to the servo
 output = value[1];  //this sets the second element (sine signal) to
 //be the output
 myservo.write(output);  //this writes that value to the servo
 Serial.println("The servo value is:");
 Serial.print(output);  //this writes the servo value to the
 //serial port
```

The complete Arduino code and Simulink diagram for the Arduino Analog I/O Lab can be seen in

Appendix E.

For the Arduino Analog Control Lab, the analog input and output code above can be combined to control the servo with the potentiometer. This lab is important because it is the first that requires the student to manipulate the data within Simulink. The Arduino programming for this lab is very straightforward; the basic idea is to combine the codes from the past two analog implementations. It is important for the data to be written to the servo before reading it back from the potentiometer. If the potentiometer value is read first then the program will be sending values from 0-1023 to the servo when the servo can only read values from 0-179. The data processing portion of the code should look like this:

```
//we need to write the signal from Simulink to the servo
output = value[1];  //this sets the second element (sine signal) to
//be the output
myservo.write(output);  //this writes that value to the servo
Serial.print("The servo value is:");
Serial.println(output);  //this writes the servo value to the
//serial port
value[1] = analogRead(potPin);  //read the value of the potentiometer
//and set it to the second element
Serial.print("The potentiometer reading is:");
Serial.println(value[1]);
```

This Arduino code will only transfer data to and from Simulink. Since the Arduino MEGA uses a 16 channel, 10-bit analog to digital converter (ADC) it will map an input voltage from 0-5V to integer values from 0-1023 [10]. The potentiometer wired to 5V and ground will send a value of 0-5V to the ADC which will convert this to a value from 0-1023 bytes. The servo can only receive values of 0 – 179 degrees, so the Simulink diagram must be able to convert the potentiometer data into degrees. The best way to accomplish this is to send the data into a subsystem where it can be processed and returned to the Arduino. This Simulink diagram, along with its subsystem, can be seen below.

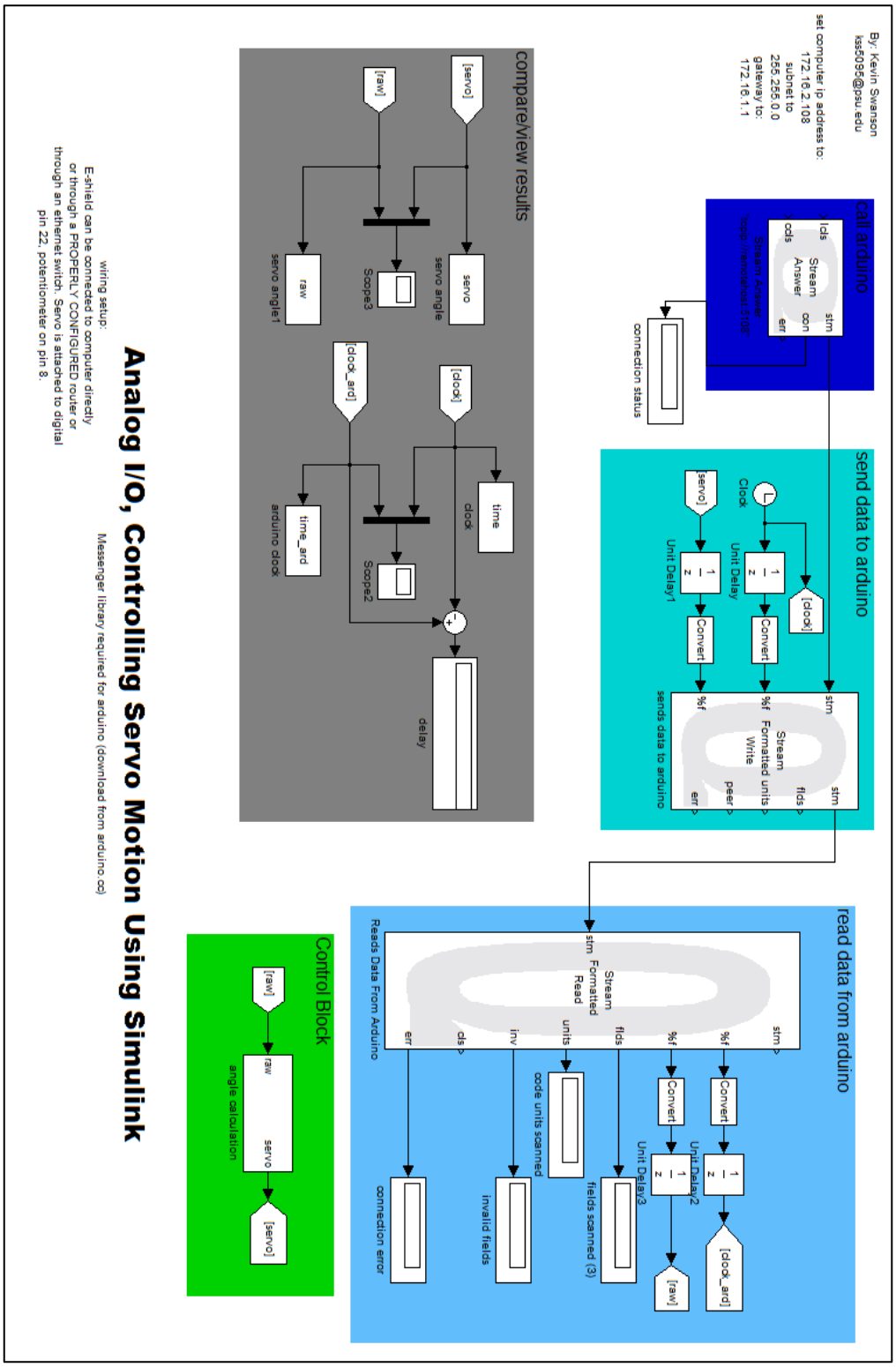

Figure 11: Arduino Analog Control Lab Simulink Diagram

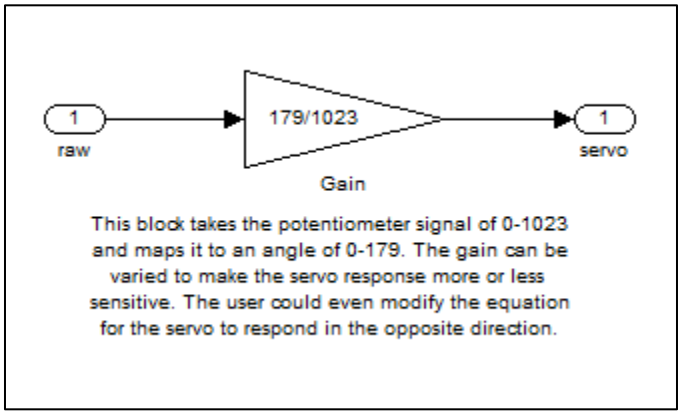

Figure 12: Data Conversion Subsystem

The complete Arduino code and Simulink diagram for the Arduino Analog Control Lab can be seen in Appendix F.

Digital I/O Using Serial Communication

The following code explores the ability to communicate between the Arduino and MATLAB via a serial connection. The first of these implementations performs the same task as the Arduino Digital Output Lab, only using a serial rather than Ethernet connection between the Arduino and PC.

The physical layout is simple; it is just an Arduino with no peripherals or additional boards connected to the PC via a serial to USB connection.

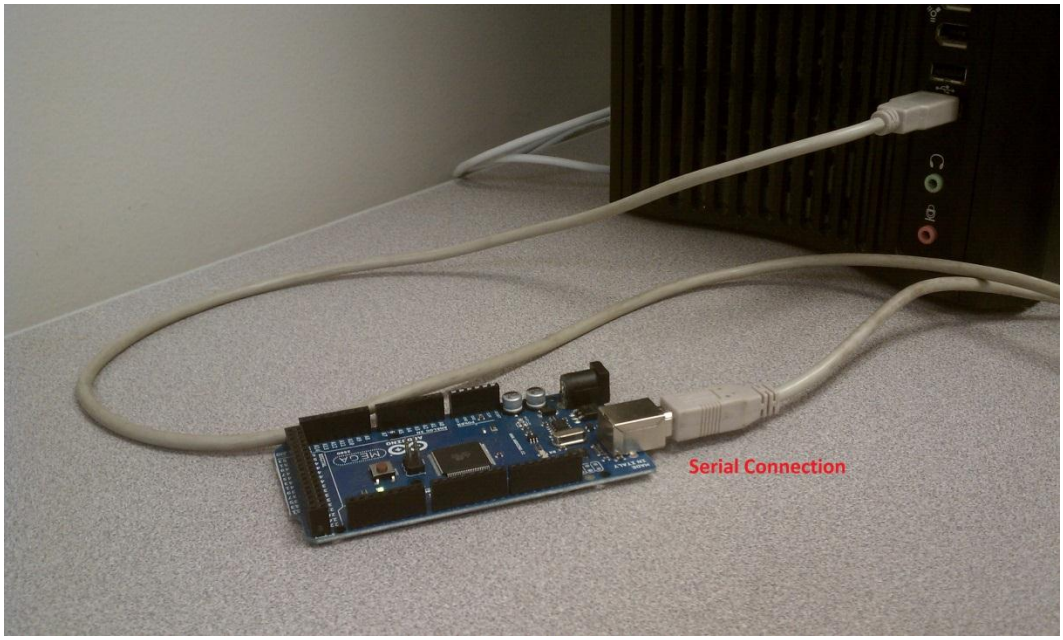

Figure 13: Hardware Setup for Serial Communication with no Additional Peripherals

Like in the previous digital implementation, one must again write an "if loop" to create the blink pattern in the Arduino code. But rather than initializing a TCP/IP client in the Arduino code, the serial setup is as easy as initiating the serial port in the setup loop with "Serial.begin(115200)" and printing the data to the serial port with "Serial.prinln(output)". The steps to initialize the serial port, create the blink pattern, and print the data can be seen in the Arduino code below.

```
int ledPin =  13;    // LED connected to digital pin 13
int output;    //initialize a value for the signal

void setup()   {
  Serial.begin(115200);    //open the serial port
  pinMode(ledPin, OUTPUT);    //set the pin as an output
}

void loop()
{
  for(int i=0;i<200;i++){  //this for loop sets the period of the
  //square wave. each period will be approximately 200 times the
  //10 ms delay, but there will still be a delay
  if(i<100){  //this sets the time that the signal will be held HIGH
    output = 1;  //pulls the signal HIGH
  }
  else{  //outside of the previous time
    output = 0;  //pull the signal LOW
```

```
 }
  digitalWrite(ledPin, output);  //write the signal to the LED
  Serial.println(output);  //write the signal to the serial port
  delay(10);
}
}
```

The programming for MATLAB is slightly more complicated and may be unfamiliar to the typical ME student. First, one must open the serial connection with the following MATLAB code:

```
clear all;
s1 = serial('COM12');                          %define serial port
s1.BaudRate=115200;                            %define baud rate
```

Unlike the Simulink programs that read data from an Ethernet connection, MATLAB does not indefinitely read values when receiving data from the serial port. Instead the user must initiate a certain number of points to be read. This can be easily performed by writing all of the read commands in a "for loop" with the counter set to the number of data points that should be read. In this implementation, it is only required that MATLAB read the digital data from the Arduino and plot it. To read the data, one could use the following code:

```
%this portion reads the arduino data - acquisition of 1000 points
for i= 1:1000;          %set number of times to run through code
    data=fscanf(s1);            %read sensor
    angle_data=str2double(data);  %convert the string to a double
    %plot the data
    plot(i,angle_data);
    hold all;
    title('Digital Output vs. Data Point');
    xlabel('Data Point');
    ylabel('Digital Output');
    axis([0 1000 -0.5 1.5]);
end
fclose(s1);                 %CLOSE THE SERIAL PORT!
```

It is important to note the "fclose(s1)" command after the for loop. It is extremely important to close the serial port after running the code. Leaving the serial port open will make that port

unusable until it is closed, which is not an easy task. If one experiences problems with an open serial port, close out of MATLAB and the Arduino program restart the programs. The full code for the MATLAB Serial Digital Read Lab can be seen in Appendix G.

The second of these serial interface programs shows how to send digital data to the Arduino. Instead of plotting the signal sent to MATLAB, this code creates a signal in MATLAB to control a peripheral on the Arduino. In this case one should write a blink program in MATLAB to tell an LED on the Arduino to blink. In the Arduino code one only needs to check if there is data in the serial port with "Serial.available()" and set the data equal to a variable with "output = Serial.read()." This output can then be written to any pin using the "digital.Write()" command. This complete code can be seen below.

```
int ledPin = 13;  // digital pin used to connect the led

void setup()
{
 Serial.begin(115200);  //open the serial port
}

void loop()
{
 int output = 0;
 if(Serial.available())
 {
  output = Serial.read();
  digitalWrite(ledPin, output);
 }
}
```

The MATLAB code is also simple. The only new command is "fwrite(s1,output)" which writes the value "output" to the serial port called "s1." Aside from this, the blinking pattern is created by sending a high signal, pausing, sending a low signal, pausing, and repeating this pattern. The code to perform the blink pattern can be seen below.

```
%this loop sends an on/off blink pattern to the Arduino
for i= 1:5;                    %set number of blinks
    output = 1;                %set output high
```

```
    fwrite(s1,output);          %print the output to the serial port
    pause(1);                   %wait a second
    output = 0;                 %repeat for low
    fwrite(s1,output);
    pause(1);
end
fclose(s1);                   %CLOSE THE SERIAL PORT!
```

Again, it is very important to close the serial port. The full code for the MATLAB Serial Digital I/O Lab can be seen in Appendix H.

Analog I/O Using Serial Communication

The final serial interface lab tests the sending and receiving of analog data to MATLAB, and tests the ability to plot the results in "real time." To accomplish this, the user will once again control a servo with a potentiometer, and then send this commanded angle to MATLAB to be plotted. The hardware setup is simply an Arduino Mega connected with a potentiometer and servo connected.

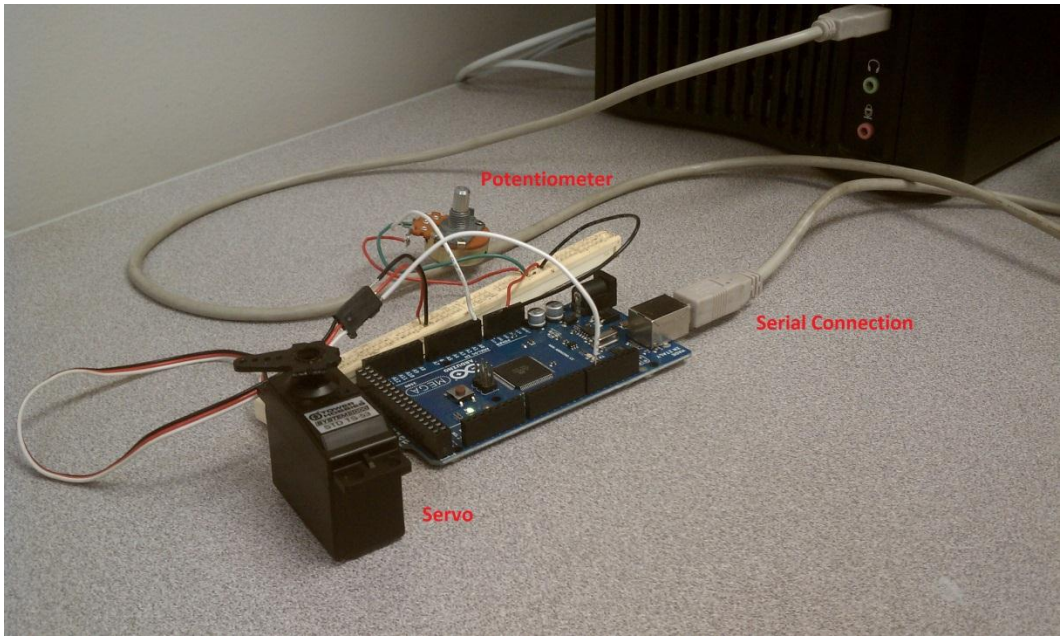

Figure 14: Hardware Setup for Serial Communication with Potentiometer and Servo

The Arduino code for this is very simple as the user only needs to run the basic servo control program and write the output to both the servo and the serial port. The Arduino code can be seen below.

```
#include <Servo.h>

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

int potpin = 0;  // analog pin used to connect the potentiometer
int val = 0;    // variable to read the value from the analog pin
int angle = 0;  //variable to hold the calculated angle

void setup()
{
 Serial.begin(115200);  //open the serial port
 myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop()
{
 val = analogRead(potpin);           // reads the value of the potentiometer (value between 0 and
1023)
 angle = map(val, 0, 1023, 0, 179);     //convert to angle
 Serial.println(angle);               // send data to and from MATLAB
 myservo.write(angle);                // sets the servo position according to the scaled value
}
```

The MATLAB code is not as trivial. Plotting the data can be accomplished in two ways. One is to simply write the plotting commands in the “for loop” and add a “pause()” function at the end to allow the program to plot the data. In this method the program will read the data from the serial port and plot the data during each iteration of the loop. While this is a simpler method it is slow because it requires the program to redraw the entire plot during each iteration. It is unnecessary to redraw the entire plot because specific aspects of the plot such as the title, axis labels, and legend do not change.

To improve on this plotting method it is better to use a “handle” pointer, which tells the program to update only parts of the plot (in this case the y value) instead of updating every feature. This dramatically increases the plotting speed. When using the simpler plotting method

the user must use the “hold on” command after each plot is made to keep all previous values on the plot during each iteration of the loop. In handle method the “hold on” command is used in the initialization and does not appear in the loop, so the previous y value is replaced by the new one in each iteration. For this reason, it is necessary to create an array of values that contains as many values as there will be plotted points, and update the plot with this array as the y value instead with of a single number. This array can be created with the following initialization code.

```
counts = 2500;                      %define number of points
angle = 0;                          %define angle
angle_data = 0;                     %define angle_data
final_angle = zeros(counts,1);      %make a matrix for angle
seconds = zeros(counts,1);          %make a matrix for seconds
```

After initializing these values the code must store the handle of the plot with the following code.

```
HandleAngle = plot(seconds,final_angle);  % Keep the handle for
%this plot
hold on
title('Commanded Angle vs Time')
xlabel('MATLAB Time (s)')
ylabel('Commanded Angle (deg)')
axis([0 15 0 180])
set (HandleAngle,'Erase', 'xor')  % Set these so it updates
```

The final step is to update the handle within the for loop by using the command “set(HandleAngle,'YData',final_angle).” This command can be seen in the following loop.

```
for i= 1:counts
    seconds(i)= toc;                %set seconds matrix
    angle = fscanf(s1);             %read serial port
    angle_data = str2double(angle); %convert the string to a
    %double
    final_angle(i) = angle_data;    %set current angle value
    %plot the data
    set(HandleAngle,'YData',final_angle)  %change just the y data
    %for plot
    set(HandleAngle,'XData',seconds)      %change just the x data
    %for plot
    drawnow
end
fclose(s1);                 %CLOSE THE SERIAL PORT!
```

The full code for the MATLAB Serial Analog I/O Lab can be seen in Appendix I.

Chapter 4

Results

The series of labs that implement QuaRC for communication between the Arduino and Simulink have many advantages that will help Mechanical Engineers expand their knowledge of Mechatronics. By starting with basic data transfer between the Arduino and Simulink and expanding with more complicated signals and control algorithms, the students will better be able to gain an understanding of Mechatronics and increase the usefulness of their systems. While the initial code that enables the user to communicate between the Arduino and Simulink is complicated, this only needs to be explained once. After an introductory lesson, a typical student should be able to easily manipulate the starter code to perform whatever type of data acquisition is necessary.

There are two major benefits that exist when using Simulink in the DAQ system. The main benefit is simply the fact that Simulink is a visual programming language and manipulating the controls structure of the data acquisition system is much easier in Simulink than in basic C code. The other benefit is the ability of Simulink to plot data in real time during an experiment. By simply adding scopes to the Simulink diagram, the user can monitor the inputs and outputs of the system.

As with all real systems, there is a feedback delay between the data sent to the Arduino and the data received by Simulink. The delay is monitored in the real time in the compare/view results block by subtracting the Simulink clock from the Arduino clock and sending this signal to a display block. Minimizing the feedback delay was a major focus in the creation of these labs. Initially, the program actually runs with a very small delay of about 0.01 to 0.03 seconds. Unfortunately, after running for an extended time the Arduino memory fills with unused bytes so

it can no longer send response data and the DAQ system fails. In the Arduino Analog I/O Lab, for example, the system fails after only 53 seconds. This system response can be seen in the plot below.

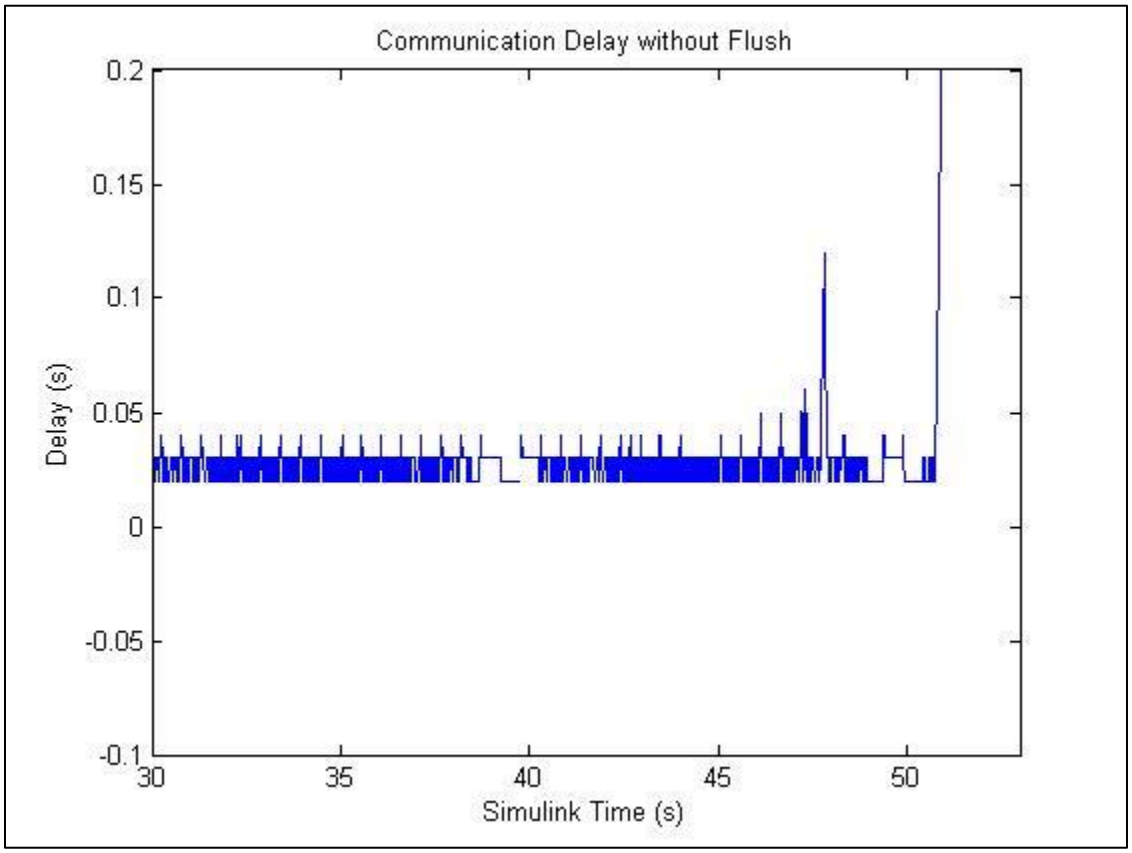

Figure 15: Communication Delay without Flush

While the response time of the system is very good, it is not very useful to have a DAQ system that fails after only 53 seconds. With help from Alex Brown, a Graduate Student in Penn State's Intelligent Vehicles and Systems Group, along with some trial and error we found that the best method for preventing this signal failure is to implement a "client.flush()" command to empty the memory. When this command is implemented it causes the feedback delay to increase dramatically to 0.1 to 0.4 seconds. This increase can clearly be seen in the plot below.

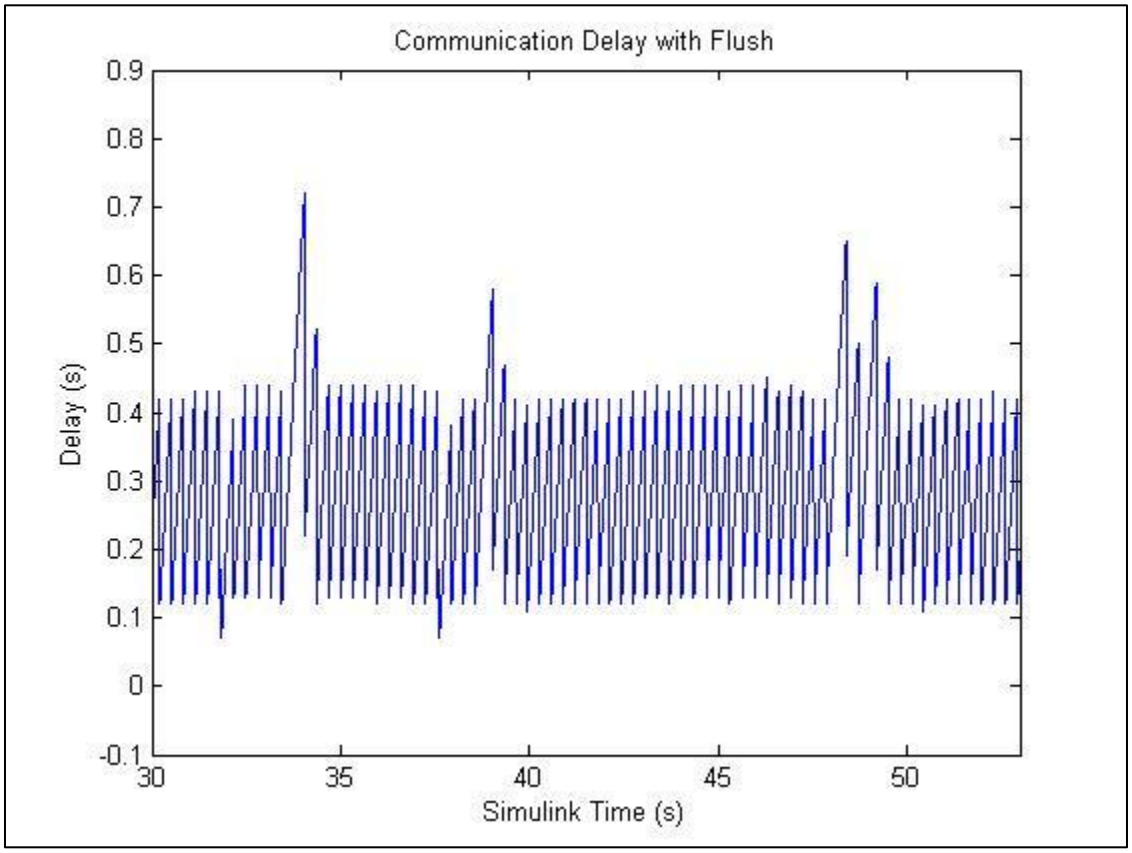

Figure 16: Communication Delay with Flush

It is unfavorable to choose either a system that is very responsive but fails after about one minute, or one with an extremely slow response, so it is necessary to implement a code that both flushes the system and allows a fast response. A balance that gives the best of both codes can be achieved by limiting the "client.flush()" command to running every 1000 cycles. This is accomplished by the code below.

```
if(a == 1000){
client.flush();  //this clears out the values to make room for
//more. it increases the comm delay drastically, so it is only
//used every 1000 cycles.
a = 0;
}
a++;
```

As seen in the plot below this code allows the system to run with a delay of only 0.01 to 0.03

seconds and prevents failure.

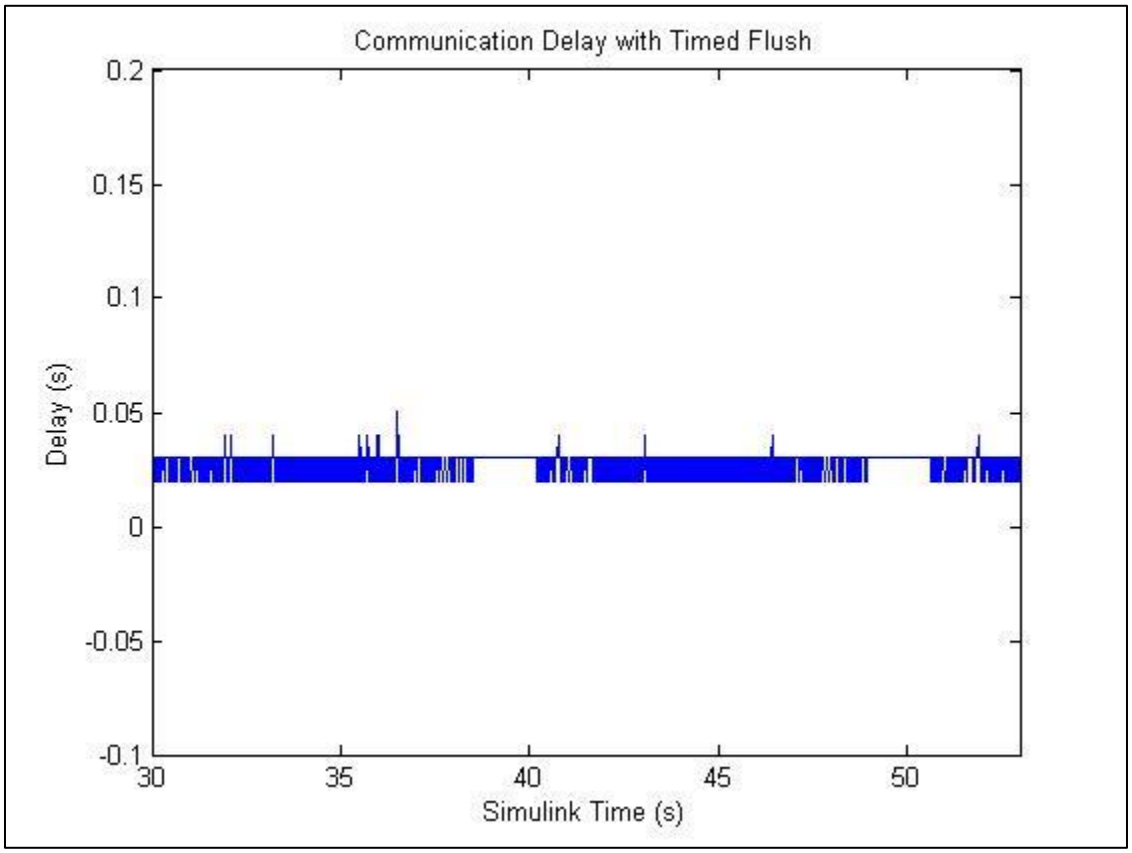

Figure 17: Communication Delay with Timed Flush

This plot clearly shows the improvement in system response. This method of data collection is a dramatic improvement on the past two systems and should be implemented in all TCP/IP to Arduino connections.

Despite the many advantages of the Ethernet connection method there are still some major problems with its implementation. These problems all involve the additional hardware and software needed to implement this system. Since the hardware setup itself requires four circuit boards it is very difficult to debug hardware problems. Additionally, there are three programs that must run to operate this system (MATLAB, Simulink, and QuaRC). Even though these programs are all linked to one another problems can arise at any level.

One major problem that consistently arises when running a DAQ system through QuaRC

is a connection problem. Even when all pieces of hardware are physically connected correctly and the software is programmed with the correct IP address there is still the possibility of connection failure in QuaRC. One possible reason for this problem is interference from other network connections. These labs were run on multiple PCs and some functioned better than others. When the labs were run on PCs belonging to Penn State University there were many connection problems. These computers are connected to Penn State's network and have firewall protection to protect this network from harm. Connection problems were much less prevalent on non-University owned equipment. On these PCs it is possible to disable firewall protection and network connections. By doing this the connection problems noticeably decreased.

Overall, the labs that use an Ethernet connection to communicate between the Arduino and Simulink achieve the goals of this thesis. They do provide a straightforward platform for programming along with an ability to plot data in real time. The implementation of these labs will benefit Mechanical Engineering students, but more work will have to be done to debug the connection problems that exist in this system.

The labs that implement a serial connection between the Arduino and MATLAB achieve some goals of the project. MATLAB language is very similar to C code, so programming through this method does not give the student the benefit of visual programming like in Simulink. Despite this, the serial connection provides a platform for plotting data without the complicated connections involved in the Simulink labs. The initial learning curve of these labs is much lower than in the Simulink labs because the starter codes are not required. All that is needed is a few lines of code that command the program to read from or write to the serial port.

Since communicating with a serial communication through MATLAB is very simple this system can be very beneficial for plotting data. These labs show that it is moderately easy to plot data from the Arduino. It is noticeable, however, that the data does not seem to plot in real time. In the example below, the potentiometer is turned when the plot reaches about two seconds, but

the data does not respond until about seven seconds.

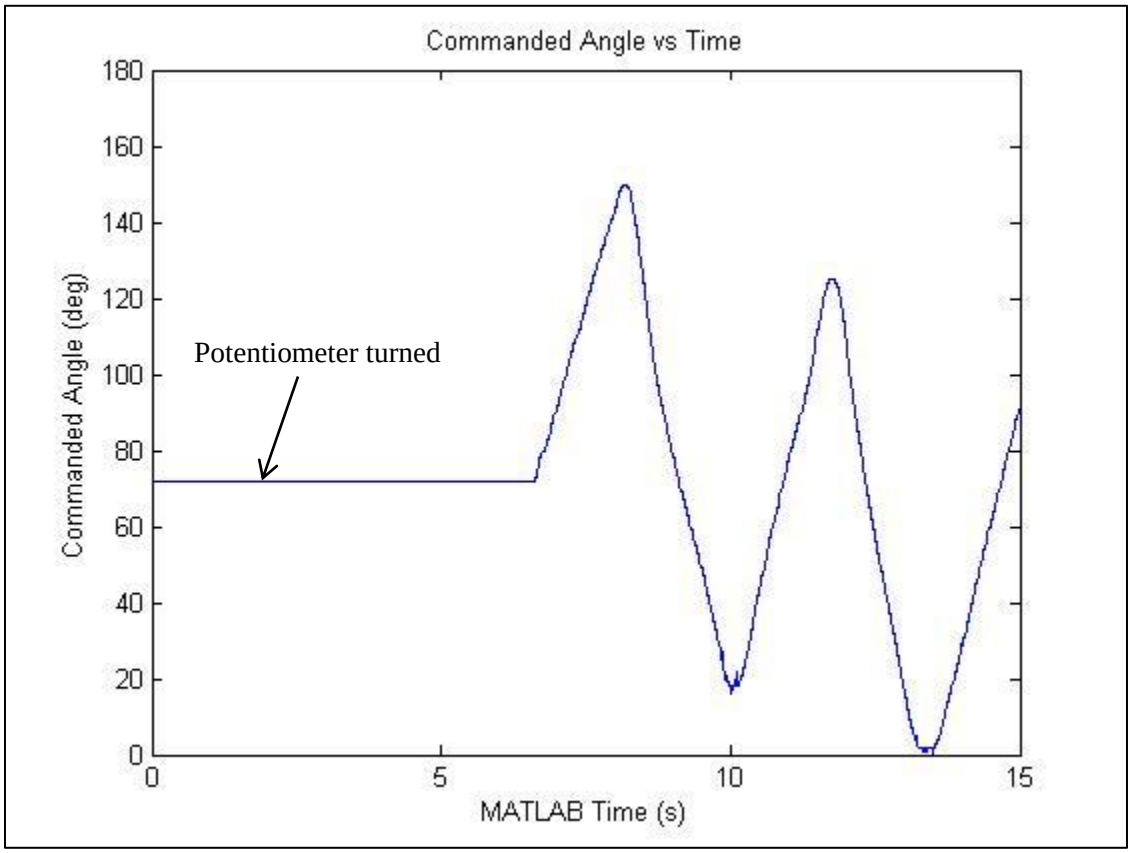

Figure 18: Plotting Delay in MATLAB Serial Analog I/O Lab

This means that either the program does not read data in real time, or it does not plot in real time. To find out whether or not the data is read in real time one must simply send time to the Arduino, have the Arduino send this time back, and calculate a delay from the two time values. The Arduino and MATLAB codes for plotting this delay can be seen below.

Arduino code:

```
void setup()
{
  Serial.begin(115200);  //open the serial port
}

void loop()
{
  double time = 0;
  if(Serial.available())
```

```
 {
 time = Serial.read();       // read data from MATLAB
 Serial.println(time);       // send data to MATLAB
 }
}
```

MATLAB code:

```
% Author: Kevin Swanson
% Program for Arduino MEGA
% Reads angle from example servo program and plots commanded
%angle
close all;
clear all;
clc;
s1 = serial('COM18');                %define serial port
s1.BaudRate=115200;                  %define baud rate
counts = 500;                        %define number of points
seconds = zeros(counts,1);           %make a matrix for seconds
seconds_ard = zeros(counts,1);       %make a matrix for arduono
seconds

%open serial port
fopen(s1);
clear data;

%create a delay while the port opens
val=0;
while(val<10000000)
    val=val+.03;
end

delay = seconds - seconds_ard;       %find delay
HandleTime = plot(seconds,delay);    %keep the handle for this
%plot
hold on
title('MATLAB Time vs. Arduino Time')
xlabel('MATLAB Time (s)')
ylabel('Delay (s)')
axis([0 10 -.1 .1])
set (HandleTime,'Erase', 'xor')  %set these so the handle updates

tic     %start the timer
%this portion reads the arduino data - acquisition of # counts
for i= 1:counts
    seconds(i)= int8(toc);           %set seconds matrix as int
    fwrite(s1,seconds(i));           %write time to arduino
    data = fscanf(s1);               %read serial port
    data_double = str2double(data);  %convert the string to a
    %double
    seconds_ard(i) = data_double;    %set current time value
```

```
        delay = seconds - seconds_ard;    %update delay
        %plot the data
        set(HandleTime,'XData',seconds)  %change just the x data for
        %plot
        set(HandleTime,'YData',delay) %change just the y data for
        %plot
        drawnow
end
        toc     %close timer
        fclose(s1);                 %CLOSE THE SERIAL PORT!
```

The problem that arises in these codes is that the Arduino command "Serial.read()" only reads int values. Because of this time values sent to and from the Arduino will be integers while the MATLAB time will be a double. For this reason, the delay is plotted two ways. Once with the MATLAB time as a double, and once with the MATLAB time converted to an integer.

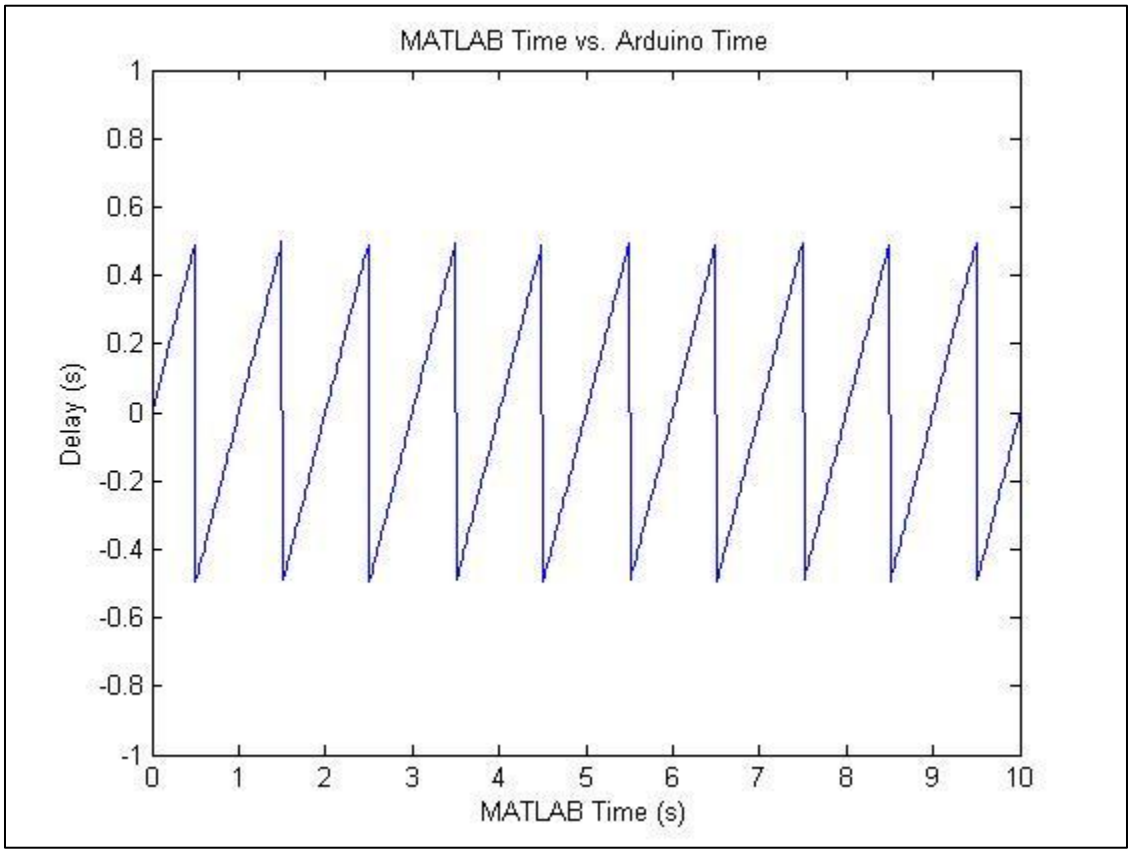

Figure 19: MATLAB Serial Analog I/O Lab Delay - Time as Double

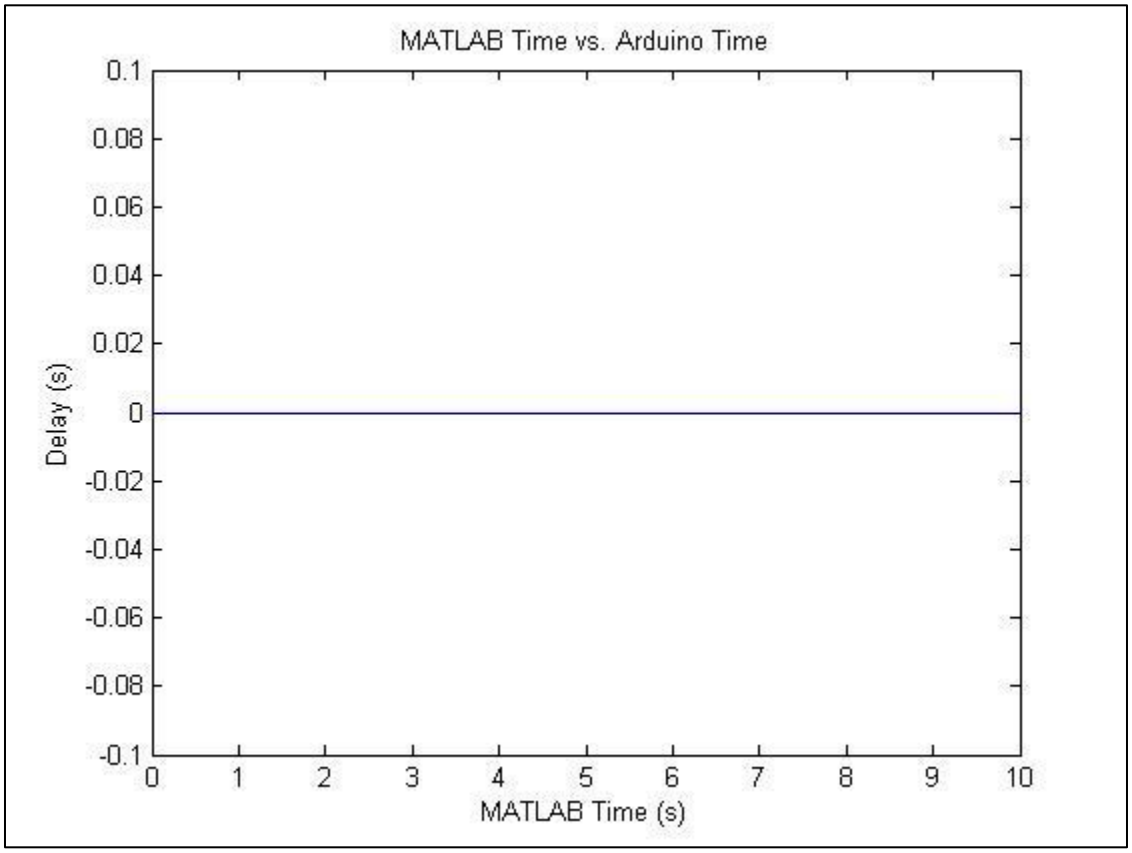

Figure 20: MATLAB Serial Analog I/O Lab Delay – Time as Integer

With time as a double there is about a one second delay, which makes sense because that is how long it will take the integer second to change to the next second value. With time as an integer there is zero delay. These two graphs show that the five second delay apparent in Figure 18 is not caused by a data transfer delay, but is instead caused by slow plotting. Using the "handles" pointer for plotting instead of a standard plot command improved this plotting delay in the MATLAB Serial Analog I/O Lab, but further investigation could still improve the plot time.

Overall, the simplicity of serial communication between the Arduino and MATLAB makes it beneficial for Mechanical Engineers learning Mechatronics, but some problems still exist. As tasks become more complicated the required commands become increasingly more complex. MATLAB also has numerous limitations such as the inability to send the Arduino values other than an int. Finally, since the slow plotting speed prevents the programs from

reaching its potential for “real time” data acquisition more investigation is needed before using MATLAB to plot real time data.

Conclusions

The Arduino to Simulink labs will be successful in improving Mechatronics Education for Mechanical Engineers because of both its user friendly visual programming language and its ability to plot data in real time. Even though this system is more expensive and complicated than a serial connection due to the required additional equipment, the benefits are significant. Currently, students are required to perform in-depth programming with C to accomplish all tasks. By implementing the Simulink labs, the students will be able to only use simple commands in the Arduino code while accomplishing more difficult control codes with block diagrams. Also, the ability to add scopes to plot all data in real time allows the student to better understand how their system is working and debug errors. Since connection problems do exist users must be aware and should look into the causes of these problems to generate possible solutions.

While the Arduino to MATLAB labs do not offer students the benefit of visual programming it does provide a straightforward medium for plotting data. Students should be able to use this serial connection as a starting point for data acquisition, but should be careful when plotting data due to very slow plotting speeds. As with the Simulink labs, students must be aware of the plotting problems and look into solutions to improve the system.

Further Recommendations

There are two main additions that could be made to these labs. The first addition should be to improve the plot time of the Arduino to MATLAB labs. This could potentially be performed

by implementing a timer object in the MATLAB code. The timer object will replace the pause function currently in the code. The timer object will basically be used to set a schedule for the data to be plotted. Unfortunately this function was unsuccessfully implemented in this thesis and will need to be looked into further in future projects.

Another addition would be to implement wireless communication between the Arduino and PC. This would be beneficial because the user could access and process data on the Arduino remotely. Christopher McNally implemented a wireless connection using an Arduino and MATLAB in his graduate design project [11]. McNally's system uses a WiShield, which is an Arduino shield for the Arduino Diecimila, Duemilanove, and Uno that provides wireless 802.11b connectivity. The project shows that it is possible to implement such a system, but the appropriate supplies would need to be purchased before it could be implemented into an Arduino to Simulink lab.

Bibliography

[1] V. Giurgiutiu, J. Lyons, D. Rocheleau, W. Liu. "Mechatronics/microcontroller education for mechanical engineering students at the University of South Carolina." *Mechatronics 15 (2005)*. pp. 1025-1036. May 2005.

[2] V. Giurgiutiu, W. Liu. "The use of Functional Modules in the Mechatronics Education of non-Electrical Engineering Students," presented at the International Conference on Engineering Education and Research "Progress Through Partnership," Ostrava, Czech Republic, 2004.

[3] O. Erdener. "Development of a Mechatronics Education Desk." M.S. thesis, The Middle East Technical University, Çankaya Ankara Provine, Turkey, December 2003.

[4] P. Osolnick. "'Devastating' cuts loom as governor unveils budget proposal." *The Daily Collegian* (March 14, 2011), sec A.

[5] S. Brennan. ME597D. Personal Interview, Topic: Development of the advanced mechatronics class. Professor of Mechanical Engineering, Penn State University, State College, Pennsylvania, March 26, 2011.

[6] PIC 16F690 Product Page. Internet: http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=PIC16F690-I/P-ND, March 27, 2011 [March 27, 2011].

[7] Arduino MEGA Product Page. Internet: http://www.adafruit.com/index.php?main_page=product_info&products_id=191&z%20enid=b508a%2092a5f41, [March 27, 2011].

[8] WIZnet Product Page. Internet: http://www.sparkfun.com/products/9473, [March 27, 2011].

[9] Adafruit Ethernet Shield Product Page. Internet: http://www.adafruit.com/index.php?main_page=product_info&products_id=83&zenid=b229c699f6965b226e90af98172d1000, [March 31, 2011]

[10] Arduino – HomePage. Internet: http://arduino.cc/en/, [April 17, 2011]

[11] C. McNally. "Arduino Based Wireless Power Meter." M.S. design project, Cornell University, Ithaca, NY, May 2010.

Appendix A – Starter Arduino Code and Simulink Diagram

The following is the starter code for reading sending data between the Arduino and Simulink.

```
#include <Wire.h>
#include "Messenger.h"
#include <Ethernet.h>
//ETHERNET CONFIG
byte mac[] = { 0x1C, 0xBD, 0xB9, 0x1C, 0x1C, 0x06 };
byte ip[] = { 172, 16, 2, 106 };
byte gateway[] = { 172, 16, 2, 1 };
byte subnet[] = { 255, 255, 0, 0 };
byte server[] = {172,16,1,106};
//Server dataserver(5100);
int port = 5106;
Client client(server,port);

//now we need to set up a messenger 'object'
//default delimiter is the space character.
Messenger message = Messenger();

double value[3] = {0,0,0};
int messageflag = 0;//this tells the program whether we've gotten something

//now we need the callback funtion for messenger
void messageReady(){
  //loop through all the available elements of our message,
  //assigning our read values as we go.

  for(int i=0;i<3;i++){
  if (message.available()){
    messageflag = 1;
    value[i] = message.readDouble();
    }
  }
}



void setup()
{
  Serial.begin(115200);
  Serial.println("hello!");
  Serial.println("My IP address is:");
  Serial.print(ip[0],DEC); Serial.print("."); Serial.print(ip[1],DEC); Serial.print(".");
Serial.print(ip[2],DEC); Serial.print("."); Serial.println(ip[3],DEC);
  Serial.println("I would like to connect to Simulink on computer:");

Serial.print(server[0],DEC);Serial.print(".");Serial.print(server[1],DEC);Serial.print(".");Serial.print(server[
2],DEC);Serial.print(".");Serial.println(server[3],DEC);
  Serial.print("on port number: ");Serial.println(port);
  // start Ethernet
```

```
  Ethernet.begin(mac,ip,gateway,subnet);
  //server.begin();
  //now we must attach that callback funtion to the messenger object
  message.attach(messageReady);
if (client.connect()){
  Serial.println("connected");
}
else{
  Serial.println("connection failed");
}
}

void loop()
{

  //first let's make sure we are connected
    // Connect to server, check for new data
if (!client.connected()){
  client.flush();
  Serial.println("problem!");
  client.stop();
  Serial.println("connecting...");
if (client.connect()){
  Serial.println("back online!");
}
}
  messageflag = 0;
  //first thing's first: read our message. this fills value into a 3x1 vector

  while(client.available()){
    message.process(client.read());
  }

 //now let's send our data back to QuaRC
 if (messageflag ==1){
    for( int i=0;i<3;i++){
    //print each value[] element
    client.print(value[i]);
    //delimit with a space
    client.print(" ");
  }
  delayMicroseconds(10);
  //end the message with a carriage return
  client.println();
  delayMicroseconds(10);
  messageflag = 0;
  client.flush();
  }

}
```

To run this program, the following hardware code must be saved in the same folder.

```
/*************************************************************
 *
 * Pennsylvania State University - Robotics Club
 * Learn more at www.psurobotics.org
 * Protected by the GNU General Public License
 *
 * This source file is developed and maintained by:
 * + Rich Mattes rjm5066@psu.edu
 *
 * File: Hardware.pde
 * Desc: Provides the hardware-related functions for the Mechbot
 * platform.
 *
 *************************************************************/


//ADDED FOR COMPATIBILITY WITH WIRING
extern "C" {
  #include <stdlib.h>
}

#include "WProgram.h"
#include "Messenger.h"



Messenger::Messenger()
{
	init(' ');
}

Messenger::Messenger(char separator)
{
	if (separator == 10 || separator == 13 || separator == 0)  separator = 32;
	init(separator);
}

void Messenger::init(char separator)
{
	callback = NULL;
	token[0] = separator;
	token[1] = 0;
	bufferLength = MESSENGERBUFFERSIZE;
   bufferLastIndex = MESSENGERBUFFERSIZE -1;
   reset();
}

void Messenger::attach(messengerCallbackFunction newFunction) {
	callback = newFunction;
}
```

```
void Messenger::reset() {
	bufferIndex = 0;
	messageState = 0;
	current = NULL;
	last = NULL;
	dumped = 1;
}

int Messenger::readInt() {

   if (next()) {
		dumped = 1;
		return atoi(current);
    }
 return 0;

}

// Added based on a suggestion by G. Paolo Sanino
long Messenger::readLong() {

		if (next()) {
			dumped = 1;
			return atol(current); // atol for long instead of atoi for int variables
		}
		return 0;

}

char Messenger::readChar() {

  if (next()) {
		 dumped = 1;
		 return current[0];
 }
  return 0;

}

double Messenger::readDouble() {
   if(next()) {
      dumped = 1;
      return atof(current);
   }
   return 0;
}

void Messenger::copyString(char *string, uint8_t size) {

		if (next()) {
			dumped = 1;
			strlcpy(string,current,size);
		} else {
			if ( size ) string[0] = '\0';
```

```
		}
}

uint8_t Messenger::checkString(char *string) {

		if (next()) {
			if ( strcmp(string,current) == 0 ) {
				dumped = 1;
				return 1;
			} else {
				return 0;
			}
		}
}



uint8_t Messenger::next() {

  char * temppointer= NULL;
  switch (messageState) {
  case 0:
    return 0;
  case 1:
    temppointer = buffer;
    messageState = 2;
  default:
    if (dumped) current = strtok_r(temppointer,token,&last);
    if (current != NULL) {
		dumped = 0;
		return 1;
		}
  }

  return 0;
}

uint8_t Messenger::available() {

  return next();

}


uint8_t Messenger::process(int serialByte) {
     messageState = 0;
    if (serialByte > 0) {

      switch (serialByte) {
      case 0:
		break;
      case 10: // LF
        break;
      case 13: // CR
```

```
      buffer[bufferIndex]=0;
      reset();
      messageState = 1;
      current = buffer;
      break;
    default:
       buffer[bufferIndex]=serialByte;
       bufferIndex++;
        if (bufferIndex >= bufferLastIndex) reset();
      }
    }
    if ( messageState == 1 && callback != NULL) (*callback)();
    return messageState;
 }


#ifndef Messenger_h
#define Messenger_h
#define MESSENGERBUFFERSIZE 64

#include <inttypes.h>

extern "C" {
// callback function
   typedef void (*messengerCallbackFunction)(void);
}


class Messenger
{

public:
 Messenger();
  Messenger(char separator);
  int readInt();
  long readLong(); // Added based on a suggestion by G. Paolo Sani
  char readChar();
  double readDouble(); // Added based on a suggestion by Lorenzo C.
  void copyString(char *string, uint8_t size);
  uint8_t checkString(char *string);
  uint8_t process(int serialByte);
  uint8_t available();
  void attach(messengerCallbackFunction newFunction);

private:
  void init(char separator);
  uint8_t next();
  void reset();

  uint8_t messageState;

  messengerCallbackFunction callback;

  char* current; // Pointer to current data
```

```
  char* last;

  char token[2];
  uint8_t dumped;

  uint8_t bufferIndex; // Index where to write the data
  char buffer[MESSENGERBUFFERSIZE]; // Buffer that holds the data
  uint8_t bufferLength; // The length of the buffer (defaults to 64)
  uint8_t bufferLastIndex; // The last index of the buffer
};

#endif
```

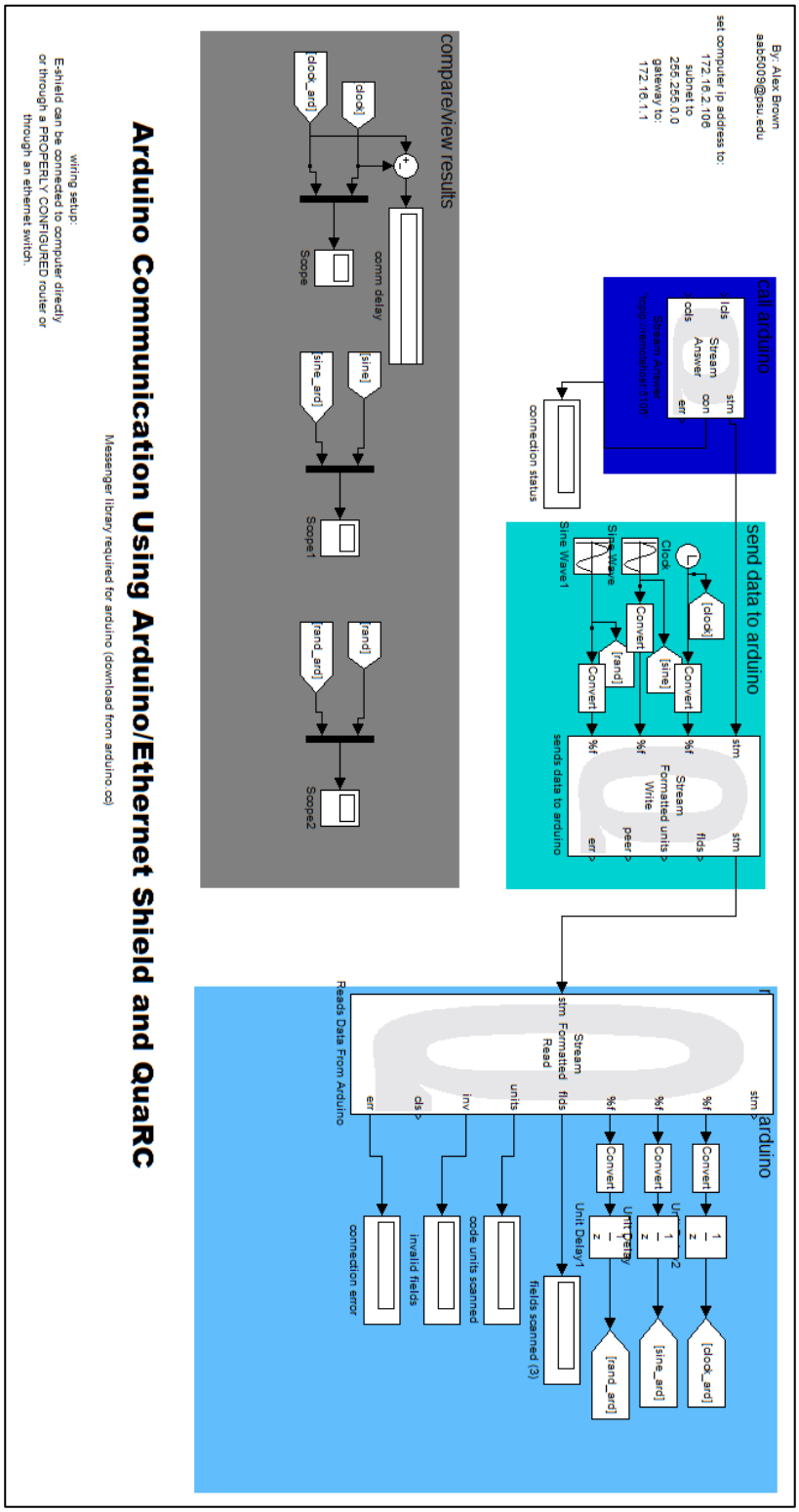

Figure 21: Starter Simulink Diagram

Appendix B – Lab 1: Arduino Digital Output

The following is the Arduino code for the first lab, not including the hardware setup.

```
/*****************************************************************
 * The majority of this code is taken from Dr. Brennan's ME597D
 * class. It provides a base code for communication with
 * Simulink. Much of the code is simply serial output to confirm
 * the connection with Simulink.
 *
 * All code that involves reading or writing values to the
 * Arduino are written by Kevin Swanson for his undergraduate
 * thesis.
 *
 * Special thanks to Alex Brown for his assistance and Rich
 * Mattes for the hardware code.
 *
 *****************************************************************/

#include <Wire.h>
#include "Messenger.h"
#include <Ethernet.h>
//ETHERNET CONFIG
byte mac[] = { 0x1C, 0xBD, 0xB9, 0x1C, 0x1C, 0x01 };
byte ip[] = { 172, 16, 2, 106 };
byte gateway[] = { 172, 16, 2, 1 };
byte subnet[] = { 255, 255, 0, 0 };
byte server[] = {172,16,1,106};
//Server dataserver(5100);
int port = 5106;
Client client(server,port);

int ledPin = 13; //designate the pin for your LED

//now we need to set up a messenger 'object'
//default delimiter is the space character.
Messenger message = Messenger();

int output = 0; //initialize a variable for the output state of the LED

//this loop prints the connection status to the serial port
void setup()
{
  pinMode(ledPin, OUTPUT);  //initialize the state of the LED pin
  Serial.begin(115200);
  Serial.println("hello!");
  Serial.println("My IP address is:");
  Serial.print(ip[0],DEC); Serial.print("."); Serial.print(ip[1],DEC); Serial.print(".");
Serial.print(ip[2],DEC); Serial.print("."); Serial.println(ip[3],DEC);
  Serial.println("I would like to connect to Simulink on computer:");

Serial.print(server[0],DEC);Serial.print(".");Serial.print(server[1],DEC);Serial.print(".");Serial.print(server[
```

```
2],DEC);Serial.print(",");Serial.println(server[3],DEC);
  Serial.print("on port number: ");Serial.println(port);
  // start Ethernet
  Ethernet.begin(mac,ip,gateway,subnet);
  //server.begin();
if (client.connect()){
  Serial.println("connected");
}
else{
  Serial.println("connection failed");
}
}

void loop()
{

  //first let's make sure we are connected
  // Connect to server, check for new data
if (!client.connected()){
  client.flush();
  Serial.println("problem!");
  client.stop();
  Serial.println("connecting...");
if (client.connect()){
  Serial.println("back online!");
}
}

for(int i=0;i<200;i++){  //this for loop sets the period of the
  //square wave. each period will be approximately 200 times the
  //10 ms delay, but there will still be a delay
  if(i<100){  //this sets the time that the signal will be held HIGH
    output = 1;  //pulls the signal HIGH
  }
  else{  //outside of the previous time
    output = 0;  //pull the signal LOW
  }
    digitalWrite(ledPin, output);  //write the signal to the LED
    client.println(output);  //write the signal to Simulink
    delay(10);
}

}
```

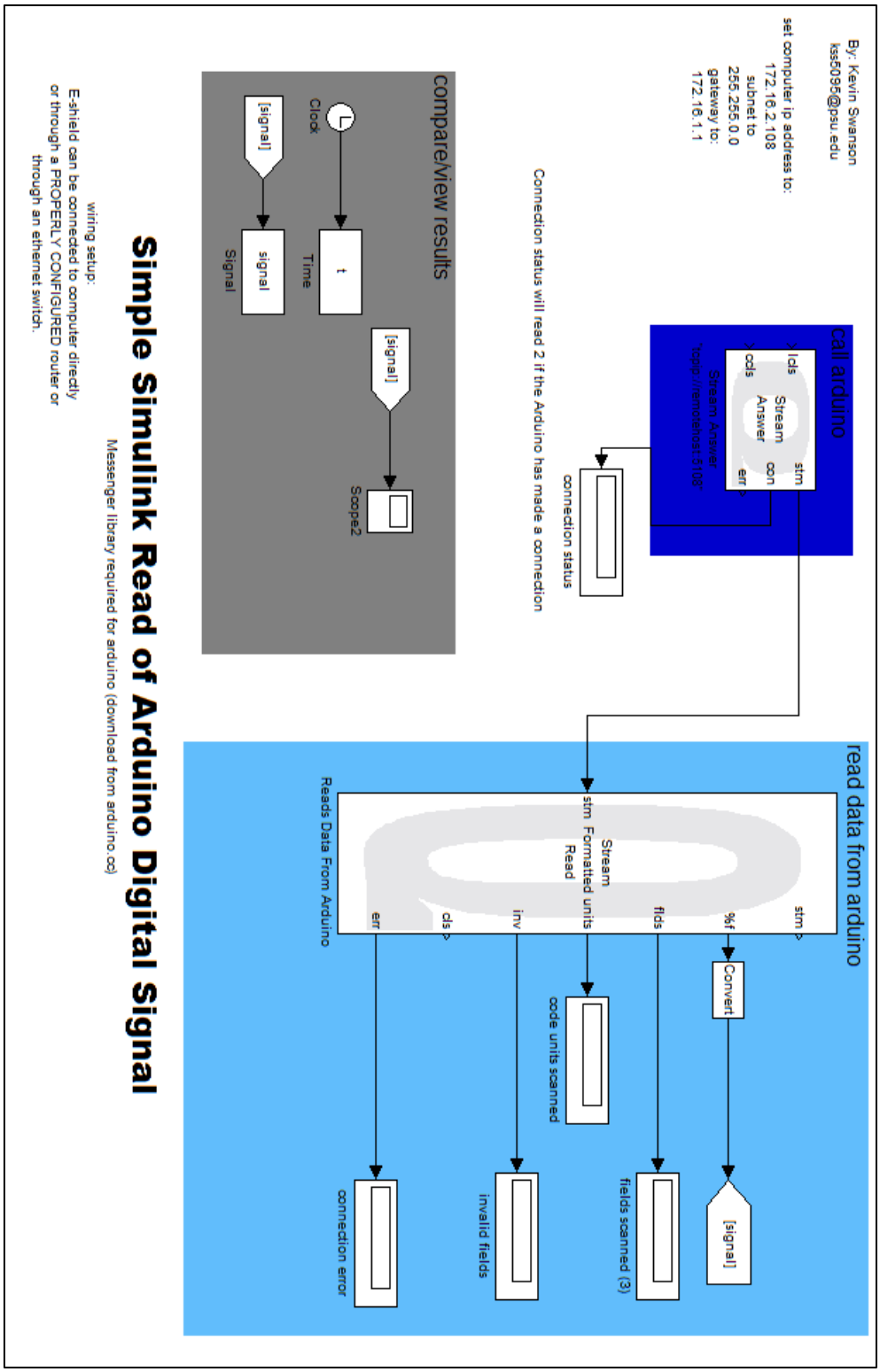

Figure 22: Lab 1 Simulink Diagram

Appendix C – Lab 2: Arduino Digital I/O

The following is the Arduino code for the second lab, not including the hardware setup.

```
/*****************************************************************
 * The majority of this code is taken from Dr. Brennan's ME597D
 * class. It provides a base code for communication with
 * Simulink. Much of the code is simply serial output to confirm
 * the connection with Simulink.
 *
 * All code that involves reading or writing values to the
 * Arduino are written by Kevin Swanson for his undergraduate
 * thesis.
 *
 * Special thanks to Alex Brown for his assistance and Rich
 * Mattes for the hardware code.
 *
 *****************************************************************/

#include <Wire.h>
#include "Messenger.h"
#include <Ethernet.h>
//ETHERNET CONFIG - set these values to correspond to your
//computer and Simulink
byte mac[] = { 0x1C, 0xBD, 0xB9, 0x1C, 0x1C, 0x01 };
byte ip[] = { 172, 16, 2, 108 };  //your Arduino
byte gateway[] = { 172, 16, 2, 1 };
byte subnet[] = { 255, 255, 0, 0 };
byte server[] = {172,16,1,108}; //your computer
//Server dataserver(5100);
int port = 5106; //should match the Simulink port
Client client(server,port);

int blinkPin = 13;  //initialize a pin the led will blink on
int a = 0;  //random value to use flush function later

//now we need to set up a messenger 'object'
//default delimiter is the space character.
Messenger message = Messenger();

int value[1] = {0};  //this holds that values read by the Arduino
//in this case, it will only be the signal sent by the pulse generator
int messageflag = 0;//this tells the program whether we've gotten something

//now we need the callback funtion for messenger
void messageReady(){
  //loop through all the available elements of our message,
  //assigning our read values as we go.
  //in this case, there is only one value, so the loop is removed
  //and only value[0] is left

  if (message.available()){
```

```
    messageflag = 1;
    value[0] = message.readInt();
    }
}



void setup()
{
  pinMode(blinkPin, OUTPUT); //set this pin to be an output
  //the following is just a message to show connectivity
  Serial.begin(115200);
  Serial.println("hello!");
  Serial.println("My IP address is:");
  Serial.print(ip[0],DEC); Serial.print("."); Serial.print(ip[1],DEC); Serial.print(".");
Serial.print(ip[2],DEC); Serial.print("."); Serial.println(ip[3],DEC);
  Serial.println("I would like to connect to Simulink on computer:");

Serial.print(server[0],DEC);Serial.print(".");Serial.print(server[1],DEC);Serial.print(".");Serial.print(server[
2],DEC);Serial.print(".");Serial.println(server[3],DEC);
  Serial.print("on port number: ");Serial.println(port);
  // start Ethernet
  Ethernet.begin(mac,ip,gateway,subnet);
  //server.begin();
  //now we must attach that callback funtion to the messenger object
  message.attach(messageReady);
if (client.connect()){
  Serial.println("connected");
}
else{
  Serial.println("connection failed");
}
}

void loop()
{

  //first let's make sure we are connected
    // Connect to server, check for new data
if (!client.connected()){
  client.flush();
  Serial.println("problem!");
  client.stop();
  Serial.println("connecting...");
if (client.connect()){
  Serial.println("back online!");
}
}
  messageflag = 0;
  //first thing's first: read our message. this fills value into a 1x1 vector
  //the pulse signal is the only element in this vector

  while(client.available()){
    message.process(client.read());
```

```
  }

//this point needs to take the only element of the read data and
//output it to a pin on the Arduino.
  digitalWrite(blinkPin, value[0]);  //this writes the pulse value
  //to the LED pin
  Serial.println(value[0]);  //this writes the pulse value to the
  //serial port

 //now let's send our data back to QuaRC
 if (messageflag ==1){
    //print each value[] element
    client.print(value[0]);
    //delimit with a space
    client.print(" ");

  delayMicroseconds(10);
  //end the message with a carriage return
  client.println();
  delayMicroseconds(10);
  messageflag = 0;
  if(a == 1000){
  client.flush();  //this clears out the values to make room for
  //more. it increases the comm delay drastically, so it is only
  //used every 1000 cycles.
  a = 0;
  }
  a++;
  }

}
```

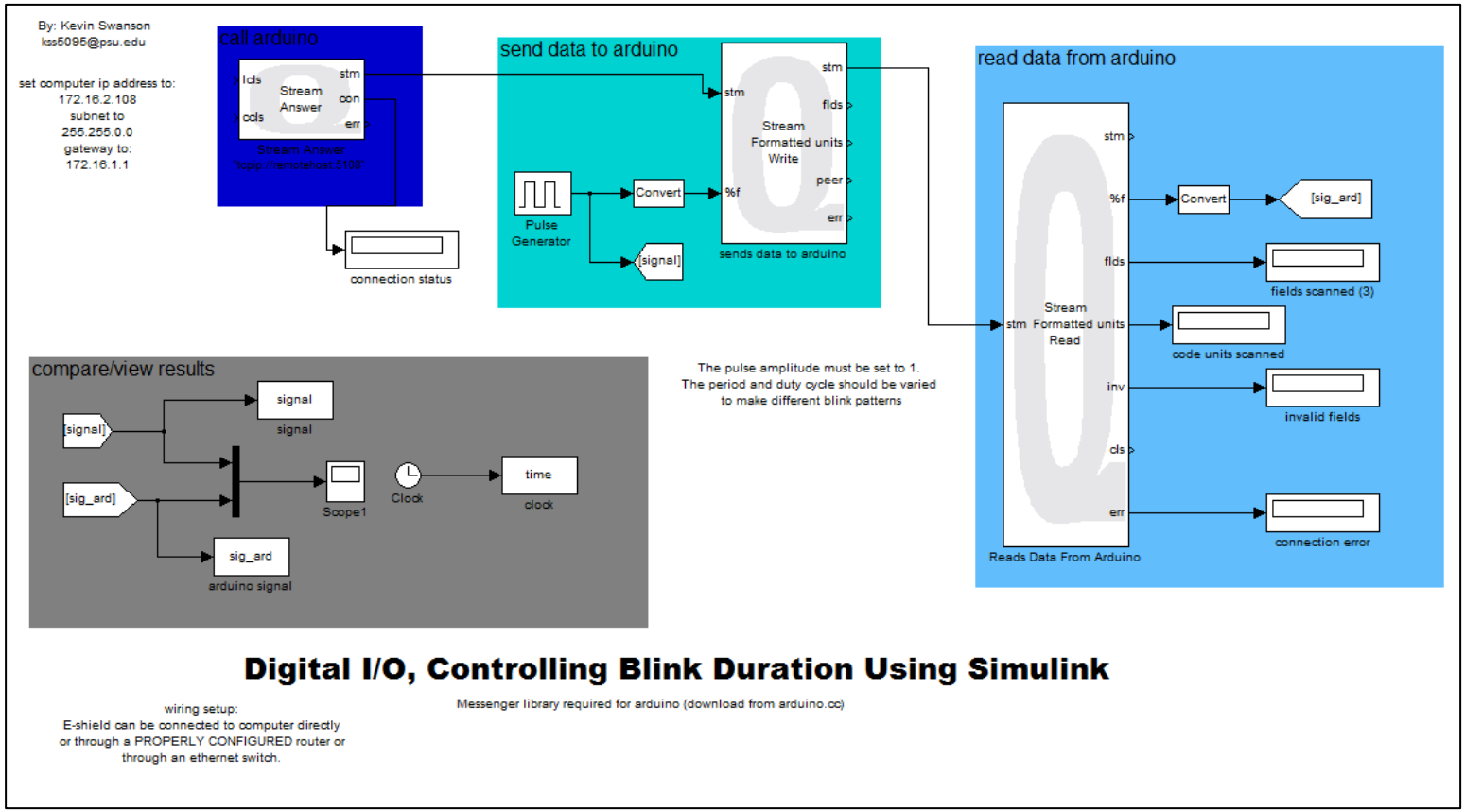

Figure 23: Lab 2 Simulink Diagram

Appendix D – Lab 3: Arduino Analog Output

The following is the Arduino code for the third lab, not including the hardware setup.

```
/*****************************************************************
 * The majority of this code is taken from Dr. Brennan's ME597D
 * class. It provides a base code for communication with
 * Simulink. Much of the code is simply serial output to confirm
 * the connection with Simulink.
 *
 * All code that involves reading or writing values to the
 * Arduino are written by Kevin Swanson for his undergraduate
 * thesis.
 *
 * Special thanks to Alex Brown for his assistance and Rich
 * Mattes for the hardware code.
 *
 *****************************************************************/

#include <Wire.h>
#include "Messenger.h"
#include <Ethernet.h>
//ETHERNET CONFIG
byte mac[] = { 0x1C, 0xBD, 0xB9, 0x1C, 0x1C, 0x01 };
byte ip[] = { 172, 16, 2, 108 };
byte gateway[] = { 172, 16, 2, 1 };
byte subnet[] = { 255, 255, 0, 0 };
byte server[] = {172,16,1,108};
//Server dataserver(5100);
int port = 5108;
Client client(server,port);

int potPin = 8; //designate the pin for your potentiometer (analog)

//now we need to set up a messenger 'object'
//default delimiter is the space character.
Messenger message = Messenger();

double output = 0; //initialize a variable for the output state of the potentiometer

//this loop prints the connection status to the serial port
void setup()
{
  Serial.begin(115200);
  Serial.println("hello!");
  Serial.println("My IP address is:");
  Serial.print(ip[0],DEC); Serial.print("."); Serial.print(ip[1],DEC); Serial.print(".");
Serial.print(ip[2],DEC); Serial.print("."); Serial.println(ip[3],DEC);
  Serial.println("I would like to connect to Simulink on computer:");

Serial.print(server[0],DEC);Serial.print(".");Serial.print(server[1],DEC);Serial.print(".");Serial.print(server[
2],DEC);Serial.print(".");Serial.println(server[3],DEC);
```

```
  Serial.print("on port number: ");Serial.println(port);
  // start Ethernet
  Ethernet.begin(mac,ip,gateway,subnet);
  //server.begin();
if (client.connect()){
  Serial.println("connected");
}
else{
  Serial.println("connection failed");
}
}

void loop()
{

  //first let's make sure we are connected
  // Connect to server, check for new data
if (!client.connected()){
  client.flush();
  Serial.println("problem!");
  client.stop();
  Serial.println("connecting...");
if (client.connect()){
  Serial.println("back online!");
}
}

  output = analogRead(potPin);  //read the analog value from the potentiometer
  client.println(output);  //write the signal to Simulink
  Serial.println(output);  //print the value to the serial monitor
  delay(10);

}
```

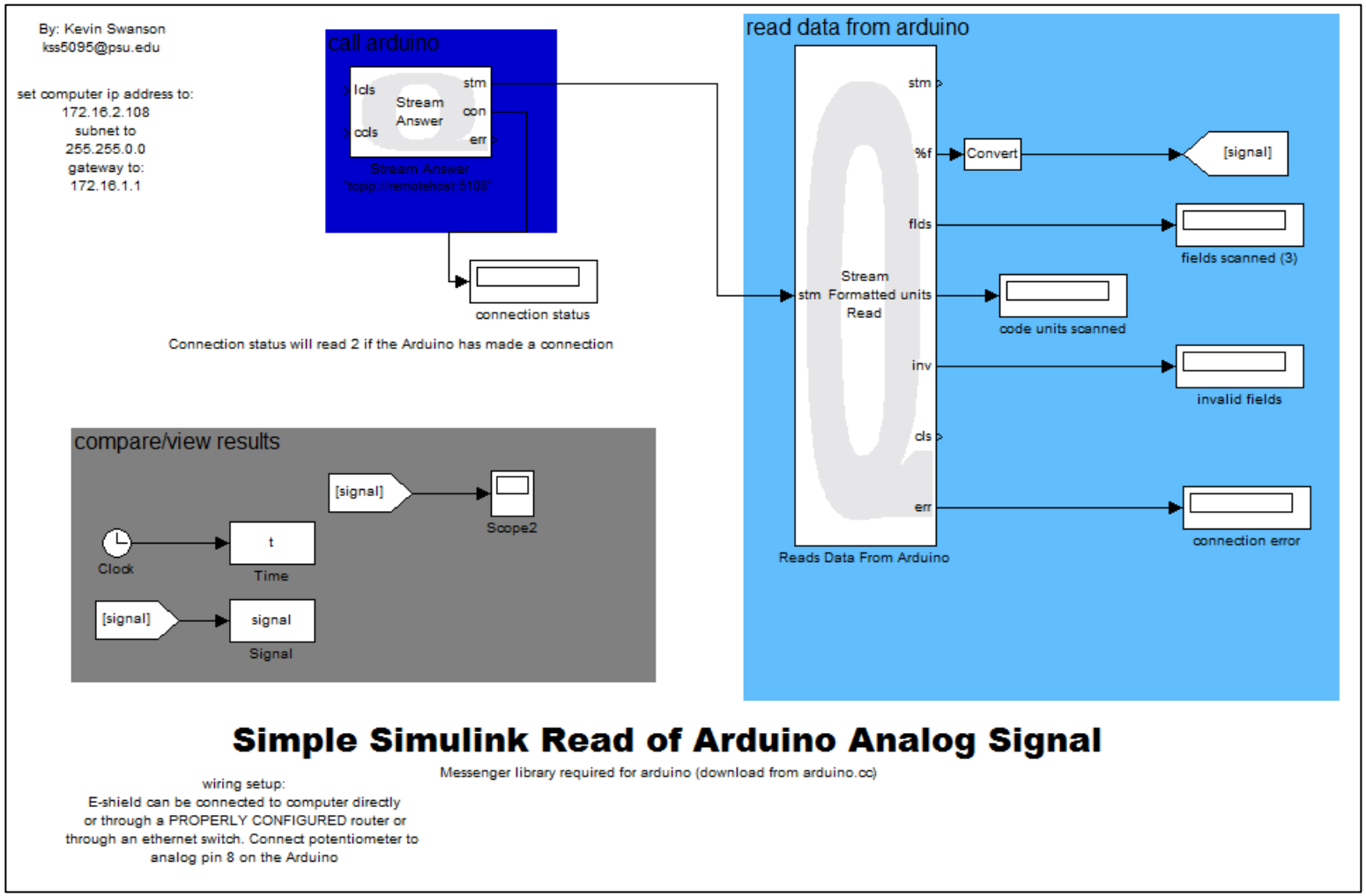

Figure 24: Lab 3 Simulink Diagram

Appendix E – Lab 4: Arduino Analog I/O

The following is the Arduino code for the fourth lab, not including the hardware setup.

```
/*****************************************************************
 * The majority of this code is taken from Dr. Brennan's ME597D
 * class. It provides a base code for communication with
 * Simulink. Much of the code is simply serial output to confirm
 * the connection with Simulink.
 *
 * All code that involves reading or writing values to the
 * Arduino are written by Kevin Swanson for his undergraduate
 * thesis.
 *
 * Special thanks to Alex Brown for his assistance and Rich
 * Mattes for the hardware code.
 *
 *****************************************************************/

#include <Wire.h>
#include "Messenger.h"
#include <Ethernet.h>
#include <Servo.h>  //include the servo library
//ETHERNET CONFIG - set these values to correspond to your
//computer and Simulink
byte mac[] = { 0x1C, 0xBD, 0xB9, 0x1C, 0x1C, 0x01 };
byte ip[] = { 172, 16, 2, 108 };  //your Arduino
byte gateway[] = { 172, 16, 2, 1 };
byte subnet[] = { 255, 255, 0, 0 };
byte server[] = {172,16,1,108}; //your computer
//Server dataserver(5100);
int port = 5108; //should match the Simulink port
Client client(server,port);

Servo myservo;  //initialize the servo
int output = 0;  //initalize a value to hold the servo angle
int a = 0;  //random value to use flush function later

//now we need to set up a messenger 'object'
//default delimiter is the space character.
Messenger message = Messenger();

double value[2] = {0,0};  //this holds that values read by the Arduino
//in this case, it will only be the clock and sine signal
int messageflag = 0;//this tells the program whether we've gotten something

//now we need the callback funtion for messenger
void messageReady(){
  //loop through all the available elements of our message,
  //assigning our read values as we go.
  //in this case, there is only one value, so the loop is removed
  //and only value[0] is left
```

```
  for(int i=0;i<2;i++){
  if (message.available()){
    messageflag = 1;
    value[i] = message.readDouble();
    }
  }
}



void setup()
{
  myservo.attach(22);  //attach the servo to pin 22 (digital)
  //the following is just a message to show connectivity
  Serial.begin(115200);
  Serial.println("hello!");
  Serial.println("My IP address is:");
  Serial.print(ip[0],DEC); Serial.print("."); Serial.print(ip[1],DEC); Serial.print(".");
Serial.print(ip[2],DEC); Serial.print("."); Serial.println(ip[3],DEC);
  Serial.println("I would like to connect to Simulink on computer:");

Serial.print(server[0],DEC);Serial.print(".");Serial.print(server[1],DEC);Serial.print(".");Serial.print(server[
2],DEC);Serial.print(".");Serial.println(server[3],DEC);
  Serial.print("on port number: ");Serial.println(port);
  // start Ethernet
  Ethernet.begin(mac,ip,gateway,subnet);
  //server.begin();
  //now we must attach that callback funtion to the messenger object
  message.attach(messageReady);
if (client.connect()){
  Serial.println("connected");
}
else{
  Serial.println("connection failed");
}
}

void loop()
{

  //first let's make sure we are connected
    // Connect to server, check for new data
if (!client.connected()){
  client.flush();
  Serial.println("problem!");
  client.stop();
  Serial.println("connecting...");
if (client.connect()){
  Serial.println("back online!");
}
}
  messageflag = 0;
  //first thing's first: read our message. this fills value into a 1x1 vector
```

```
 //the pulse signal is the only element in this vector

 while(client.available()){
   message.process(client.read());
 }

 //we need to write the signal from Simulink to the servo
 output = value[1];  //this sets the second element (sine signal) to
 //be the output
 myservo.write(output);  //this writes that value to the servo
 Serial.println("The servo value is:");
 Serial.print(output);  //this writes the servo value to the
 //serial port

 //now let's send our data back to QuaRC
if (messageflag ==1){
   for(int i=0;i<2;i++){
   //print each value[] element
   client.print(value[i]);
   //delimit with a space
   client.print(" ");
 }
 delayMicroseconds(10);  //these delays are also necessary for letting
 //the servo move to its destination
 //end the message with a carriage return
 client.println();
 delayMicroseconds(10);
 messageflag = 0;
 if(a == 1000){
 client.flush();  //this clears out the values to make room for
 //more. it increases the comm delay drastically, so it is only
 //used every 1000 cycles.
 a = 0;
 }
 a++;
 }

}
```

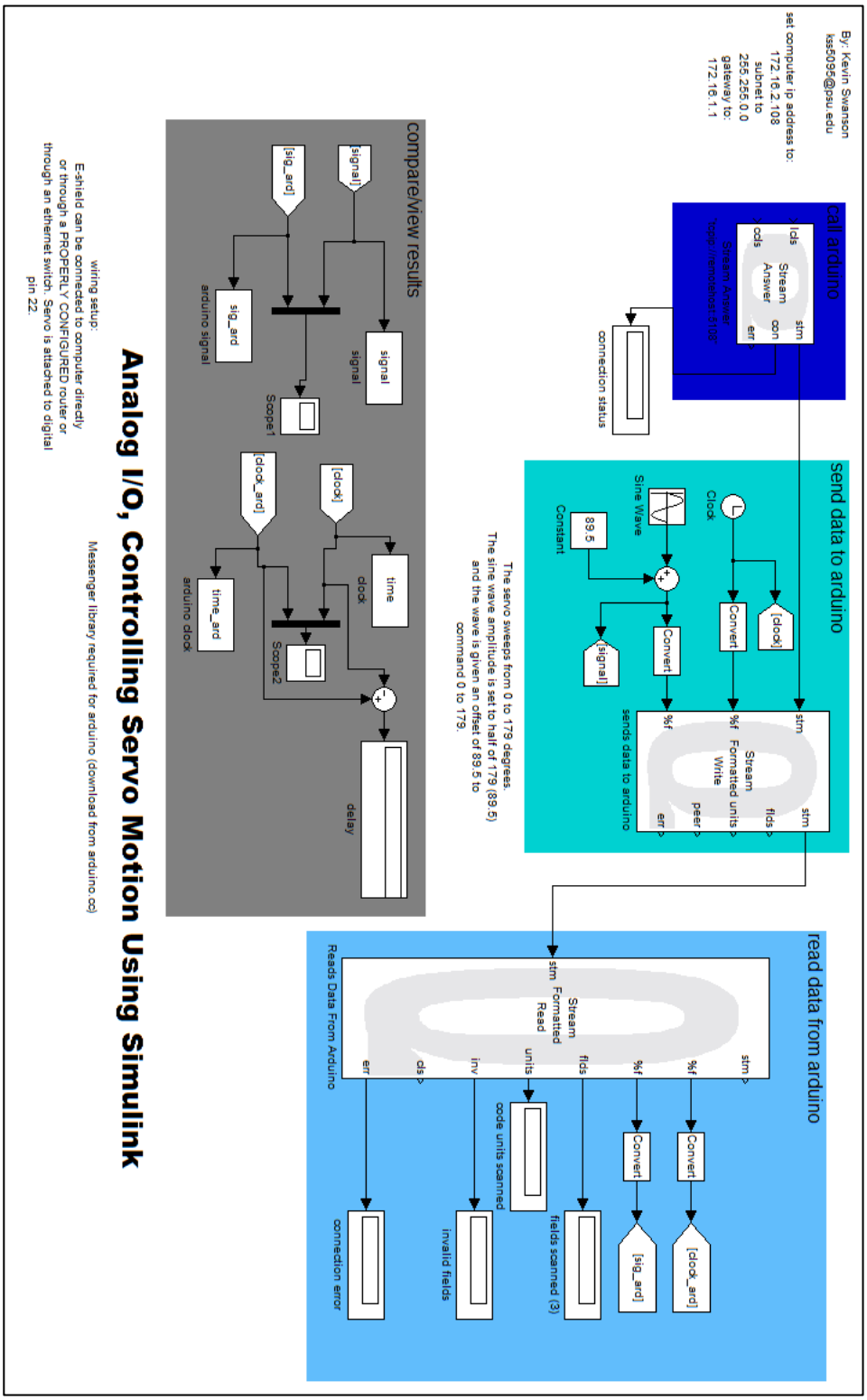

Figure 25: Lab 4 Simulink Diagram

Appendix F – Lab 5: Arduino Analog Control

The following is the Arduino code for the fifth lab, not including the hardware setup.

```
/*****************************************************************
 * The majority of this code is taken from Dr. Brennan's ME597D
 * class. It provides a base code for communication with
 * Simulink. Much of the code is simply serial output to confirm
 * the connection with Simulink.
 *
 * All code that involves reading or writing values to the
 * Arduino are written by Kevin Swanson for his undergraduate
 * thesis.
 *
 * Special thanks to Alex Brown for his assistance and Rich
 * Mattes for the hardware code.
 *
 *****************************************************************/

#include <Wire.h>
#include "Messenger.h"
#include <Ethernet.h>
#include <Servo.h>  //include the servo library
//ETHERNET CONFIG - set these values to correspond to your
//computer and Simulink
byte mac[] = { 0x1C, 0xBD, 0xB9, 0x1C, 0x1C, 0x01 };
byte ip[] = { 172, 16, 2, 108 };  //your Arduino
byte gateway[] = { 172, 16, 2, 1 };
byte subnet[] = { 255, 255, 0, 0 };
byte server[] = {172,16,1,108}; //your computer
//Server dataserver(5100);
int port = 5108; //should match the Simulink port
Client client(server,port);

int potPin = 8;  //pin 8 for the potentiometer
Servo myservo;  //initialize the servo
int output = 0;  //initalize a value to hold the servo angle
int a = 0;  //random value to use flush function later

//now we need to set up a messenger 'object'
//default delimiter is the space character.
Messenger message = Messenger();

double value[2] = {0,0};  //this holds that values read by the Arduino
//in this case, it will only be the clock and sine signal
int messageflag = 0;//this tells the program whether we've gotten something

//now we need the callback funtion for messenger
void messageReady(){
  //loop through all the available elements of our message,
  //assigning our read values as we go.
  //in this case, there is only one value, so the loop is removed
```

```
  //and only value[0] is left

  for(int i=0;i<2;i++){
  if (message.available()){
    messageflag = 1;
    value[i] = message.readDouble();
    }
  }
}



void setup()
{
  myservo.attach(22);  //attach the servo to pin 22 (digital)
  //the following is just a message to show connectivity
  Serial.begin(115200);
  Serial.println("hello!");
  Serial.println("My IP address is:");
  Serial.print(ip[0],DEC); Serial.print("."); Serial.print(ip[1],DEC); Serial.print(".");
Serial.print(ip[2],DEC); Serial.print("."); Serial.println(ip[3],DEC);
  Serial.println("I would like to connect to Simulink on computer:");

Serial.print(server[0],DEC);Serial.print(".");Serial.print(server[1],DEC);Serial.print(".");Serial.print(server[
2],DEC);Serial.print(".");Serial.println(server[3],DEC);
  Serial.print("on port number: ");Serial.println(port);
  // start Ethernet
  Ethernet.begin(mac,ip,gateway,subnet);
  //server.begin();
  //now we must attach that callback funtion to the messenger object
  message.attach(messageReady);
if (client.connect()){
  Serial.println("connected");
}
else{
  Serial.println("connection failed");
}
}

void loop()
{

  //first let's make sure we are connected
    // Connect to server, check for new data
if (!client.connected()){
  client.flush();
  Serial.println("problem!");
  client.stop();
  Serial.println("connecting...");
if (client.connect()){
  Serial.println("back online!");
}
}
  messageflag = 0;
```

```
 //first thing's first: read our message. this fills value into a 1x1 vector
 //the pulse signal is the only element in this vector

 while(client.available()){
   message.process(client.read());
 }

 //we need to write the signal from Simulink to the servo
 output = value[1];  //this sets the second element (sine signal) to
 //be the output
 myservo.write(output);  //this writes that value to the servo
 Serial.print("The servo value is:");
 Serial.println(output);  //this writes the servo value to the
 //serial port
 value[1] = analogRead(potPin);  //read the value of the potentiometer
 //and set it to the second element
 Serial.print("The potentiometer reading is:");
 Serial.println(value[1]);

 //now let's send our data back to QuaRC
if (messageflag ==1){
   for(int i=0;i<2;i++){
   //print each value[] element
   client.print(value[i]);
   //delimit with a space
   client.print(" ");
 }
 delayMicroseconds(10);  //these delays are also necessary for letting
 //the servo move to its destination
 //end the message with a carriage return
 client.println();
 delayMicroseconds(10);
 messageflag = 0;
 if(a == 1000){
 client.flush();  //this clears out the values to make room for
 //more. it increases the comm delay drastically, so it is only
 //used every 1000 cycles.
 a = 0;
 }
 a++;
 }

}
```

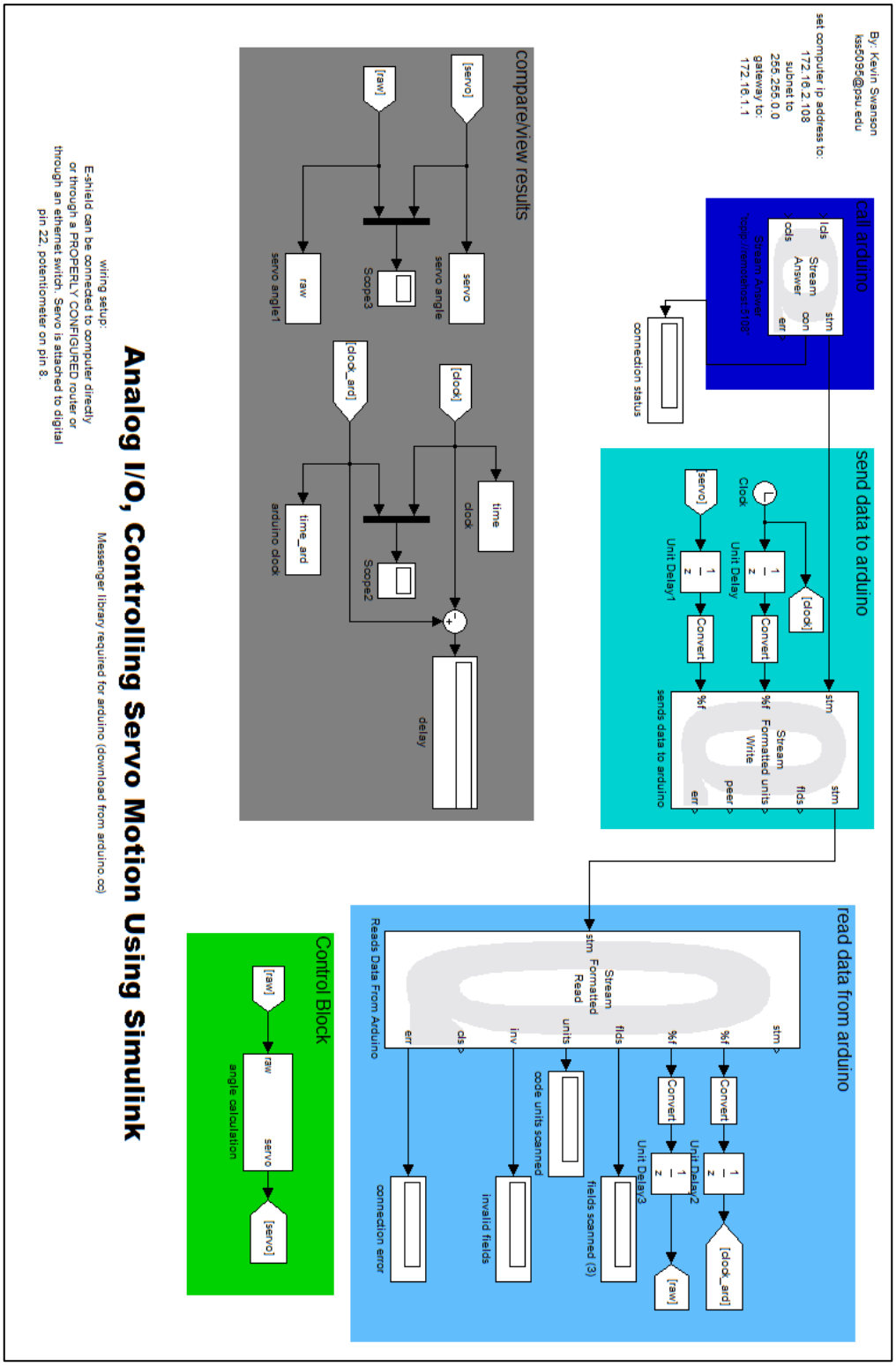

Figure 26: Lab 5 Simulink Diagram

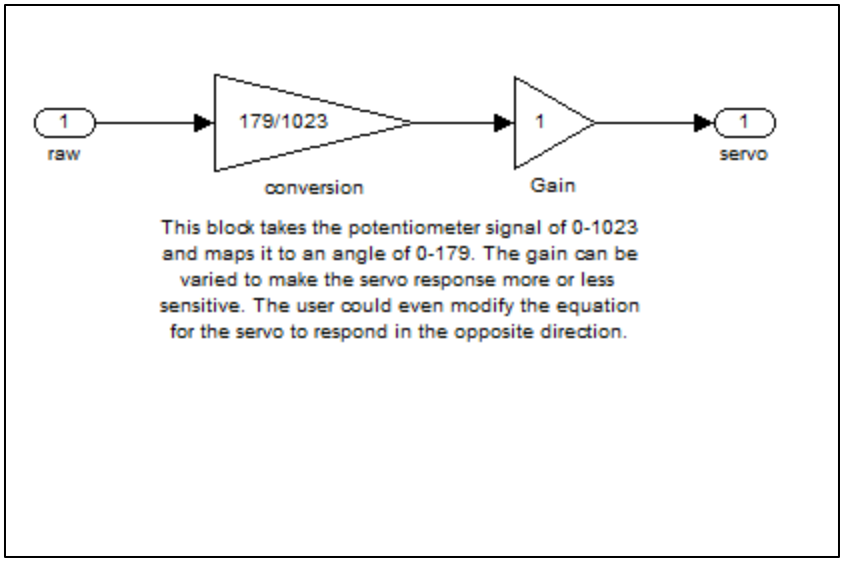

Figure 27: Lab 5 Control Block

Appendix G – Lab 6: MATLAB Serial Digital Read

The following is the Arduino code for the sixth lab.

```
////////////////////////////////////////////////////////////////
//
//  This program written by Kevin Swanson to
//  demonstrate the ability to communicate between
//  an Arduino and MATLAB with a serial connection
//
//  No wiring is required for this program, it imitates
//  the Arduino MEGA blink program
//
//
//
//
////////////////////////////////////////////////////////////////
int ledPin =  13;    // LED connected to digital pin 13
int output;    //initialize a value for the signal

void setup()   {
  Serial.begin(9600);    //open the serial port
  pinMode(ledPin, OUTPUT);    //set the pin as an output
}

void loop()
{
  for(int i=0;i<200;i++){  //this for loop sets the period of the
  //square wave. each period will be approximately 200 times the
  //10 ms delay, but there will still be a delay
  if(i<100){  //this sets the time that the signal will be held HIGH
    output = 1;  //pulls the signal HIGH
  }
  else{  //outside of the previous time
    output = 0;  //pull the signal LOW
  }
    digitalWrite(ledPin, output);  //write the signal to the LED
    Serial.println(output);  //write the signal to the serial port
    delay(10);
}
}
```

The following is the MATLAB code for the sixth lab.

```
% Author: Kevin Swanson
% Program for Arduino MEGA
% Reads digital output of Arduino and plots this output
clear all;
s1 = serial('COM12');                              %define serial port
s1.BaudRate=9600;                                  %define baud rate

%open serial port
fopen(s1);
clear data;

%create a delay while the port opens
val=0;
while(val<10000000)
    val=val+.03;
end

%this portion reads the arduino data - acquisition of 1000 points
for i= 1:1000;                  %set number of times to run through
code
    data=fscanf(s1);            %read sensor
    angle_data=str2double(data);        %convert the string to a double
    %plot the data
    plot(i,angle_data);
    hold all;
    title('Digital Output vs. Data Point');
    xlabel('Data Point');
    ylabel('Digital Output');
    axis([0 1000 -0.5 1.5]);
end
fclose(s1);                 %CLOSE THE SERIAL PORT!
```

Appendix H – Lab 7: MATLAB Serial Digital I/O

The following is the Arduino code for the seventh lab.

```
////////////////////////////////////////////////////////////////
//
//  This program written by Kevin Swanson to
//  demonstrate the ability to communicate between
//  an Arduino and MATLAB with a serial connection
//
//  No wiring is required for this program, it imitates
//  the Arduino MEGA blink program
//
//
//
//
////////////////////////////////////////////////////////////////
int ledPin = 13;  // digital pin used to connect the led

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

void loop()
{
  int output = 0;
  if(Serial.available())
  {
    output = Serial.read();
    digitalWrite(ledPin, output);
  }
}
```

The following is the MATLAB code for the seventh lab.

```
% Author: Kevin Swanson
% Program for Arduino MEGA
% Sends a high/low signal to the Arduino as a blink program
clear all;
s1 = serial('COM12');                          %define serial port
s1.BaudRate=9600;                              %define baud rate

%open serial port
fopen(s1);
clear data;

%create a delay while the port opens
val=0;
while(val<10000000)
    val=val+.03;
end

%this loop sends an on/off blink patter to the Arduino
for i= 1:5;                    %set number of blinks
    output = 1;                %set output high
    fwrite(s1,output);         %print the output to the serial port
    pause(1);                  %wait a second
    output = 0;                %repeat for low
    fwrite(s1,output);
    pause(1);
end
fclose(s1);                   %CLOSE THE SERIAL PORT!
```

Appendix I – Lab 8: MATLAB Serial Analog I/O

The following is the Arduino code for the eighth lab.

```
////////////////////////////////////////////////////////////////
//
//  This program written by Kevin Swanson to
//  demonstrate the ability to communicate between
//  an Arduino and MATLAB with a serial connection
//
//  A servo and potentiometer are needed for this program
//  Pins are designated in code comments
//
//
//
//
////////////////////////////////////////////////////////////////
#include <Servo.h>

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

int potpin = 0;  // analog pin used to connect the potentiometer
int val = 0;    // variable to read the value from the analog pin
int angle = 0;  //variable to hold the calculated angle

void setup()
{
  Serial.begin(9600);  //open the serial port
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop()
{
  val = analogRead(potpin);            // reads the value of the potentiometer (value between 0 and 1023)
  angle = map(val, 0, 1023, 0, 179);     //convert to angle
  Serial.println(angle);                // send data to and from MATLAB
  myservo.write(angle);                 // sets the servo position according to the scaled value
}
```

The following is the MATLAB code for the eighth lab.

```
% Author: Kevin Swanson
% Program for Arduino MEGA
% Reads angle from example servo program and plots commanded angle
close all;
clear all;
clc;
s1 = serial('COM18');                              %define serial port
s1.BaudRate=9600;                                  %define baud rate
counts = 2500;                                     %define number of
points
angle = 0;                                         %define angle
angle_data = 0;                                    %define angle_data
final_angle = zeros(counts,1);                     %make a matrix for
angle
seconds = zeros(counts,1);                         %make a matrix for
seconds

%open serial port
fopen(s1);
clear data;

%create a delay while the port opens
val=0;
while(val<10000000)
    val=val+.03;
end

HandleAngle = plot(seconds,final_angle);  % Keep the handle for this
plot
hold on
title('Commanded Angle vs Time')
xlabel('MATLAB Time (s)')
ylabel('Commanded Angle (deg)')
axis([0 15 0 180])
set (HandleAngle,'Erase', 'xor')  % Set these so it updates

tic         %start timer
%this portion reads the arduino data - acquisition of "counts" points
for i= 1:counts
    seconds(i)= toc;                %set seconds matrix
    angle = fscanf(s1);             %read serial port
    angle_data = str2double(angle); %convert the string to a double
    final_angle(i) = angle_data;    %set current angle value
    %plot the data
    set(HandleAngle,'YData',final_angle)  %change just the y data for
plot
    set(HandleAngle,'XData',seconds)      %change just the x data for
plot
    drawnow
end
fclose(s1);                 %CLOSE THE SERIAL PORT!
```

ACADEMIC VITA of Kevin Swanson

Local: 212 W. Fairmount Avenue, State College, PA 16801 (571)-262-1217
Permanent: 12483 Marstan Moor Lane, Herndon, VA 20171 kss5095@psu.edu

Education

The Pennsylvania State University, Schreyer Honors College – *University Park, PA*
(2007-Present) B.S. expected in May 2011 in Mechanical Engineering

Coursework: modeling dynamic systems, vehicle dynamics, engineering design, microcomputer interfacing, fluid flow dynamics, heat transfer, engineering mechanics: statics, dynamics, and strength of materials

Research

Intelligent Vehicles and Systems Group, Penn State University – *University Park, PA*
(May 2010-Present)

Worked on multiple projects including the design and construction of an inductive guidance system along with testing of tire deflation devices and subsequent reporting. Current work is the improvement of the mechatronics classes at Penn State by using existing software.

Activities/Awards

Penn State Dance Marathon (THON) Entertainment Captain (September 2010-February 2011)
Atlas THON – Administrative Assistant (June 2009-April 2010)
THON Morale Committee (October 2008-February 2009, October 2009-February 2010)
The Chi Phi Fraternity – Secretary (December 2008-May 2009)
Eagle Scout, Boy Scouts of America
Schreyer Honors College Academic Scholarship

Skills

Microsoft Office, SolidWorks, AutoCAD, MATLAB, Simulink, Japanese language