

# **Light Display Simulation**

**by**

**Hunter Holmes hholmes1**

**Report submitted on 28 November 2018**

**EE 333 – Engineering Programming Using Objects**

**Department of Electrical and Computer Engineering  
The University of Alabama at Birmingham**

## **ABSTRACT**

TBD

## Table of Contents

<b>ABSTRACT.....</b>	<b>I</b>
<b>INTRODUCTION.....</b>	<b>1</b>
<b>PROJECT DEFINITION .....</b>	<b>1</b>
CONSTRAINTS .....	1
GOALS .....	1
FEATURES (IN MoSCoW LIST FORMAT).....	2
APPLICABLE STANDARDS .....	2
<b>DESIGN.....</b>	<b>2</b>
INITIAL DESIGN PROCESS .....	2
APPROPRIATE FOR OBJECT ORIENTED APPROACH .....	3
DESIGN DECISIONS.....	3
OBJECT ORIENTED DESIGN .....	5
<b>DISCOVERY AND USE OF ONLINE INFORMATION.....</b>	<b>8</b>
<b>DEBUG.....</b>	<b>8</b>
<b>RESULTS .....</b>	<b>9</b>
<b>DISCUSSION.....</b>	<b>10</b>
<b>CONCLUSIONS .....</b>	<b>10</b>
<b>REFERENCES.....</b>	<b>10</b>
<b>APPENDIX.....</b>	<b>11</b>

## **INTRODUCTION**

The overall goal of this project is to improve the designer's skills and understanding of object-oriented design and implementation. Some programming languages allow the designer to create objects in code that have definitions based on real-world objects. This type of programming is called Object-Oriented. An example of an object-oriented programming language is Java which is used in this project. Although objects can be used to model a physical part of a problem they are also used for internal parts of the system, for example clocks which maintain synchronization or loggers which keep track of system activity.

## **PROJECT DEFINITION**

The program chosen for this project is a light control system. Hardware implementation is not included in this project, but lights which are a part of the module will be connected to a controller which makes decisions regarding light behavior based on user input. Components of this system are to be designed and modeled using objects. A controller will send commands to an array of modules which will send commands to the individual lights that are a part of that module. Each of these components shall be modeled as objects. The lists that follow, detail the constraints, goals, and specific objectives of this project.

### ***Constraints***

- CO-01: The developer shall program the solution in Java
- CO-02: The program shall be written using NetBeans
- CO-03: The modeling classes shall not contain I/O
- CO-04: The solution shall be built using Object-Oriented Design
- CO-05: Modules shall be connected in series

### ***Goals***

- GO-01: The system should be scalable regarding number of lights connected
- GO-02: The system should work with light modules possessing a variety of features and architectures
- GO-03: The project should possess test code that demonstrates its capabilities

### ***Features (in MoSCoW list format)***

The desired features include:

- Must-01: Have light modules synchronized with one another
- Must-02: Allow user to have some control of behavior
- Should-01: Support at least 10,000 lights total
- Should-02: Support 4 light modules per channel
- Should-03: Contain a variety of preset modes to send to light modules
- Should-05: Detect type and relative position of newly attached modules
- Could-01: Be implemented with hardware
- Could-02: Control lights' brightness and color
- Could-06: Be user friendly
- Won't-01: Include a GUI

### ***Applicable Standards***

- STD-01: Documentation will follow Javadoc standards
- STD-02: Source code will follow standards set by the UAB Department of Electrical and Computer Engineering

## **DESIGN**

### ***Initial Design Process***

Upon receiving project, the assignment document was reviewed carefully with notes being taken regarding which physical objects were being modeled, constraints, and clear goals. The system was considered as it would be expected to behave. A user may input a command to an individual light module or to the array of modules in its entirety. The system would then take this input and distribute commands to the proper modules which would distribute commands to its respective lights. This scenario sets up the basic structure for the overall operation of the system.

Observing that this system relies on commands being passed down from systems to subsystems, a few objects like controller, light module, and lights were readily apparent. Reviewing the document once more, all physical items which were being modeled were written down as potential objects to be designed in the program. The properties of each of these objects were studied and then written down as descriptions for each object. For

example, a light object would have properties such as an ON/OFF setting, brightness value, and color value. These properties are important to the object-oriented design process as they allow for the creation of classes with the same properties that can be changed and observed.

With some potential objects defined, the classes necessary for writing the program began to take shape. Many classes were defined to be the same as the objects they modeled. For example, the class used for a module object would be a Module.java. Some other classes would also be needed to model some internal/non-physical components, such as a clock.

After defining the basic operation scenario as well as the prospective classes, interaction diagrams were created. These diagrams create visualization for queries and commands that are sent from one class to another in a system cycle for a particular scenario. There are many possible scenarios that could be used for interaction diagrams but only 2 diagrams that were most enlightening were used. A class diagram for the system was also created. This diagram demonstrates how classes are dependent on one another. This type of diagram is not dependent on a particular scenario so only one is needed.

### ***Appropriate for Object Oriented Approach***

The light module controller project lends itself to an object-oriented approach. The system is comprised of interconnected components which relay information to one another. This fits very well with the concepts of commands and queries which are found in an object-oriented programming design. The problem also insists that some components should have standardized properties but should also be duplicated over and over, most notably the light modules and the lights themselves. A class called “module” could be created and then as many module objects as are necessary could be created out of the patterning class. This is just one example, but this technique will be used heavily in this project

### ***Design Decisions***

#### AL-01 How should a message for a particular module identify itself?

AL-01A Each module asks the controller if the message belongs to it

AL-01B Sending the module’s ID along with the message

ALD-01 Adding information about the intended recipient would save a great deal of time if a message was for example for the last module in a series. The modules in line before it would simply ignore the message if it did not include their unique ID within the system. If a message is intended for all modules in a series, then the ID portion could be left out as all modules need to handle the message

AL-02 How should properties of newly connected module be established?

AL-02A User inputs module information

AL-02B Module identifies itself by its unique ID

ALD-02 For this problem option A is the most appropriate solution. A user can create a new module from a list of predetermined types. The user would then put this unique ID in the add method for the controller which would determine the properties of the module based on portions of the ID. If hardware implementation were used this feature may be withdrawn as the controller would simply poll the module to determine its ID. Since this implementation is completely virtual the user will have to input some identifying information about the module.

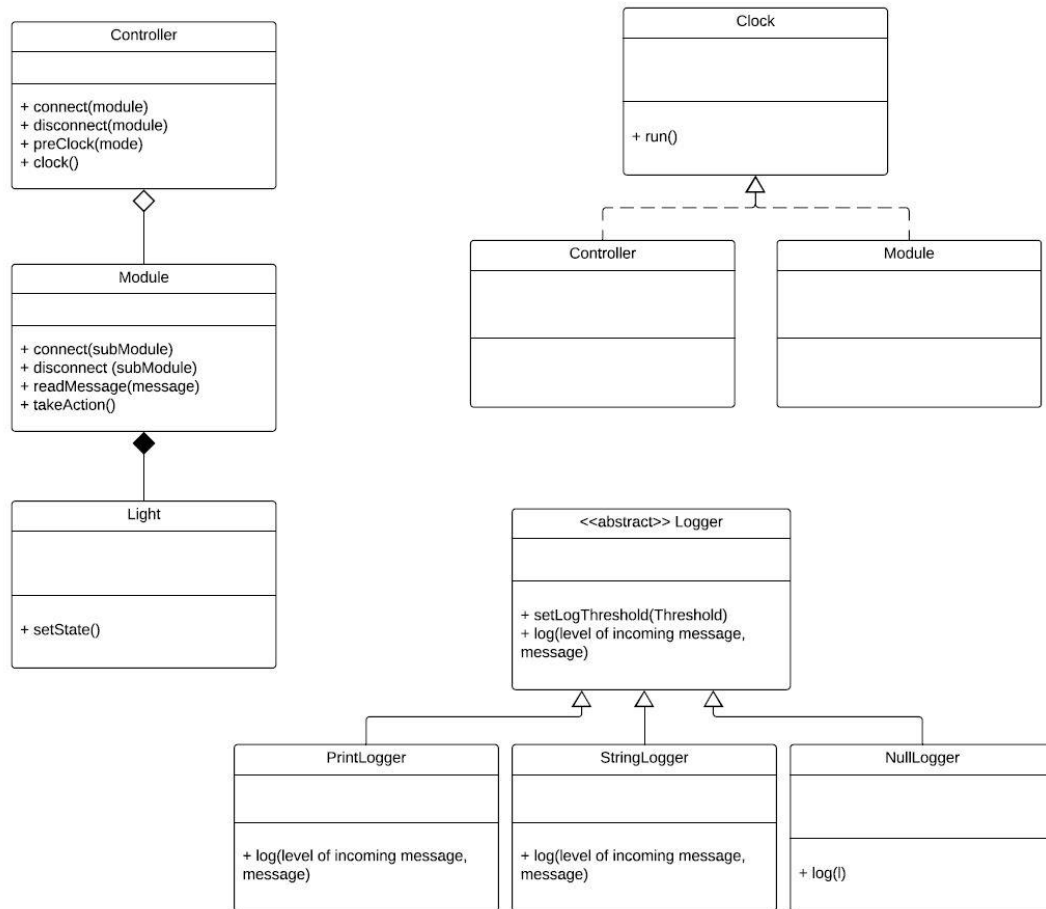
AL-03 How should timing synchronization be dealt with between modules?

AL-03A Use a clocking system

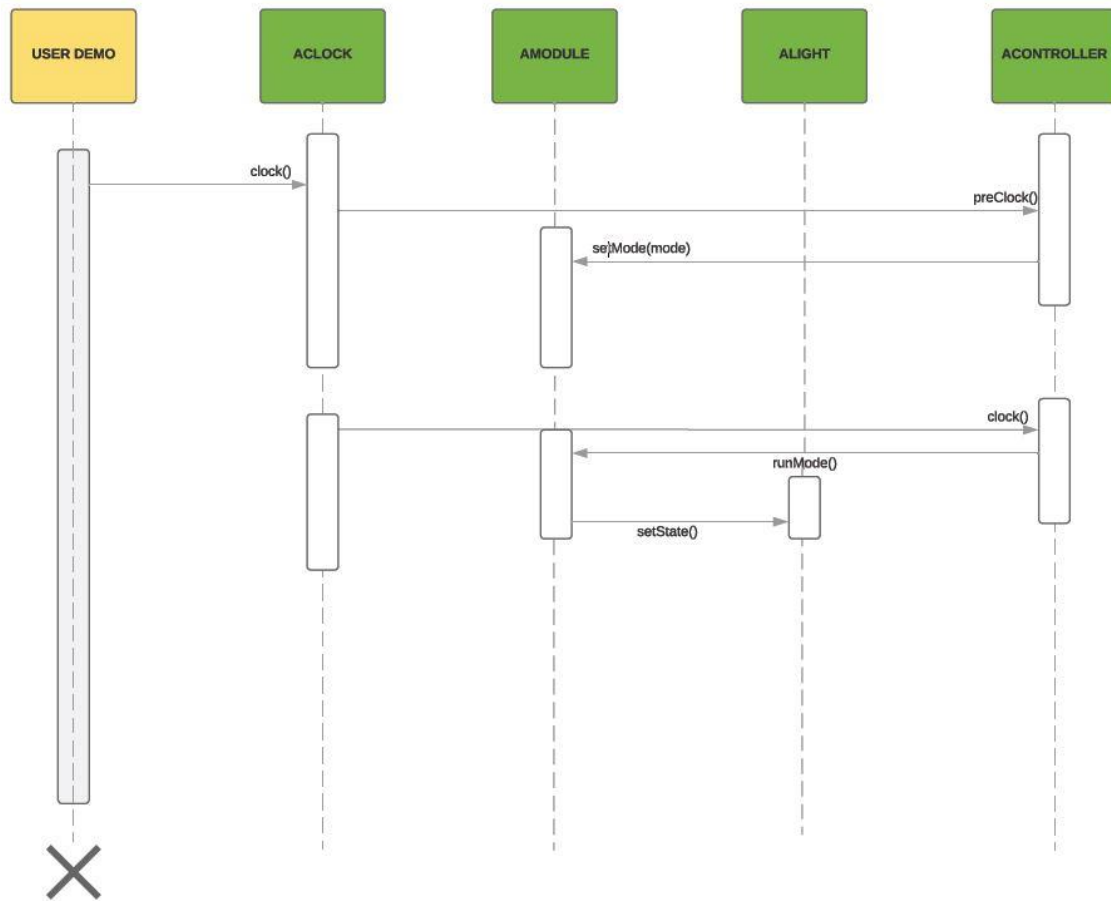
AL-03B Timing is ignored, simply send commands to modules when input is changed

ALD-03 Using a clocking system, option A, is the cleanest solution to this problem. It adds extra work to the build and design of the system, but it deals with the issue of timing. Simply due to processing time some portions of the system may become out of time with the rest of the system. Having a shared clock between all modules would allow them all the chance to receive their next mode command in one cycle and then act on the next cycle. Since the clock command would be miniscule in size and processing time, the delay would be minimized.

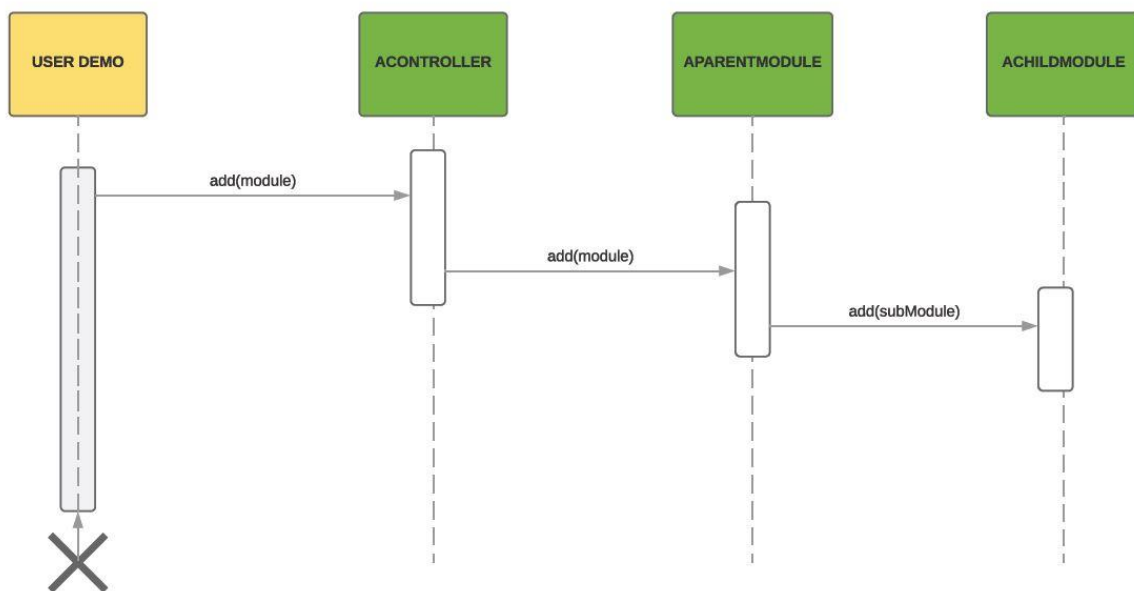
## Object Oriented Design



**Fig. 01 Class Diagram**

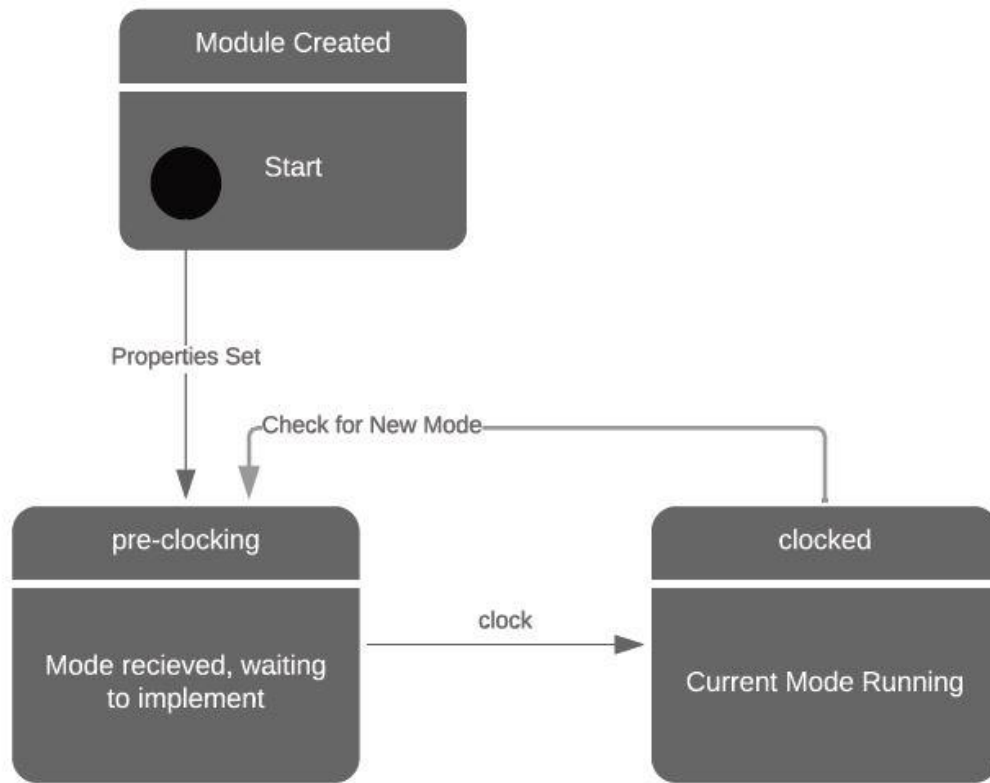


**Fig. 2 Mode Change Interaction Diagram**



**Fig. 3 Add Module Interaction Diagram**





**Fig. 4 Module State Diagram**

## **DISCOVERY AND USE OF ONLINE INFORMATION**

To maintain the potential for hardware implementation in this project it was necessary to consider what sort of information could be transmitted along a wire and passed in series. Continuing with the design decision to pass a message along it was necessary to understand how a binary message could be created and passed around. Numerical data types in java are base 10 so it is not as simple as setting an int equal to the binary value desired. After reseaching online, it became clear that if a binary message was to be used it would have to be stored as a string and converted back and forth between this string binary value and an integer decimal value. The Object versions of primitive data types, in this case Integer for int, proved to be very useful. Integer as an object has a great deal of built in functions designed for parsing between data types. It was also discovered that Strings have many useful functions for separating a string into segments so that it may be compared to other characters.

## **DEBUG**

After a framework of essential functions was established it was time to begin the process of running sample data and debugging. Fortunately, NetBeans contains a powerful tool, the Debugger, this built in tools allows for the developer to step through each step of a scenario for a program. The user is allowed to make selected break-points so data can be observed just before it is modified by a certain line.

Once the first run was attempted it was clear the debugger would prove useful. An `ArrayIndexOutOfBoundsException` Exception was encountered, but the cause was not readily apparent. First, setting a break point on the line of failure did not seem to reveal the issue, so the Step Into tool was used. With Step Into, the developer is allowed to move through each line of executable code and observe changes that are made in slow and controlled way. Digging deeper it was clear that no error checking had been included in a for loop that was looking for light modules that did not yet exist in the array. A simple if statement to preface the loop corrected the issue quickly. This type of issue was the first of many that were encountered as the program was tested further. The debugger tool helped discover several more cases involving loops that left the boundaries of arrays. Since the breakpoints stop the execution just before the line they are placed on, the user can place the cursor directly over a variable of interest to see its current value. This is particularly useful in `NullPointerException` and `OutOfBoundsException` exceptions as the developer can see which values are going to cause the crash before it actually happens.

## RESULTS

After working through errors like the ones discussed in the debugging section, output was returned. Initially simple cases were tested, for example only one small light moduled connected running the simplest mode. Slowly more modules were added, and more challenging cases were tested, to verify the functionality of certain components. Currently only two modes are useable in the application. The framework exists for more, but the “ALL ON” and “ALL OFF” seem to demonstrate the functionality of the system. A TestP5 class is used to demo the capabilities of the program and a sample run with logged output is shown below in Fig. 5 and Fig. 6.

```
* Assignment: P5
* Vers: 1.0.0 11/07/2018 hah - initial coding
*/

/**
 *
 * @author Hunter Holmes hholmes1@uab.edu
 */
public class TestP5 {

    public static void main(String[] args) {
        Controller controller;
        Module m1;
        Module m2;
        Module m3;
        Logger logger;
        Clock clock;

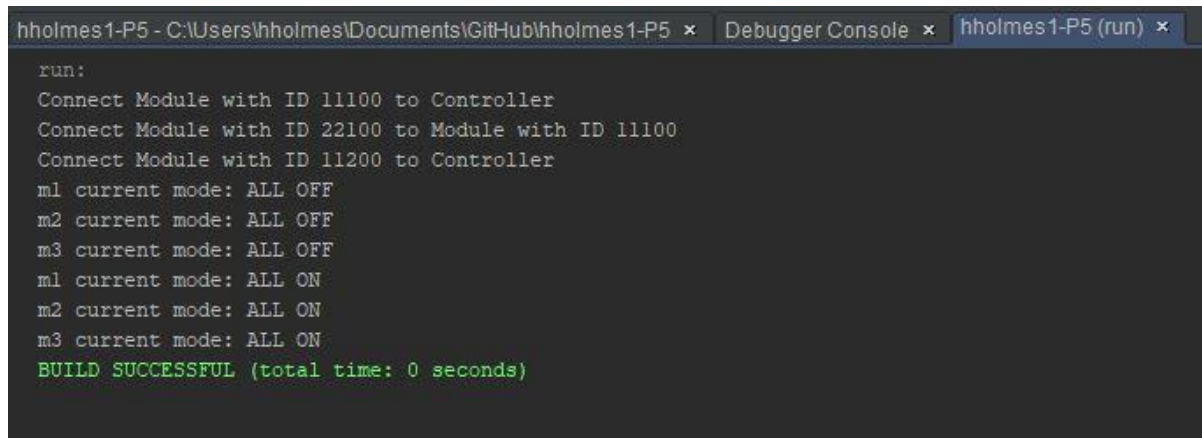
        logger = new PrintLogger(00);
        controller = new Controller(logger);
        m1 = new Module("GRID", 10, 10, logger);
        m2 = new Module("STRING", 10, logger);
        m3 = new Module("GRID", 10, 10, logger);
        clock = new Clock();
        controller.connect(m1, 0, 0);
        m1.connect(m2);
        controller.connect(m3, 0, 1);

        System.out.println("m1 current mode: " + m1.modeString);
        System.out.println("m2 current mode: " + m2.modeString);
        System.out.println("m3 current mode: " + m3.modeString);

        clock.run(controller, "ALL ON");

        System.out.println("m1 current mode: " + m1.modeString);
        System.out.println("m2 current mode: " + m2.modeString);
        System.out.println("m3 current mode: " + m3.modeString);
    }
}
```

Fig. 5. TestP5.java



```
run:
Connect Module with ID 11100 to Controller
Connect Module with ID 22100 to Module with ID 11100
Connect Module with ID 11200 to Controller
m1 current mode: ALL OFF
m2 current mode: ALL OFF
m3 current mode: ALL OFF
m1 current mode: ALL ON
m2 current mode: ALL ON
m3 current mode: ALL ON
BUILD SUCCESSFUL (total time: 0 seconds)
```

Fig. 6 Test Output

## DISCUSSION

Admittedly, the resulting Light Module Controller program could possess more features to demonstrate its flexibility. With limited capabilities the project demonstrates the power of object-oriented design to break down a complex system into simpler parts. Many elements are able to pass information quickly and efficiently without the need to rewrite segments of code or repeatedly making copies of variables. Within this object-oriented design project there were elements that demonstrated the difficulty of work without some of the OOP luxuries. When designed the messages that would be used to send commands to the modules, it was tempting to use a `setMode` method from the controller to simply tell the module what it should do. While most, if not all, other components of this system rely on the capacity of Objects, the difficulty of this particular task was a powerful tool in highlighting the flexibility of a fully object-oriented system.

Another observation made is that it is quite difficult to plan for what might be needed to complete a project like this one. Some goals were laid out without considering how difficult they might be. Given chances to develop, release, and receive feedback a developer may be able to achieve many more original goals than they would if they attempted to design and build everything at once. Proper planning and scope management are key when developing problem solutions like the one in this project.

## CONCLUSIONS

While this project proved to be challenging, it was very enjoyable to see it begin to work together to solve a complex problem. The project involved learning several new skills with Java and with NetBeans. Skills of Object-Oriented Design were also grown and improved through this work. After work began, many goals and design methods were changed as elements of the problem became clearer. Lessons learned in the project can be well applied to many types of problems and can certainly be used to find more efficient solutions.

## REFERENCES

1. (2006, May, 22) *IEEE Transactions LaTeX and Microsoft Word Style Files* .  
<http://www.ieee.org/portal/pages/pubs/transactions/stylesheets.html>
2. (2017, November, 28) *eBook Libray System.pdf* by Caleb Osburn
3. (2018, October, 15) *P5.pdf* David Green.

## **APPENDIX**

Code written for this project is included in the attached file named “p5-hholmes1.zip”