Department of Electrical Engineering
South Dakota State University

EE465 -- Senior Design II
04/30/2020
Senior Design Project Final Design Report
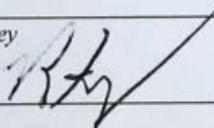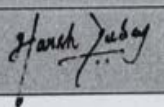
H&M Configuration Dongle

Design Engineers:
Mohamed Ayoub, Harsh Dubey

Project Customer:
Matt Sickler
Daktronics Inc.

Project Approval

Project Governance		
Role *(Add/delete as applicable)*	**Name(s)** *(First initial. Last name)*	**Approval Signature / Date[1]**
Customer	*M. Sickler*	Matt Sickler 10/23/2019 ★
Sr. Engineer 1	*R. Fourney*	10/23/19
Project Manager	*S. Tan*	
Project Team		
Name(s) *(Last name, First name, Middle initial.)*		**Approval Signature / Date**
Ayoub, Mohamed, M.		Mohamed, 10/23/19
Dubey, Harsh		Harsh Dubey 10/23/19

★ **See electronic signature in Appendix A.**

Executive Summary

Project Title: H&M Configuration Dongle
Team Members: Mohamed Ayoub & Harsh Dubey
Customer: Matt Sickler

The display controllers that Daktronics produce are normally accessed over a network using a web browser to configure and control the video displays. If the network fails or the display controller's network settings are corrupted, the display controller becomes inaccessible. Currently, when the display controller's network settings are corrupted, some of them can be recovered using a keyboard and a monitor to configure the network settings using a command-line; other controllers don't have monitor or keyboard support and are either thrown away or reset using complex methods.

To solve this problem, a device was designed to gain physical access to the network settings of Daktronics's display controllers (host device) and to reconfigure these network settings as needed. The design consisted of a display screen module, a controller module, push buttons module, and an application module for the host device. An OLED screen, for the display module, was used to display the host device's system information along with its network settings. Six push buttons, for the push buttons module, were used so that the user can configure the network settings of the host device. A microprocessor, for the controller module, was used to develop a custom communication protocol that allows the microprocessor to receive and send data to the host device using a USB cable. The USB cable was also used to power the system. A .NET C# application, for the application module, was developed using Visual Studios to communicate with the Configuration Dongle. Lastly, a setup file was created for the C# application to allow the user to install and uninstall the application easily.

Overall, the project was successful. The microprocessor was able to interface with the OLED screen using an 8-bit Serial Peripheral Interface (SPI). The microprocessor communicated with the host device using the C# application. The microprocessor retrieved the network settings of the host device and displayed those settings on the OLED screen. Additionally, the microcontroller was able to use the push buttons for user input and was able to configure the network settings of the host device based on the user's input. The C# application received requests from the microprocessor and replied with the requested data or applied changes to the network settings of the host device. The C# application's setup file successfully installed the C# application on the host device.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Definition of Problem

Daktronics is a company based in the U.S. that designs, manufactures, sells, and services video displays, scoreboards, digital billboards, and related products. The display controllers that Daktronics produces are Linux based machines that are normally accessed over a network using a web browser to configure and control the video displays. If the network fails or the display controller's network settings are corrupted, the display controller becomes inaccessible. This prevents the user from accessing the display controller to implement necessary changes to the video displays.

Currently, older models of the display controllers have keyboard and display support, which allows technicians to change the network settings of the display controller directly through the terminal. Newer models of the display controllers, however, do not have keyboard and display support and are shipped back to Daktronics. Daktronics then either recovers those units using complex methods or marks the units as "broken" and disposes them. This wastes time and money for Daktronics. A device is needed to gain physical access to the network settings of those controllers (host devices) and to reconfigure these network settings as needed.

1.2 Background

One possible solution makes use of a Wi-Fi dongle, having the display controller connect to the ad hoc network generated by the Wi-Fi dongle, [5]. However, this solution is of no use if the network settings are corrupted. In this case it becomes difficult to access the display controller through any kind of network, [4].

A second possible solution makes use of a Bluetooth dongle to communicate with the display controller, [1]. In this case, the technician would still have to change the configuration settings which is a complex procedure according to Daktronics. The technician could also use a smartphone and connect to the display controller using a USB cable and then make use of specialized application software in the smartphone to change the network settings, [2]. However, Daktronics company policies preclude the use of personal smartphones to change the settings of company products.

Therefore, developing a Configuration Dongle would be the preferred solution. The dongle should be inexpensive, compact, easy to carry, powered by the host device, and does not need a network connection to communicate with the host device.

2. Functional Design Requirements

2.1 Objectives

The primary objective of the project is to develop and build a device (Configuration Dongle) that will communicate with Daktronics's host device using USB to read and to configure the host device's network settings. Another objective of the project is to design the communication protocol of the Configuration Dongle so that the beginning and the end of a message can be detected unambiguously. Additionally, the device must be able to detect dropped or corrupted data and recover that data. Finally, the communication protocol of the device should be designed so it is backward and forward compatible and should have the Capability Detection feature.

2.2 Specifications

The specifications for the project are listed below:

1. The Configuration Dongle must include at least a 16x2 LCD/OLED display or a matrix OLED display to display the system information and current network settings of the host device.
2. The Configuration Dongle must include at least 3 push buttons on the right-hand side of the PCB.
3. The Configuration Dongle must include one USB port.
4. The Configuration Dongle must be a minimum of 1.5" by 2.5" in size but not larger than 3" by 6".
5. The Configuration Dongle must be able to read and write the IPv4 address of the host device under Static mode.
6. The Configuration Dongle must be able to read and write IPv4 Netmask in Classless Inter-Domain Routing (CIDR) format.
7. The communication protocol between the Configuration Dongle and the host device must use at least 16-bit Cyclic Redundancy Check (CRC).

2.3 Design Constraints

The design constraints of the project are listed below:

- The Configuration Dongle must be powered by a USB.
- Team does not have prior experience with computer networks.
- Team does not have prior experience working with Linux.

3. Technical Design Solutions

3.1 Design Solution

The Configuration Dongle was designed to gain physical access to the network settings of the host device and to reconfigure these network settings as needed. The system level block diagram is shown in Figure 1.
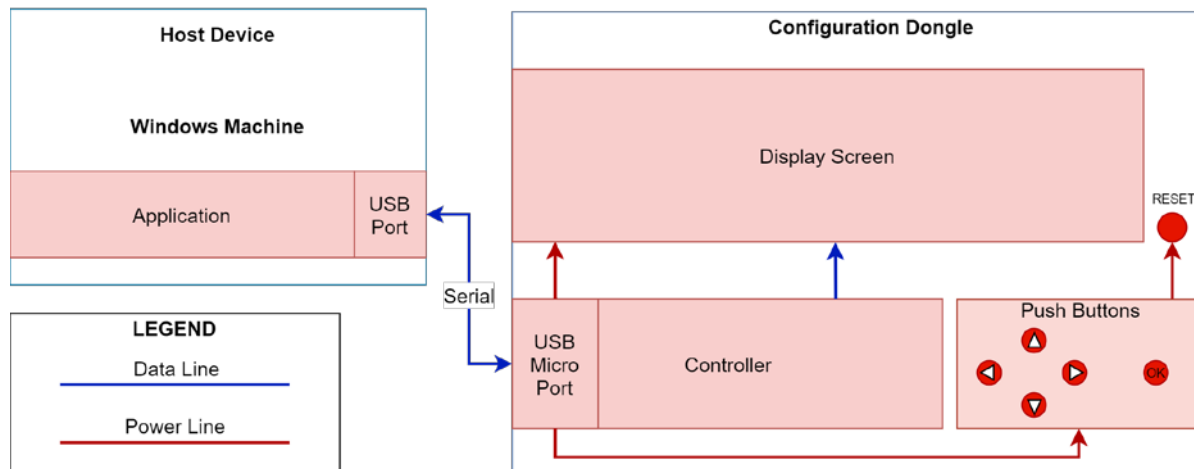

Figure 1. Design Project Block Diagram

The design solution to the problem, in accordance with the functional design requirements discussed in **Section 2**, is divided into several submodules. A display screen was used to display the host device's system information along with its network settings. The navigation module (push buttons) was required for the user to configure the network settings of the host device. A custom communication protocol was developed within the controller module to receive and send data to the host device. The USB port was used to establish a physical connection between the host device and the Configuration Dongle, to initiate the communication protocol, and to power the Configuration Dongle. On the host device's side, the application module was developed to communicate with the Configuration Dongle and to apply changes to network settings of the host device.

3.2 Display Screen

A 4x20 OLED screen, Figure 2, by Newhaven Display was used to display the network settings of the host device and to provide an interactive user interface to configure the network settings of the host device as needed. The OLED screen has operating voltage of 2.2V-5V and has current draw of 75mA. Since Daktronics is in Brookings, SD, the display screen had to be able to sustain harsh weather conditions, and the OLED screen is rated at -40ºC.
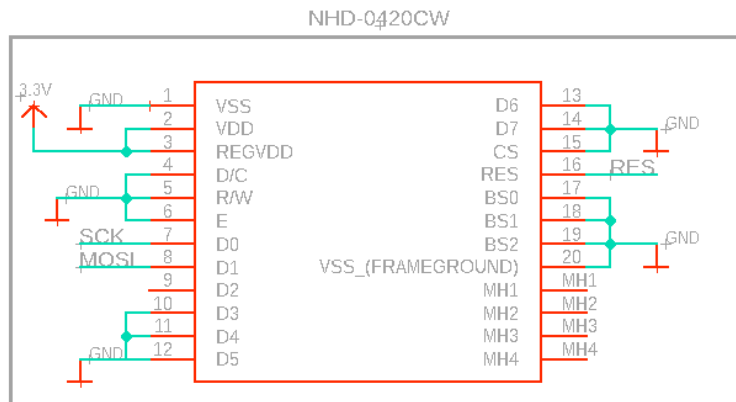
Figure 2. NHD-0420CW OLED Schematic

The OLED screen was successfully interfaced using 8-bit SPI. To setup the OLED screen, the SPI.h header file was included. To setup the OLED screen, series of commands, Figure 3, were used by the recommendation of the manufacturer to initialize the OLED screen. To operate the OLED screen and write functions such as LCD_print(), the instruction commands shown in Table 1 were used.

Figure 3. OLED Display Setup (byte rows = 0x08)

Table 1: OLED Instruction Commands

Instruction	Byte
Clear display	0x01
Return cursor to home	0x02
Display on, cursor off	0x0C
Display on, steady cursor	0x0E
Shift cursor left	0x10
Shift cursor right	0x14

3.3 Push Buttons

For user input, six tactile push buttons, Figure 4, were used, with one button serving as a "RESET" button which restarts the OLED screen and the microprocessor, and the rest serving as navigational push buttons. Two of the buttons served as the "LEFT" and "RIGHT" buttons, which allowed the user to move the cursor through an address they want to change character by character. Two other buttons served as the "UP" and "DOWN, which allowed the user to change the individual characters (0-9) in an address. Lastly, the "OK" button was used as a trigger to an interrupt function called `check_ok()`, Figure 5. The function toggles an integer variable called "ok_flag" when the "OK" button is pressed while the program is displaying the network settings. The "ok_flag" variable allows the user to switch from display mode to configure mode, discussed in Sections *3.4.1* and *3.4.2*.

The `pinMode()` function in the Arduino IDE was used to initialize the five push buttons. The `attachInterrupt()` function was used to enable the interrupt routine when the "OK" button is pressed, Figure 6. The "RESET" button did not need to be initialized as it was hard-wired to the physical reset pin of both the OLED screen and the microprocessor.
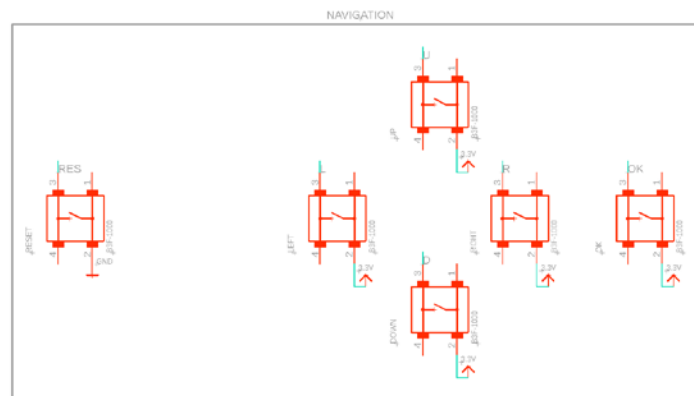

Figure 4. Push Buttons Schematic

```
void check_ok(void)
{
    if(digitalRead(ok) == HIGH)
    {
        while(digitalRead(ok) == HIGH);
        ok_flag = !ok_flag;
    }
}
```

Figure 5. `check_ok()` Code Snippet

```
//PUSH-BUTTONs SETUP------------------------------------
pinMode(up, INPUT_PULLDOWN); //UP pin 13
pinMode(down, INPUT_PULLDOWN); //DOWN pin 12
pinMode(left, INPUT_PULLDOWN); //LEFT pin 11
pinMode(right, INPUT_PULLDOWN); //RIGHT pin 10
pinMode(ok, INPUT_PULLDOWN); //OK pin 6
attachInterrupt(digitalPinToInterrupt(ok), check_ok, CHANGE);
```

Figure 6. Push Button Initialization Code Snippet

3.4 Controller

Adafruit's Feather ATSAMD21 Cortex M0 microprocessor, Figure 7, was used as the control unit in the Configuration Dongle. The microprocessor's built-in USB stack was used to enumerate the designed serial communication protocol, so that the Configuration Dongle can communicate with the host device using a USB cable. The microprocessor program was developed using the Arduino IDE v1.8.10.
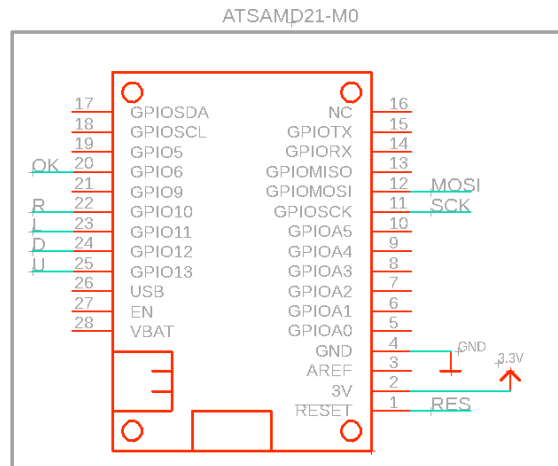

Figure 7. Feather ATSAMD21 Cortex M0 Pinout

When the Configuration Dongle is plugged into the host device, the program starts by setting up the OLED screen, setting up the input pins for the push buttons, and starting the serial protocol. The program can be broken down into two parts: display mode and configure mode, discussed in Sections *3.4.1* and *3.4.2*.

By default, the program starts in display mode and displays all the network settings of the host device. The program switches to configure mode when the "OK" button is pressed, allowing the user to change, and set the network settings of the host device.

3.4.1 Display Mode

The display mode routine starts by requesting the following from the host device: network mode, hostname, IPv4 address, IPv4 netmask, and IPv4 gateway address. The host device then replies with the requested data and the microprocessor prints the current network settings of the host device on the OLED screen, Figure 8.

Figure 8. Network Settings on the OLED Screen

The function `getmode()` for example, Figure 9, starts by sending "Mode" to the host device. The host device then parses the message and replies with the network mode of the host device. The network mode sent by the host device is then read by the microprocessor character by character and is then stored in a string variable. Then, the string is passed to an `LCD_print()` function that prints the string on the first line of the OLED screen. The same process was used for requesting and displaying the hostname and IPv4 gateway address on the OLED screen.

```
void getMode()
{
  mode = "";

  while (!Serial.available()){
    Serial.println("Mode");
  }

  while (Serial.available() > 0)
  {
      char c = Serial.read();  //gets one character from serial buffer
      mode += c; //stores the character in a string variable
  }
  LCD_print(mode, line[0]);
}
```

Figure 9. `getMode()` Code Snippet

IPv4 address and IPv4 netmask, however, was required to be displayed in CIDR notation. The `getIP()` function, Figure 10, follows the same process as `getMode()` but requests the IPv4 address of the host device instead of the network mode. After the IPv4 address is received, the function requests the IPv4 netmask by calling the function `getSub()` and storing the netmask in a string variable. The variable is then converted into a char array and is passed into a `toCIDR()` function that converts the IPv4 netmask into CIDR. The CIDR is then appended onto the received IPv4 address and is printed on the third line of the OLED screen.

```
void getIP()
{
  rcv_ip = "";

  while (!Serial.available()){
    Serial.println("IP");
  }
  while (Serial.available() > 0)
  {
      char c = Serial.read();  //gets one byte from serial buffer
      rcv_ip += c; //stores the character in a string variable
  }

  //SUBNET Mask
  String subnet = getSub();
  char sub_char[16] = {0};
  subnet.toCharArray(sub_char, 16);

  rcv_ip += '/';
  rcv_ip += toCidr(sub_char); //Append subnet mask in CIDR notation onto the IP address
  LCD_print(rcv_ip, line[2]);
}
```

Figure 10. `getIP()` Code Snippet

3.4.2 Configure Mode

When the "OK" button is pressed, the Configuration Dongle goes into configure mode and prompts the user to set the network mode, Figure 11. The user can choose between DHCP and Static network modes by navigating the block cursor using the "UP" and "DOWN" buttons and confirm their choice using the "OK" button.

Figure 11. Set Network Mode User-Prompt

If the user chooses DHCP mode, Figure 12, the Configuration Dongle sends "DHCP" to the host device. Once the host device receives the message, the host device replies with "OK" and the Configuration Dongle shows "Setting DHCP..." on the first line of the OLED screen. The host device proceeds by setting the network mode to DHCP and replies with another "OK" when it done. Once the Configuration Dongle receives the last "OK" message from the host device, the Configuration Dongle shows "Success" on the fourth line of the OLED screen and goes back to the display mode routine.

Figure 12. Setting Network Mode to DHCP

If the user chooses Static mode, the same communication process occurs, but additionally, the user is prompted to first enter their desired IPv4 address, Figure 13, and IPv4 netmask, Figure 14. The user can use the line cursor to navigate through the address using the "LEFT" and "RIGHT" buttons, change the individual numbers in the address from 0-9 using the "UP" and "DOWN" buttons, and confirm using the "OK" button.
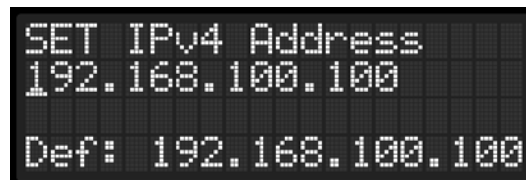
Figure 13. IPv4 Address User-Prompt

Figure 14. IPv4 Netmask User-Prompt

Once the user enters an IPv4 address and IPv4 netmask, the addresses are sent to the host device and the Configuration Dongle shows "Setting Static IP…" on the first line of the OLED screen. After the host device configures the network settings, the Configuration Dongle shows "Success" on the fourth line of the OLED screen, Figure 15, and goes back to the display mode routine.

Figure 15. Setting Network Mode to Static

3.5 PCB
To design the PCB, the circuit schematics of the Configuration Dongle was made using EAGLE Schematic, Figure 16. Using the EAGLE library, the footprints for most of the components were placed and routed using nets. The footprints for the microprocessor and the OLED screen had to be added manually to the EAGLE library. The footprints can either be found online or by emailing the manufacturer.
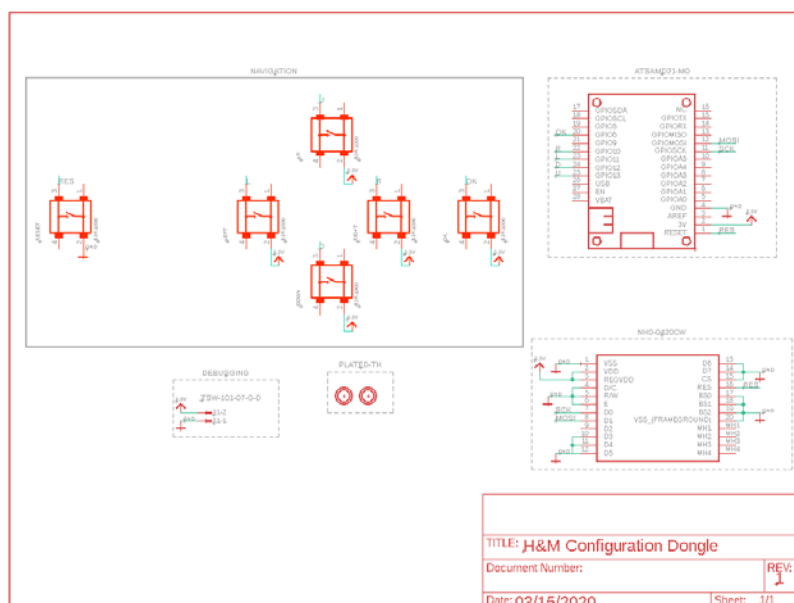
Figure 16. Configuration Dongle Schematic

After the Configuration Dongle schematic was completed, the PCB board, Figure 17, was designed with two layers using EAGLE PCB by clicking on the "Generate/switch to board" button located on the top left hand corner. The board traces were made using Eagle's autoroute function, which automatically routed most of the traces on the top level (red traces) and the rest of the traces on the bottom layer (blue traces). Two solid ground polygons were placed to cover both the top layer and the bottom layer to ensure ease of routing components to ground, Figure 18. The size of the board was then adjusted to be 4.45" wide and 2.65" long.

Figure 17. Configuration Dongle Traces

Figure 18. Grounding Planes (red-top & blue-bottom)

To make the PCB modular, Figure 19, a 20-pin female header was soldered onto the board for the OLED screen. Additionally, a 16-pin female header and a 12-pin female header were soldered onto the board for the microproccessor. This allows the user to easily replace the OLED module and the microproccesor module if they become defective. The user can also add M2.5 spacer standoffs if desired on any of the M2.5 through-holes around the PCB board, mircoprocessor, and OLED screen.
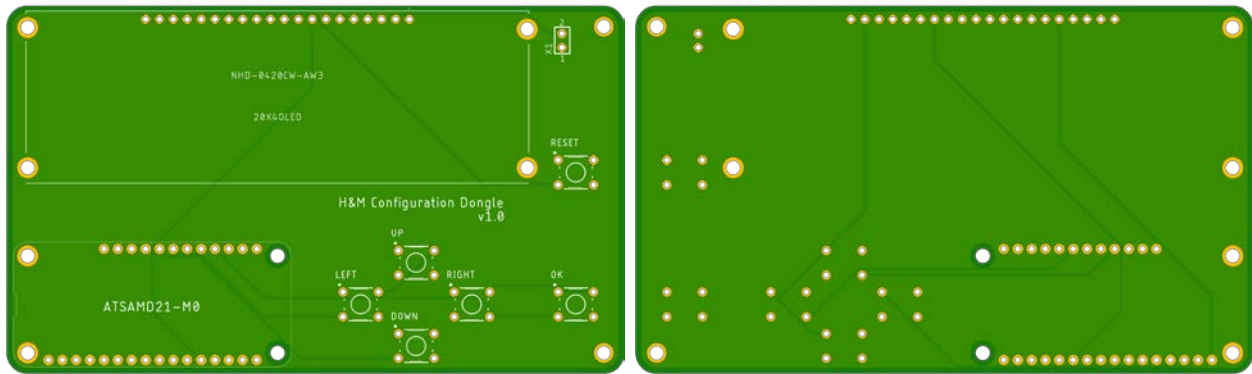
Figure 19. PCB Front & Back View

3.6 Host Device's Application

A .NET C# application was developed using Visual Studios 2019 v16.5.4 to communicate with the Configuration Dongle. The C# application utilized Microsoft .Net Framework v4.8.04161 to compile the program. To create the application, the following headers were used: System, System.Managment, System.IO.Ports, and System.Net. Additionally, a public struct was created to locally store the network settings of the host device locally, Figure 20. The host device's application can be broken down into four sections: getting local network settings, setting DHCP and Static mode, main routine, and application setup.


```csharp
1  using System;
2  using System.Management;
3  using System.IO.Ports;
4  using System.Net;
5
6  namespace testProgram
7  {
       0 references
8      class MainClass
9      {
10         //Local Network Settings Struct
           3 references
11         public struct local_net
12         {
13             public string myMode, myHostname, myIPv4, mySubnet, myCIDR, myGateway;
14         }
15
```

Figure 20. Header Files & Public Struct Code Snippet

3.6.1 Main Routine

The C# main routine, Figure 21, starts by finding and storing the local network settings of the host device using a function called `get_info()`, discussed in **3.6.2** Getting Local Network Settings. After the local network settings have been stored, the program looks for all accessible COM port names and stores them in a variable called "portNames". Using a for-loop, each port in "portNames" is checked using the try function to see if the COM port is open. If the COM port is open, the application assumes it is the Configuration Dongle and setups up the COM port for serial communication. Note that all other COM ports in-use on the host device, such as keyboards, mice, and USB drives, have security systems that disallow them from being discovered so that applications, such as this one, do not disrupt them. This means that all other connected ports are

safe from this application. The serial communication was setup with 9600 baud rate to match the Configuration Dongle. To enable handshake, the variable "serialport.DtrEnable" was set to true; without it, the application cannot receive any data from the Configuration Dongle. After handshake was enabled, the application waits to receive a request from the Configuration Dongle. Once it receives a request, the request is parsed through a series of if-statements and replies to the Configuration Dongle with the requested data. However, if the Configuration Dongle requests to set the network mode to DHCP or Static mode, the application calls either the `setDHCP()` function or the `setStatic()` function, discussed in 3.6.3 Setting DHCP & Static Mode.

Figure 21. Main Routine Code Snippet

3.6.2 Getting Local Network Settings

A function called `get_info()` was developed to obtain and store the network settings of the host device locally. The function, Figure 22, starts by finding the hostname and the IPv4 address of the host device. The function then initializes all the members of the struct mentioned above. This allows the Configuration Dongle to show that the IPv4 address is "127.0.0.1" when it is not connected to any network. The number "127.0.0.1" is an IP address, that is used for localhost loopback purposes. The function then proceeds by creating an instance of the Management class for network adapter settings and gets all the info for all the network adapters available in the host device. A for-loop was then created to parse through the info and find the active network adapter with a network connection. This was achieved by placing an if-statement in the for-loop. The program falls into the if-statement only if an IPv4 address exits in the network adapter. Once the program falls into the if-statement, the network settings of that network adapter is stored in the struct. However, a separate try function was used for the gateway address since assigning gateway addresses is optional in Static mode.

Figure 22. `get_info()` Code Snippet

3.6.3 Setting DHCP & Static Mode

To set the network mode to DHCP, a `setDHCP()` function, Figure 23, was developed. The function starts the same way as `get_info()`, to find the "IPEnabled" network adapter. Once the program falls into the if-statement, the function obtains the parameters and values of the network adapter's DNS settings and stores it in a variable called "ndns", which is then to "null". The function then enables DHCP, and since the "ndns" variable is null, the network adapter can automatically assign the DNS settings accordingly.

Figure 23. `setDHCP()` Code Snippet

To set the network mode to Static, a `setStatic()` function, Figure 24, was developed. The function behaves the same way as `setDHCP()` except it obtains the parameters of the network adatper's "EnableStatic" settings and stores them in the ManagmentBaseObject type "newIP". To

set the IPv4 address and netmask, the received IPv4 address and netmask from the Configuration Dongle were then stored in the member variables "IPAddress" and "SubnetMask" of the "newIP" object. The function then sets the network mode to Static with the IPv4 address and netmask that was received from the Configuration Dongle.

Figure 24. `setStatic()` Code Snippet

Setting the network mode to DHCP or Static mode in Windows requires administrative privileges, and without them, the functions will execute but will not apply. To solve this problem an XML Schema Manifest file, Figure 25, was created. The manifest allows the application to run with administrative privileges. After the manifest was created, the manifest was used by going into the project's properties under Visual Studios and choosing the created manifest, Figure 26. This step was needed, otherwise, an embedded manifest would have been used automatically with default settings, which would not allow the program to run with administrative privileges.

Figure 25. XML Manifest File
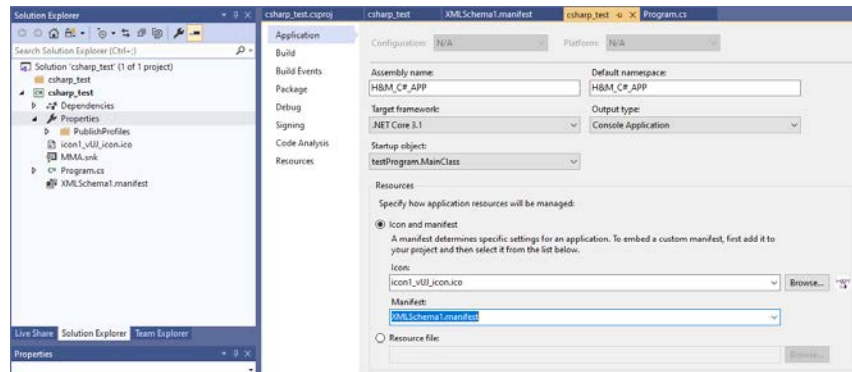
Figure 26. Project Manifest Under Properties

3.6.4 Application Setup

To ensure ease of use, the C# application was published/generated under one .exe file. This was done in Visual Studios by clicking on "Publish [Project Name]" under the "Build" dropdown tab on the top left corner, Figure 27. After being redirected to the publish settings, Figure 28, the "Produce single file", "Enable ReadyToRun compilation", and "Trim unused assemblies" checkboxes were checked. This allow the C# application to be published under a single .exe file.

Figure 27. Publish C# Application

Figure 28. Publish C# Application Settings

To allow the user to install and uninstall the application easily, a setup file was created to install the C# application. The setup file was created u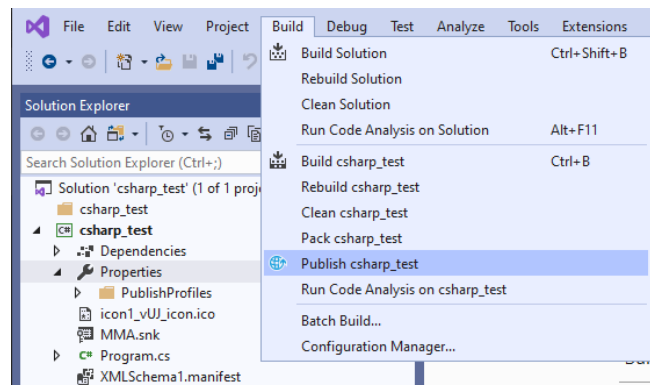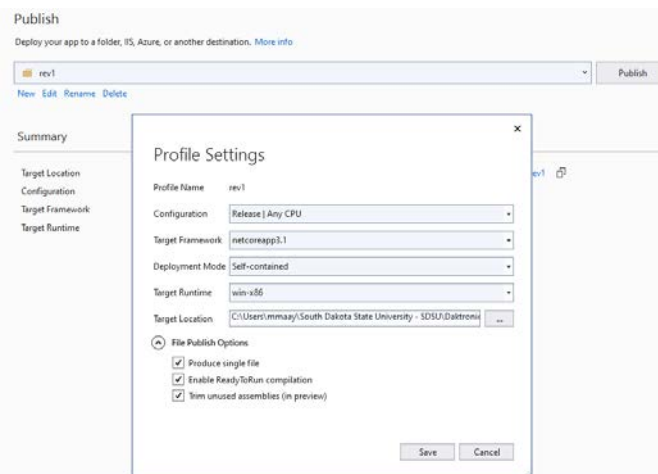sing Inno Setup Compiler 6.0.4(u). The code, Figure 29, was developed using Inno's user-manual. The define section, was used to provide the compiler with the name of the application, application version, publisher name, and the name of the .exe file. The [Setup] section was made to give the compiler the path to where the setup file should be outputted and the name of the setup file. The [Tasks] allows the setup file to ask the user to create a desktop shortcut of the C# application when they install the application. The [Files] section provides the compiler for the source file path of the C# application. The [Run] section was automatically generated by Inno to run the C# application after the user is done installing the application. This section was also modified to allow the C# application to also run on startup in the background with administrative privileges, no-login needed. This would allow Daktronics to be able to plug-in the Configuration Dongle at any-time if the C# application is installed on their servers. This will also prevent Daktronics from needing to reopen the C# application if their server accidently restarts. Note that this feature worked for nine of ten Windows computers that were tested.

Figure 29. Setup File Using Inno Code Snippet

4. Project Costs and Bill of Materials

The Configuration Dongle project was given a $400 budget. All components that were purchased for the project and their respective costs were recorded in Table 2. The table shows that the Configuration Dongle project was completed under budget. Additionally, all the components that were used in the final product were highlighted in green and the components that were not used were highlighted in red. The 128x32 OLED ATSAMD21 microprocessor addon came with the backup microprocessor as a bundle, but even though it was easier to interface, it was unused due to its small size. The seven USB type B ports was originally intended to be used in the first few design iterations of the Configuration Dongle. However, the ATSAMD21 microprocessor already has a USB type-B micro port, which is why the USB type B ports that were purchased were not used.

Table 2: Bill of Material

Item Description	Retail Cost ($)	Link/Contact Info
Display Screen	----------	--
4x20 OLED Screen	31.77	
4x20 OLED Screen (Backup)	31.77	https://www.digikey.com/short/zz4wc8
4x20 OLED Screen (Backup)	27.84	
Subtotal	111.14	
Controller	----------	--
ATSAMD21 Cortex M0	26.01	https://www.adafruit.com/product/2772
ATSAMD21 Cortex M0 (Backup)		https://www.adafruit.com/product/2772
12-pin & 16pin female Header Pins		https://www.amazon.com/gp/product/B01ABIT8FI/ref=ppx_yo_dt_b_asin_title_o05_s02?ie=UTF8&psc=1
128x32 OLED, M0 add-on	43.89	https://www.amazon.com/gp/product/B01BMRDVSC/ref=ppx_yo_dt_b_asin_title_o05_s01?ie=UTF8&psc=1
Subtotal	69.9	
PCB, Assembly Parts, and Push Buttons	----------	--
JLCPCB v0.1	$53.45	https://jlcpcb.com/
JLCPCB v0.2 (Final Version)	$40.85	https://jlcpcb.com/
25 SWITCH TACTILE SPST-NO		https://www.digikey.com/short/zz4wtv
5 CONN HDR 20POS 0.1 TIN		https://www.digikey.com/short/zz4w35
5 CONN HDR 12POS 0.1 TIN		https://www.digikey.com/short/zz4w3n
5 CONN HDR 16POS 0.1 TIN	$32.26	https://www.digikey.com/short/zz4wq8
10 Male Header Pin, 40 Pin Header Strip	4.57	https://www.amazon.com/gp/product/B07PKKY8BX/ref=ppx_yo_dt_b_asin_title_o08_s00?ie=UTF8&psc=1
7 USB Type B Ports	35.54	https://www.digikey.com/short/zz4wqh
Subtotal	166.67	
Total	347.71	

5. Project Status

5.1 Progress Review

All the objectives and specifications, discussed in **Section 2**, of the project have been completed on time and under budget. Additionally, the team developed a PCB for the project and a setup file for the C# program to make the Configuration Dongle easy to use.

5.2 Gantt Chart
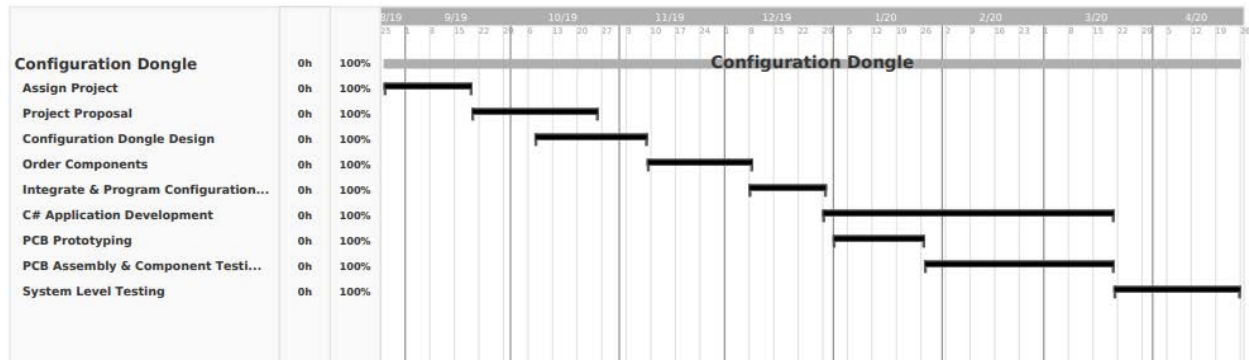
Figure 30. Gantt Chart

6. Team Member and Project Responsibilities

This section of the report highlights the team members contribution to the project, which was recorded in Table 3.

Table 3: Percent responsibility for each team member on project

Name	Moe	Harsh
Design Solutions	50%	50%
Interfacing OLED Screen	50%	50%
OLED Screen Hardware	50%	50%
Soldering	50%	50%
Push Buttons	50%	50%
OLED GUI	100%	0%
PCB	100%	0%
Host Device's Application	100%	0%
Host Device's Application Setup File	100%	0%
Controller Firmware	0%	100%
CDC	0%	100%
Communication Protocol Design	0%	100%
3D Enclosure Design	0%	100%
Testing/ Debugging	60%	40%
Documentation	70%	30%
Presentation	50%	50%

7. Conclusions and Future Works

Overall, the project was successful. The microprocessor was able to interface with the OLED screen using an 8-bit Serial Peripheral Interface (SPI). The microprocessor communicated with the host device using the C# application. The microprocessor retrieved the network settings of the host device and displayed those settings on the OLED screen. Additionally, the microcontroller was able to use the push buttons for user input and was able to configure the network settings of the host device based on the user's input. The C# application received requests from the microprocessor and replied with the requested data or applied changes to the network settings of the host device. The C# application's setup file successfully installed the C# application on the host device and was able to launch automatically in the background when the host device operating system restarts for any apparent reason.

For future work, it is recommended to continue developing the C# application to make it compatible with Linux systems. It is also recommended to design and built an enclosure for the Configuration Dongle to extend the device's shelf life.

References

[1]   Ferguson, Greyson, et al. "How to Connect a Printer Using Bluetooth." *Techwalla*.
         Accessed on: Oct. 8, 2019. [Online]. https://www.techwalla.com/articles/how-to-connect-
         a-printer-using-bluetooth.

[2]   Jonas. "Change Windows IP Address Settings Remotely with Psexec and
         Netsh." *Jonamiki.com*, 26 Jan. 2018. Accessed on: Oct. 8, 2019. [Online].
         http://jonamiki.com/2014/12/19/change-windows-ip-address-settings-remotely-with-
         psexec-and-netsh/.

[3]   Mitchell, Bradley. "How to Change Your IP Address (And Why You Might Want
         To)." *Lifewire*, Lifewire, 4 Oct. 2019. Accessed on: Oct. 8, 2019. [Online].
         https://www.lifewire.com/change-your-ip-address-818150.

[4]   "Network Configuration." *Network Configuration - an Overview | ScienceDirect Topics*.
         Accessed on: Oct. 8, 2019. [Online]. https://www.sciencedirect.com/topics/computer-
         science/network-configuration.

[5]   Ruddy. "WiFi Dongle - What Is It and How Does It Work?" My Webspot - Pocket WiFi
         Rental, 10 Jan. 2019. Accessed on: Oct. 8, 2019. [Online]. https://www.my-
         webspot.com/blog/wifi-dongle-how-it-work/.

Appendix A: System Setup Procedure

This section of the report provides the system requirements, shown in Table 4, for installing the C# application and a step by step instructions on how to install the C# application.
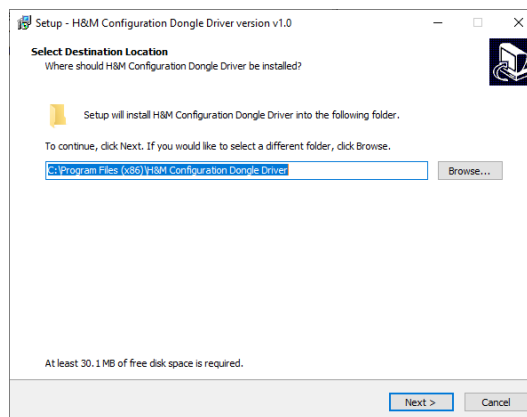
Table 4: System Requirements

| Operating System	Windows 7 SP1, Windows 8.1, or Windows 10
Architecture	x64 or x86 (64-bit or 32-bit)
Processor	1.6 GHz or higher
RAM Needed	33mb or higher
Drive Size	30mb or higher

Procedure:

1. Run "H&M_CD_Setup.exe".
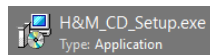
2. Choose where you want to install the H&M Configuration Dongle Driver and then Click "Next".
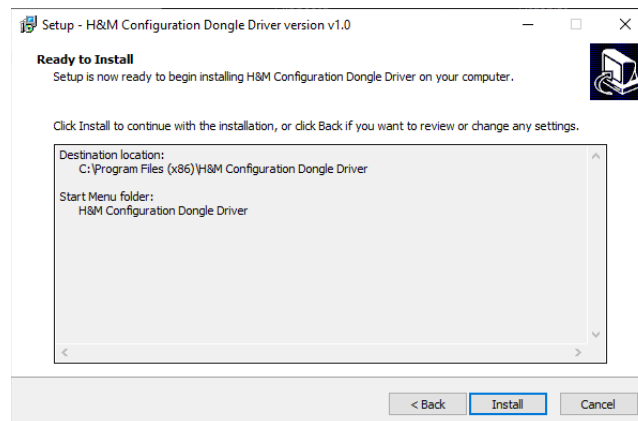

3. If you wish to not create a "Start Menu Folder", check the "Don't create a Start Menu folder" checkbox and click "Next".
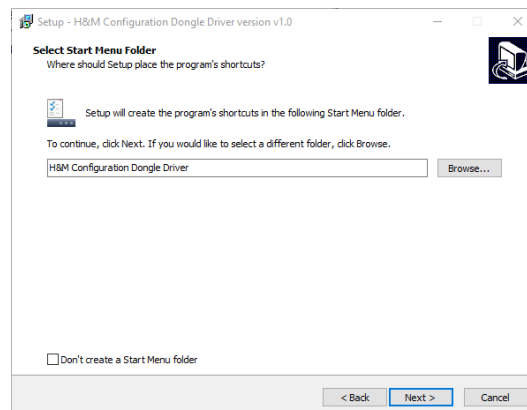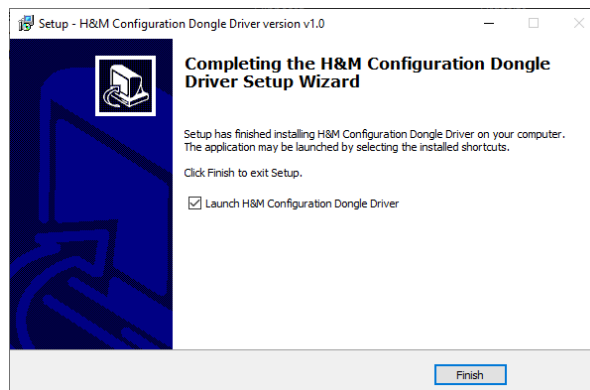
4. Check the "Create a desktop shortcut" checkbox if you would like to have a desktop shortcut of the application and click "Next".
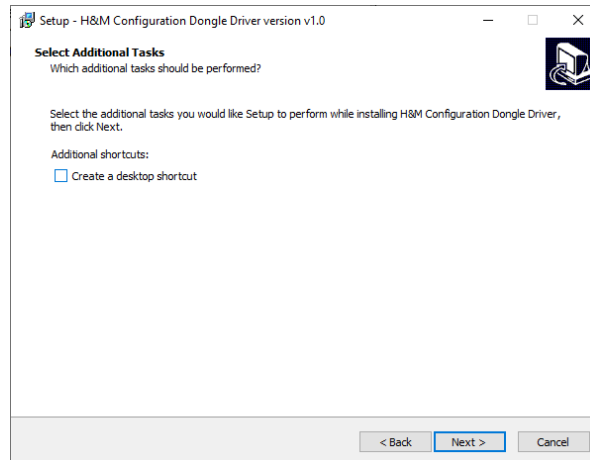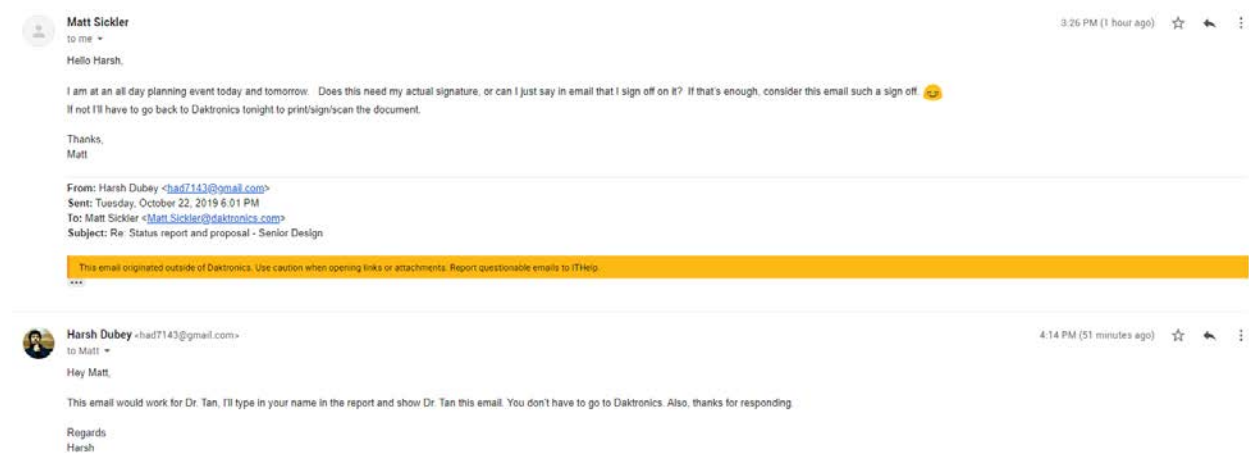

5. Click "Install".

6. Check the box to launch the H&M Configuration Dongle Driver and then click "Finish".

Appendix B: Electronic Approvals

Matt Sickler's Signature

Appendix C: Document Change Control

Revision History			
Date	**Version**	**Synopsis of Change**	**Author**
04/26/2020	0.1	Rough Draft	Moe
04/27/2020	0.2	Applied SDSU Writing Center's recommended changes	Moe
04/28/2020	0.3	Added Executive Summary	Moe
04/28/2020	0.4	Fixed TOC, List of Figures, and List of Tables	Moe
04/30/2020	1.0	Draft submitted	Moe
05/06/2020	1.1	Applied Dr. Tan's recommended changes to the FDR	Moe
05/06/2020	1.2	Added a Conclusions & Future work section	Moe
05/06/2020	2.0	Submitted the final version of FDR	Moe