WI-FI THERMAL PRINTER

BY: Scott Clandinin

ELECTRONIC SYSTEMS ENGINEERING TECHNOLOGY

SASKATCHEWAN POLYTECHNIC MAY 2016

# Acknowledgements

Firstly, I would like to thank the project managers Anthony Voykin, Larry Cruse, and Michael Lastante for their help and advice on the project through the design process. After completing this project, I have a greater understanding of why they continually reminded the class to “keep it simple” when choosing our projects.

I’d also like to express my appreciation for the help of Jerome Wagner for his great help in troubleshooting issues with my printed circuit board. Finally, I’d like to thank my other instructors at Saskatchewan Polytechnic for their direct or indirect impact on the process.

# Abstract

There is a growing trend towards integrating devices with wireless internet to allow information to be easily and instantly be sent and received. This project attempts the receiving of data through communication with the internet by Wi-Fi to retrieve data and print the data out on a receipt printer. At the touch of a button a user can have the current weather conditions in Saskatoon and the word-of-the day printed off for them. As Wi-Fi technology and accessibility increases, the use of it will massively expand in the field of electronics.

# Executive Summary

Transmitting and receiving information over long distances is nothing new in today’s world, and it is often crucial for one to be able to instantly send and receive data. However, communication to remote areas can be expensive, and sometimes not economically feasible. At times, it is even necessary for a person to physically travel to remote locations just to retrieve data of interest. Not only is this necessary for businesses, but the average consumer is also interested in instant communication.

This project investigates the benefits of using Wi-Fi to instantly retrieve information from an online source. The Wi-Fi Thermal Printer allows a person to get the current weather conditions and the word-of-the day at the touch of a button. This is ideal for consumers as it allows them to easily and quickly get the information they want without having to search and sort through unimportant details. It will be beneficial to embrace the use of Wi-Fi in business and everyday life, as it will only become more prominent in how we communicate with the world around us.

# Table of Contents

[Acknowledgements 1](#_Toc450159656)

[Abstract 2](#_Toc450159657)

[Executive Summary 3](#_Toc450159658)

[Table of Contents 5](#_Toc450159659)

[List of Figures and Tables 8](#_Toc450159660)

[1.0 Introduction 9](#_Toc450159661)

[2.0 Mechanical Description 10](#_Toc450159662)

[2.1 Paper Feeder 12](#_Toc450159663)

[2.1.1 Roller 12](#_Toc450159664)

[2.1.2 Gear 12](#_Toc450159665)

[2.2 Opening Mechanism 12](#_Toc450159666)

[2.2.1 Latch 12](#_Toc450159667)

[2.2.2 Rod 13](#_Toc450159668)

[2.2.3 Flip Opener 13](#_Toc450159669)

[3.0 Process Description 14](#_Toc450159670)

[3.1 Background 14](#_Toc450159671)

[3.2 API Configuration 14](#_Toc450159672)

[3.2.1 Weather API 15](#_Toc450159673)

[3.2.2 Word-of-the-Day API 15](#_Toc450159674)

[3.3 HTTP Communication 16](#_Toc450159675)

[3.4 Data Parsing 17](#_Toc450159676)

[3.4.1 JavaScript Object Notation 17](#_Toc450159677)

[3.4.2 Parsing JSON 18](#_Toc450159678)

[4.0 Instructions for Operation 20](#_Toc450159679)

[4.1 Parts Included 20](#_Toc450159680)

[4.2 Getting Started 21](#_Toc450159681)

[4.2.1 Installing the Paper 21](#_Toc450159682)

[4.2.2 Powering the Weather Station 22](#_Toc450159683)

[4.3 Operation 22](#_Toc450159684)

[4.3.1 Testing Paper Feed 22](#_Toc450159685)

[4.3.2 Printing Current Weather Data 23](#_Toc450159686)

[4.4 Troubleshooting 23](#_Toc450159687)

[5.0 Theory of Operation 25](#_Toc450159688)

[5.1 Wireless Internet 25](#_Toc450159689)

[5.2 Application Program Interface (API) 26](#_Toc450159690)

[5.2.1 World Weather Online 27](#_Toc450159691)

[5.2.2 Open Weather Map 28](#_Toc450159692)

[5.3 HTTP Requests 28](#_Toc450159693)

[5.4 Receipt Printer Commands 29](#_Toc450159694)

[6.0 Component Selection 30](#_Toc450159695)

[6.1 Microcontroller 30](#_Toc450159696)

[6.2 Wi-Fi Module 31](#_Toc450159697)

[6.3 Thermal Receipt Printer 33](#_Toc450159698)

[6.4 Power Supply 35](#_Toc450159699)

[6.4 3.3V Voltage Regulator 35](#_Toc450159700)

[6.5 LCD 36](#_Toc450159701)

[6.6 Momentary Push Button 36](#_Toc450159702)

[6.7 Notification LED 37](#_Toc450159703)

[6.8 Power Switch 37](#_Toc450159704)

[6.9 Case 37](#_Toc450159705)

[7.0 Conclusion 39](#_Toc450159706)

[8.0 Recommendations 41](#_Toc450159707)

[References 43](#_Toc450159708)

[Glossary of Abbreviations 44](#_Toc450159709)

[Appendices 45](#_Toc450159710)

[A. Block Diagram 45](#_Toc450159711)

[B. Circuit Schematic 46](#_Toc450159712)

[C. Printed Circuit Board – Top and Bottom 47](#_Toc450159713)

[D. Printed Circuit Board Bottom 48](#_Toc450159714)

[E. Bill of Materials 49](#_Toc450159715)

[F. Gantt Chart 50](#_Toc450159716)

[G. Software 51](#_Toc450159717)

[G.1 Main Code 51](#_Toc450159718)

[G.2 Port Initialization 55](#_Toc450159719)

[G.3 Defines 56](#_Toc450159720)

[G.4 Function Prototypes 58](#_Toc450159721)

[G.5 Wi-Fi 59](#_Toc450159722)

[G.6 LCD 65](#_Toc450159723)

[G.7 Thermal Printer 67](#_Toc450159724)

[G.8 Weather Parsing 69](#_Toc450159725)

[G.9 Delay Functions 71](#_Toc450159726)

[G.10 USART 2 Enable 72](#_Toc450159727)

# List of Figures and Tables

[Figure 1 - Printer Dimensions 10](file:///C:\Users\User\Documents\Tech%20Thesis%20Report.docx#_Toc450163989)

[Figure 2 - JSON Format Example 17](#_Toc450163990)

[Figure 3 - Weather Station Top View 18](file:///C:\Users\User\Documents\Tech%20Thesis%20Report.docx#_Toc450163991)

[Figure 4 - Correct Paper Orientation 19](file:///C:\Users\User\Documents\Tech%20Thesis%20Report.docx#_Toc450163992)

[Figure 5 - BT-2727 Enclosure 35](#_Toc450163993)

[Figure 6 - Block Diagram 42](file:///C:\Users\User\Documents\Tech%20Thesis%20Report.docx#_Toc450163994)

[Figure 7 - Circuit Diagram 43](file:///C:\Users\User\Documents\Tech%20Thesis%20Report.docx#_Toc450163995)

# Introduction

This report outlines the features and design process of my technical thesis project, the Wi-Fi Thermal Printer. This project is a program requirement of the second year of Electronic Systems Engineering Technology at Saskatchewan Polytechnic. The goal of this project was to combine the use of wireless internet and a thermal receipt printer to allow a user to retrieve specific information from an online source at the push of a button. The success of the project was mainly based on how the specification requirements set by the project managers were met.

This project is required to find current weather information for Saskatoon through wireless internet, and print off the results onto receipt paper. The device is also required to indicate when the printer is out of paper. In addition to this, the device will also print off the “word of the day” including a description of the word. This report will detail descriptions of what the device is and how it works, as well as describe part the design process and the decisions that lead to the final product. Finally, I will conclude with how the project turned out and the challenges that were faced, and recommend potential improvements.

# **2.0 Mechanical Description**

This section is a mechanical description of the thermal receipt printer I am using for my technical thesis design. The printer is the only part of the project with any mechanical motion besides the button and power switch. I’m using the CSN-A2 printer to display current weather data to the user. It is a small rectangular unit with a width of 111mm, a length of 65mm, and a height of 57mm. The top of the printer has a façade that extends 7mm above the main rectangle body, and 4mm out over the sides in each direction. The outer shell of the unit is mainly smooth, black plastic.

The main moving part of the printer that the user interacts with is the lid, with a surface area of 65mm by 54mm. This can be opened to change the paper roll. There is a small slit under the front of the lid where the paper is fed through. At the edge of the lid is the paper feeder, consisting of a free spinning roller with a gear at the end. Below that, and on the side of the lid is the opening mechanism. The mechanism consists of a latch and a small oval connected on either side of a rod attached to the roof of the lid.

Figure 1 from the CSN-A2 User Manual details the dimensions and appearance of the printer. From top-to-bottom, and left-to-right, the figure shows the 3-dimensional view, top view, side view, bottom view, and main body dimensions. All values are in millimetres.

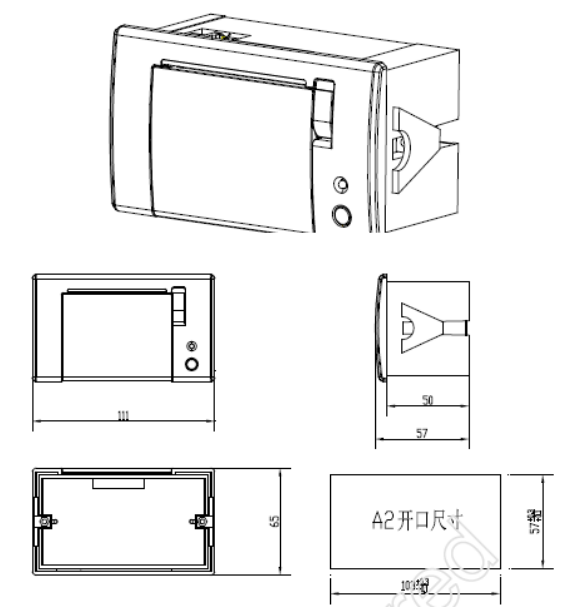


Figure 1 - Printer Dimensions

## 2.1 Paper Feeder

A 65mm long rod with a roller section and a gear on the end serves to feed the paper off the roll.

### 2.1.1 Roller

The roller is 50mm long and has a diameter of 5mm. It is made of a gripping plastic that helps pull on the paper to roll it out. The roller is mounted to be free spinning.

### 2.1.2 Gear

A small plastic gear is located on the end of the paper feeder rod opposite the latch. It has a 6mm radius and is 2mm wide. When the lid is closed, the gear makes contact with another small gear below. This spins the roller to feed the paper through the slot.

## 2.2 Opening Mechanism

The opening mechanism is a rod with a latch on one end, and an oval shaped piece of on the opposite site designed to pop the lid open.

### 2.2.1 Latch

The latch serves to allow the user to open the lid, and is located right beside it. Half of it is smooth plastic, curving down towards the middle. The other half closer to the edge is corrugated for grip when opening. It has a length of 17mm and a width of 8mm.

### 2.2.2 Rod

A 61mm long plastic rod with a diameter of 3mm sits between the latch and the flip opener. When the lid is closed, the rod is held in place by a very small clip. When closed, the lid sits under a lip that keeps it shut. Opening the lid causes the rod to be free spinning.

### 2.2.3 Flip Opener

Attached to the end opposite the latch is a small ovular bit of plastic 7mm long. Pulling the latch open causes this to rotate downwards. As it rotates down, it pushes against a ledge 6 mm from the top of the printer. This pops the lid out from under the lip allowing access to the paper storage.

# 3.0 Process Description

## 3.1 Background

The Wi-Fi module chosen for this project allows sending and receiving data wirelessly from the internet. This will be utilized to get the current temperature and wind speed in Saskatoon. Several ordered steps need to be done in order to utilize this device. The process of online data acquisition is done through what is called an Application Program Interface (API). A websites API serves as an easy and efficient way for site users to access data of interest. A website like www.openweathermap.org allows anyone to easily access weather data through their API. It is essentially a service that allows one to access a custom URL that contains data of interest that can be easily parsed and used. The main steps include selecting the data of interest, using the Wi-Fi module commands to retrieve data from a website, and parsing the data that has been returned.

## 3.2 API Configuration

There are slightly different methods of setting up the API URL for each service. The process will be detailed below for the two API services used in this project. Open Weather Map is used for weather data, and Wordnik is used for the word-of-the-day.

### 3.2.1 Weather API

To begin, the data API URL of interest needs to be set up with the city and data parameters to help narrow in on useful data. With the service Open Weather Map, it is necessary to create a profile with them to receive an API “key” that is needed to make an API request. The free version of this service allows a person to request current weather data, or a 5 day forecast with weather predictions every 3 hours. Parameters must be appended to the API URL in order to select the desired unit (imperial or metric), and to validate the API call with the unique API key. These parameters are appended to the link in the form of “&parameter=value”. The completed URL to receive metric data of current weather in any city would appear as:

http://api.openweathermap.org/data/2.5/find?q=<city>&units=metric&appid=<API-Key>

### 3.2.2 Word-of-the-Day API

Wordnik is a free service with many functions such as defining words, listing related words, pronunciation of words, and others. Similar to Open Weather Map, it also requires an API “key” for data requests. This particular API URL requires little customization. The URL needed to request the word of the day would be:

http://api.wordnik.com/v4/words.json/wordOfTheDay?api\_key=<API-Key>

## 3.3 HTTP Communication

To communicate with the website servers, use of HTTP is required. There is a step-by-step process to follow for communication. First, the microcontroller needs to send commands to the Wi-Fi module that are needed to make the API Request for the URL determined in the first step. Next, it needs to connect to the Saskatchewan Polytechnic wireless internet. This is done with a simple command that only needs to know the network name and password (if there is one). The next step is to make a connection with the website domain. The parameters of this request are the domain that will be connected to, the type of data transfer used, and the port number.

To properly use this to connect to the website, the data transfer type will be Transmission Control Protocol (TCP), and the port that deals with HTTP requests is 80. Next, it is required to specify the length of the GET request that will be sent to the server. GET (not an acronym) is an HTTP request method that can be used to pull data from the URL of interest. The main part of the GET request is the customized URL from the first part. This tells the server to send the data from that URL. The length of this request is simply the number of all characters and spaces in the GET request line. Finally, the GET request is sent to the server, which replies with the raw data.

## 3.4 Data Parsing

The last step is parsing the data that has been received from the server.

### 3.4.1 JavaScript Object Notation

JavaScript Object Notation (JSON) is a format for the interchange of data. This is typically the default data format for API services. Other data structures include Hyper Text Markup Language (HTTP) and less commonly Comma Separated Values (CSV). The structure of JSON is a collection of name/value pairs that allows for identifiers to be associated with values and other identifiers (Introducing JSON). The length of data is not always the same, but the identifiers are, so a person can always find the data associated with the identifier. The order of identifiers is nearly always exactly the same each time, allowing byte-by-byte parsing techniques to be used.

### 3.4.2 Parsing JSON

With the limited RAM of microcontrollers, it isn’t always possible or ideal to parse the data with typical JSON parsing techniques. For large amounts of data, it is necessary to parse byte-by-byte. This is possible by taking note of the patterns of data to be expected. Unwanted data can be skipped by waiting until characters such as parenthesis and colons appear. Since identifiers and values will follow the same order for each API call, it is possible for the programmer to know that the desired piece of data will always appear in-between a certain pair of quotation marks. An example of the JSON format can be seen in Figure 2 from w3schools.com.

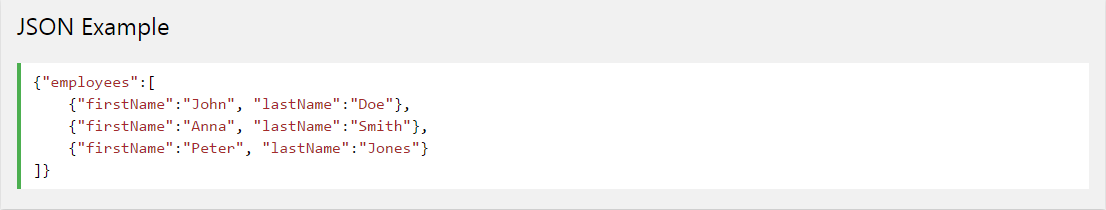


Figure 2 - JSON Format Example

# 4.0 Instructions for Operation

This quick and easy-to-follow guide will help you set up, use, and maintain the Wi-Fi Weather Printer. The device is powered from a regular wall outlet and connects automatically to your wireless network. No tools are needed to get started, and there is no assembly required. In a few simple steps you will be ready to print out the current weather in your city.

## 4.1 Parts Included

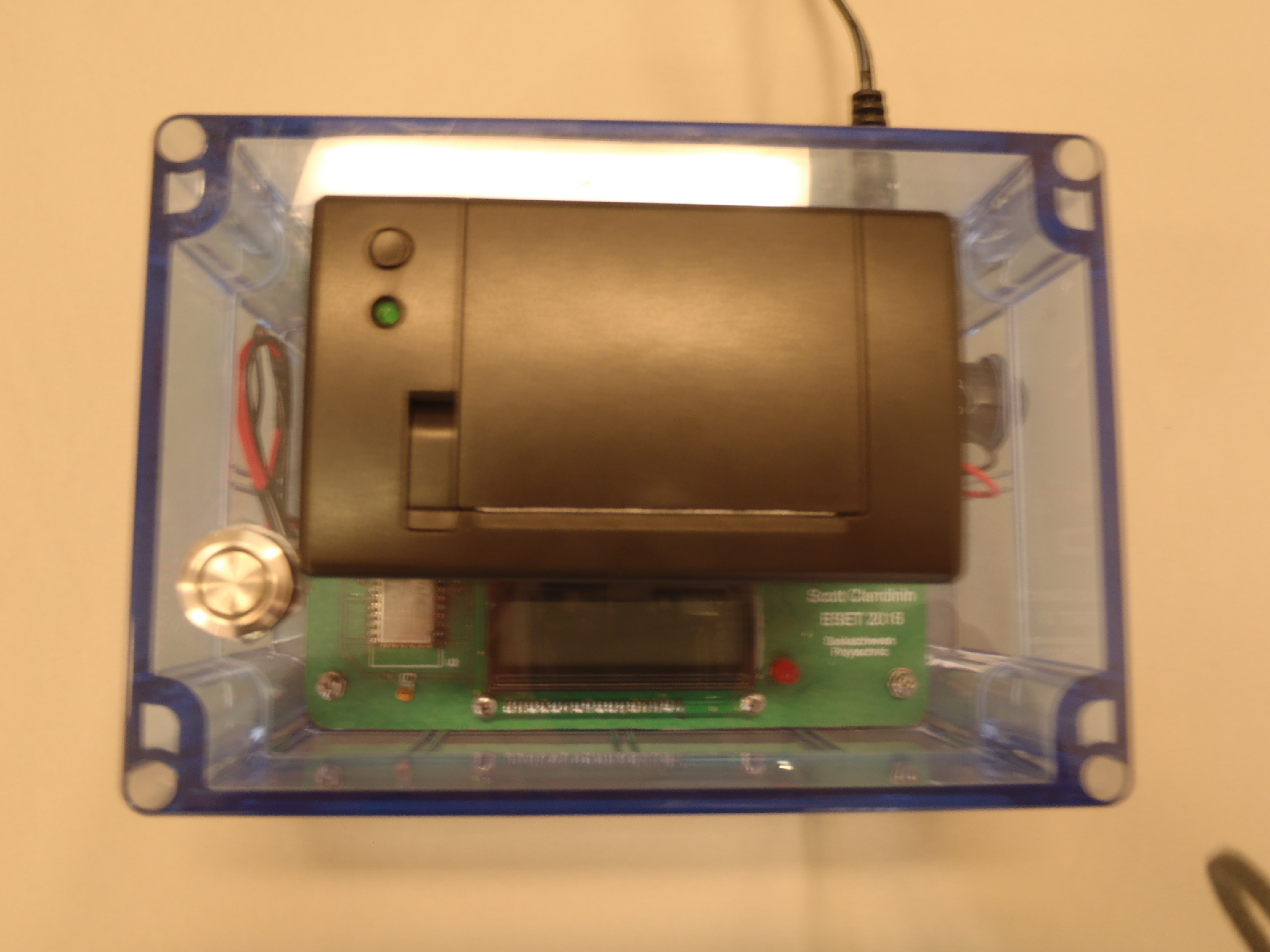
* 1x Weather Station
* 1x AC wall adapter
* 2x receipt printer paper roll (Length: 58 mm long, Diameter: 39 mm)

Figure 3 - Weather Station Top View

## 4.2 Getting Started

The following steps detail the process required to set up the device to be ready for printing.

### 4.2.1 Installing the Paper

1. Flip latch located on top of Weather Station to open the paper compartment.
2. Place the paper roll into the compartment as shown in Figure 3.
3. Firmly close the lid until it clicks. At least 1 cm of paper should extrude outside of the closed printer. This ensures the paper will not feed back into the compartment and jam the printer.



Figure 4 - Correct Paper Orientation

### 4.2.2 Powering the Weather Station

1. Place the Weather Station on a flat, sturdy surface near an outlet. Orient the unit printer-side up.
2. Connect AC adaptor barrel plug into the back of the Weather Station.
3. Plug AC adaptor into standard 120V AC wall socket.

|  |
| --- |
| **CAUTION: THIS PRODUCT IS MEANT FOR INDOOR USE ONLY. KEEP AWAY FROM AREAS OF EXTREME HEAT AND COLD. DO NOT PLACE NEAR SOURCES OF WATER.** |

## 4.3 Operation

To test and operate the device, follow the steps below.

### 4.3.1 Testing Paper Feed

1. Flip Switch into the “ON” position to power unit.
2. Push the small black button located on the printer case to feed the paper out by one line. If it feeds out smoothly, skip step 3.
3. If the paper jams or does not feed out properly, follow the paper installation steps again. Repeat step 2.

### 4.3.2 Printing Current Weather Data

1. Push the large button below the printer case to request a print-out of the current weather. It can take several seconds to connect to the internet before printing.
2. Remove printed paper by tearing down from side to side.

|  |
| --- |
| **Note: Status LED will constantly blink if the printer is low on, or out of paper.** |

## 4.4 Troubleshooting

Should any problems occur when operating the device, refer to these troubleshooting tips.

|  |  |
| --- | --- |
| **Problem** | **Troubleshooting Tips** |
| Paper will not print, LED flashing | * Flashing LED indicates the printer is low on, or out of paper |
| Paper will not print, LED not flashing | * Check that your wireless network is up * Check to see if the paper is jammed * Ensure you are plugged into a working outlet |
| Paper jammed | * See Page 2 and reinstall paper roll |

# 5.0 Theory of Operation

The basis of this project is communication with the internet through Wi-Fi to retrieve weather data, and print the data out on a receipt printer. The process is to connect to a wireless network, then connect to a website with easy to retrieve weather data, next to request the weather from Saskatoon, and finally display that to the user. To make use of this process, understanding of Wi-Fi, Application Program Interface, Hypertext Transfer Protocol, and receipt printer command sets were needed. Understanding of these topics was necessary to choose the components for my project.

## 5.1 Wireless Internet

When considering ways for receiving and transmitting data, Wi-Fi is quickly becoming a better choice due to it becoming cheaper and more accessible. If a device is in an area with wireless internet, there is no need for transmitting data long distances, physically connecting to the internet, or manually retrieving data. Data can be stored and retrieved from anywhere at any time. This is often referred to as the Internet of Things (IoT). “At its core, IoT is simple: it’s about connecting devices over the internet, letting them talk to us, applications, and each other. The popular, if silly, example is the smart fridge: what if your fridge could tell you it was out of milk, texting you if its internal cameras saw there was none left, or that the carton was past its use-by date?” (Kobie, 2015).

For this project, only 1-way communication is needed, as the device only retrieves data from a website. It does not communicate back any data.

## 5.2 Application Program Interface (API)

“An application-programming interface (API) is a set of programming instructions and standards for accessing a Web-based software application or Web tool. A software company releases its API to the public so that other software developers can design products that are powered by its service” (Roos, 2007). There are APIs available for things like Google Calendar, Twitter, Facebook, and of course weather. There are many standalone websites devoted entirely to being an API service. These sites make it easy to get data of interest without having to search through a huge amount of unwanted website data. This is the type of site preferred to get the current weather. From their service one can specify which things like city, forecast, and number of days desired.

Some weather API services include openweathermap.org, wunderground.com, developer.forecast.io, and worldweatheronline.com. I am currently using Open Weather Map, as it is a free, popular, and reliable service.

### 5.2.1 World Weather Online

I began this project using the service from World Weather Online, and built the prototype around it. I was drawn to the service because it was simple, well documented, and allowed for very good data selectivity, and the ability to receive data in comma separated values (CSV) format. By early March, the prototype would not return weather data successfully, and I found that the URL I was using was returning an error.

According to their official Facebook page, the site had been receiving DDoS (distributed denial-of-service) attacks to disrupt the service. For 2-3 weeks users were reporting errors with the service. After about a week, I was able to view the API URL from my computer again, but making the actual API call from the Wi-Fi module only worked sporadically. I decided that I would need to find a new service to use as World Weather Online was no longer reliable.

### 5.2.2 Open Weather Map

Open Weather Map is a very popular service that even tops the google search results for “weather API”. The service was not as flexible as World Weather Online, but the documentation was good, and I expected it would be one of the most reliable services available. The main downside to using this service was that CSV format was not available, so I had to change project to work with JSON data.

## 5.3 HTTP Requests

HTTP stands for Hypertext Transfer Protocol, and it is the language of websites and the transfer of information. Once connected to a web server, HTTP requests can be made to either get or post data to a server. There are several of these requests, but the most basic of these are GET and POST. A GET request is used to retrieve certain information from a server. A POST request is used to send data to a server. This project only needs to make use of the GET request, as it is only needed to retrieve weather data from a server. GET requests follow a certain syntax, and has several optional fields to narrow in on results (HTTP Tutorial).

For the needs of this project, it is only necessary to need the server and the URL containing the data of interest. Shortly after making the request, the server will respond with the data located at the URL. From there data can be parsed to isolate values such as temperature and wind speed.

## 5.4 Receipt Printer Commands

Most available receipt printers on the market use to use a command set called ESC/POS, which stands for Epson Standard Code/ Point of Sale. The printer I chose makes use of this command set to format the information I display to the user. Some commands available include: justification, bold, double size, font type, and letter orientation (Kashino Technology Limited).

# 6.0 Component Selection

This section explains the rationale for why I selected the different components that make up my project. Components that best meet the requirements of the project were chosen after weighing their advantages and disadvantages. Each component selection is based on the Theories of Operation discussed above.

## 6.1 Microcontroller

In our classes, we studied Microchips PIC18F series of microcontrollers, so it makes sense to use a microcontroller from this family for my project. I used Microchip’s search function to narrow down the search based on my needs. I need a microcontroller with a 3.3 Volt operating voltage as communication with my chosen Wi-Fi module needs to be done with 3.3 Volts. Two onboard USART (Universal Synchronous Asynchronous Receiver Transmitter) modules are needed for this project, one for the printer and one for the Wi-Fi module.

The microcontroller that best fits the needs of this project is the PIC18LF25K22. It fits the specifications listed above, and has mid-range program memory and RAM (Random Access Memory). Information regarding this microcontroller was found from Microchips website.

## 6.2 Wi-Fi Module

A Wi-Fi module is needed to wirelessly connect to the internet and retrieve data. Some of the ideal properties of such a module would include: USART communication, simple to use, reasonable cost, and good documentation.

The ESP8266 is a very popular Wi-Fi module among electronic hobbyists, both for its low cost and simple command set. Because of its popularity, it has a lot of documentation and even a small community built around using it. I have prior experience with this device, and already had a basic understanding of using it.

**ESP8266 Specifications:**

* 3.3 Volt operating Voltage
* USART communication
* 215 mA peak current draw
* no external antenna needed
* < $5

The XBee is another fairly popular device for connecting to the internet, so it has good documentation. The command set isn’t too complicated, but more so than the ESP8266. High cost and high power consumption are negatives.

**XBee 56B Specifications:**

* 3.3 Volt operating Voltage
* USART communication
* 309 mA peak current draw
* antenna optional
* $35

This device has the lowest power consumption, and average cost compared to competitors, but the main downside is its complexity. It is a very powerful device with a large amount of functionality and features. As my project only requires basic Wi-Fi communication, this amount of functionality is more than is needed, and learning it would take a much longer time than the other two devices. That is reason enough for me to go with another option.

**Microchip RN1723 Specifications:**

* 3.3 Volt operating Voltage
* USART communication
* 120 mA peak current draw
* antenna optional
* $16

In the end I decided to go with the ESP8266 module, as it best fits my requirements, and I already have some experience with it. The ESP8266 family contains over a dozen different types. I will be using the ESP-12E board. It is one of the most recent boards, is a surface mount device, and does not need an external antenna.

## 6.3 Thermal Receipt Printer

Finding an appropriate receipt printer for this project was a bit tricky. The ideal printer would be small, sleek, reasonably priced, come with USART communication, and with good documentation to make it easy to implement. Most printers on the market are actually meant for point-of-sale purposes, so they are too expensive ($100-$400), have an awkward shape that would make it difficult to mount in a case, and use a USB data connection rather than USART.

I was familiar with the CSN-A2 Thermal Receipt Printer before choosing this project, and it was part of my inspiration for doing it. This printer is popular in hobbyist projects online, which made it an attractive option for my needs. The specifications of this printer are listed below.

**Specifications:**

* small size, rectangular shape
* $50-$60
* USART communication
* 5 Volt -12 Volt operation
* paper sensor
* sleep mode for power saving
* maximum 1.5 Amp current draw

The CSN-A2 printer fits all of my requirements. After looking at other products, I could not find any instances where another printer had any benefits over the CSN-A2.

## 6.4 Power Supply

I need a power supply of 5 Volts as that is the minimum voltage required to power the printer. I can use a voltage regulator for the 3.3 Volt devices. A power supply with peak current output of 2 Amps would be suitable for my needs. A wall-mount power supply that fit these specifications was the WSU050-2000, with a cost of $13.04 per unit. I found this transformer using the search function of the online store Digikey.

## 6.4 3.3V Voltage Regulator

Since my power supply is required to be 5 Volts, I need a voltage regulator to step down the voltage from 5 to 3.3 to power the microcontroller, LCD, and Wi-Fi module. I need a surface mount voltage regulator capable of supplying enough current for the microcontroller, LCD, and Wi-Fi module. Using Digikey’s search, I found the LM1117IDTX-3.3/NOPB voltage regulator from Texas Instruments. This regulator can supply up to 800mA, much more than I expect to be using. It has a cost of $1.96.

## 6.5 LCD

I will be using an LCD to display some information to the user, such as whether or not the device is connected to the internet. In our labs we used the NHD-0216HZ-FSW-FBW-33V3C LCD, and I have learned how to use its command set. This LCD runs at 3.3 Volts, and has a display of 16x2 characters. There are many devices like this one, as this 16x2 design is very common. Because of that, I will use the one I am familiar with.

## 6.6 Momentary Push Button

When they user wants to get the current weather, all they need to do is push a button. The kind of push button I was looking for was a momentary normally-open push button switch. There are many types of switches, and the specifications are not very important, so I based my decision almost entirely on aesthetics. I wanted a sleek illuminated button, so I decided upon a metal unit with an illuminated LED circle around the push button. The manufacturer number for the button I purchased is ZJ44803. The LED is rated for up to 12 Volts, and the switch contacts are rated for 5 Amps and 250 Volts AC.

## 6.7 Notification LED

The notification LED was another component where the main consideration was aesthetics and simplicity. The only requirements are that it operates at 3.3 Volts and is made to be mounted in a case. I decided upon the 5586003007F panel indicator light from Dialight. It is a simple blue LED with diameter of 3 mm.

## 6.8 Power Switch

The power switch is not a crucial component, so I decided to go with a black simple rocker switch. From Digikey I found a simple two terminal circular rocker switch, RR511D1121 manufactured by E-Switch.

## 6.9 Case

The case must be large enough to house all components. The printer and PCB take up the majority of the space for my project, so considerations were mainly based around those. I wanted a simple rectangular shape for the case, and my main decision was whether to choose a clear case, or a solid colour case.

After searching through cases on Digikey, I ended up choosing a rectangular case made of translucent blue polycarbonate, so that the user will be able to see everything inside the case. The dimensions available that best fit my requirements are shown below, where A, B, and C are 6.73”, 4.67”, and 3.15” respectively. These dimensions belong to product BT-2727 by Bud Industries. The image below was taken from the BT-2727 product page on Digikey.

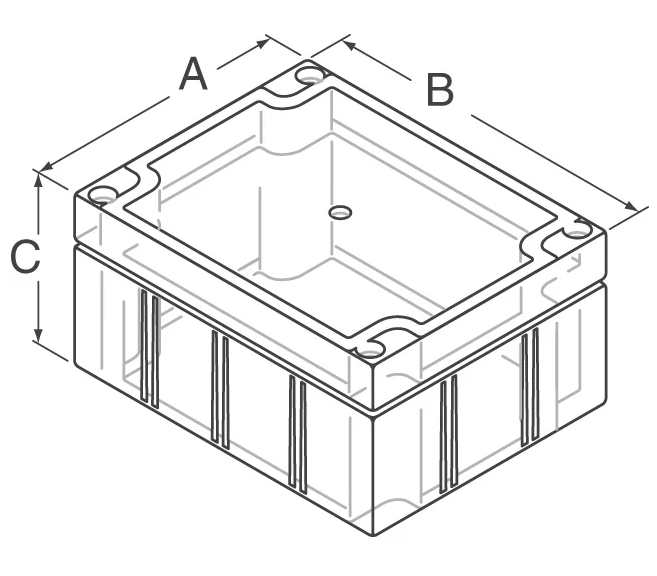


Figure 5 - BT-2727 Enclosure

# 7.0 Conclusion

The completed project was able to meet the requirements set by the project managers. The device gathered weather information and the word-of-the day through Wi-Fi, and parsed through that information to display data of interest to the user by receipt paper. The LCD would give the user feedback as to whether a connection to the Wi-Fi was successful or not, and if the printer still contained paper and was ready to print. One difficulty the device has is in a situation of very low Wi-Fi signal strength. Though the situation was rare, if it were to lose connection to the Wi-Fi in the middle of receiving data or in between API requests, the device would freeze up.

One limitation to the functionality of the design is in handling large amounts of information, as some API services are capable of sending thousands of characters of data through to the microcontroller. Because such a large amount of data cannot be stored in most microcontrollers without external storage, I had to use parsing techniques in between single character transmissions rather than saving all of the data at once before parsing. The former method of parsing can be more tedious and error-prone than the latter, but it was still able to reliably do the job for my needs.

The work I have done for this design stands as a good basis for working with information transfer with Wi-Fi and being able to use information gathered from an online source to interface with a wireless device.

# 8.0 Recommendations

There are several ways this design could be improved, but the main ways would be to perform better in low signal strength areas, and being more efficient when parsing through very large amounts in incoming data.

The Wi-Fi module used in this project was a version with an on-PCB antenna, which performs poorly in low Wi-Fi signal areas. An easy way to improve this would be to use a module version with the option for an external antenna. Another improvement to the software would be better handling of lost signals mid-transmission. In its current state this would cause the program to freeze, but with some more advanced handling techniques, that could be avoided.

The other recommendation I would make is that if it were necessary to parse through very large amounts of data, it would be an option to do the data requisition by another server, parsing it there with more reliable techniques, and then sending only the data of interest to the Wi-Fi module. Though it would be more expensive and require more knowledge of HTTP, doing the heavy parsing server-side would free up processing for the microcontroller to do other things, and would make maintenance on the project much easier. If for example an API service shut down, I could find a new one and make the necessary changes server-side without having to reprogram the device. If one of the API services I used in this design were to close down, the device would be rendered useless, and would require updating and reprogramming.

# References

Espressif Systems IOT Team. (2015, July 3). *ESP8266 AT Instruction Set*. Retrieved from www.sparkfun.com: https://cdn.sparkfun.com/assets/learn\_tutorials/4/0/3/4A-ESP8266\_\_AT\_Instruction\_Set\_\_EN\_v0.30.pdf

*HTTP Tutorial*. (n.d.). Retrieved from TutorialsPoint.com: http://www.tutorialspoint.com/http/index.htm

*Introducing JSON*. (n.d.). Retrieved from JSON.org: http://www.json.org/

Kashino Technology Limited. (n.d.). *CSN-A2 User manual.* Retrieved January 21, 2016, from Adafruit: https://www.adafruit.com/datasheets/CSN-A2%20User%20Manual.pdf

Kobie, N. (2015, May 6). *What is the internet of things?* Retrieved from TheGuardian.com: http://www.theguardian.com/technology/2015/may/06/what-is-the-internet-of-things-google

Roos, D. (2007, November 23). *How to Leverage an API for Conferencing*. Retrieved from HowStuffWorks.com: http://money.howstuffworks.com/business-communications/how-to-leverage-an-api-for-conferencing1.htm

w3school. (n.d.). *JSON Tutorial*. Retrieved from w3school: http://www.w3schools.com/json/

# Glossary of Abbreviations

API: Application Program Interface

HTTP: Hypertext Terminal Protocol

IoT: Internet of Things

CSV: Comma Separated Values

ESC/POS: Epson Standard Code/ Point of Sale

RAM: Random Access Memory

USART: Universal Synchronous Asynchronous Receiver Transmitter

# Appendices

## A. Block Diagram

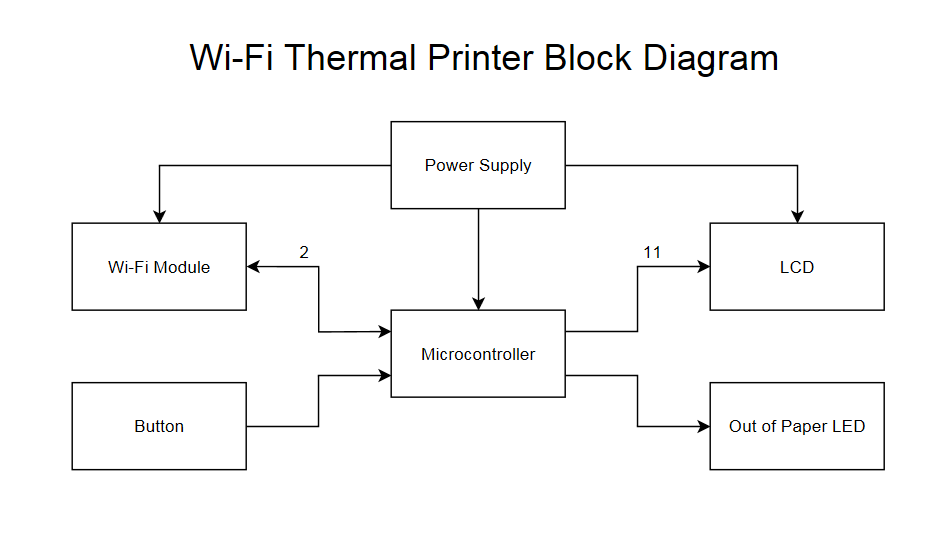


Figure 6 - Block Diagram

## B. Circuit Schematic

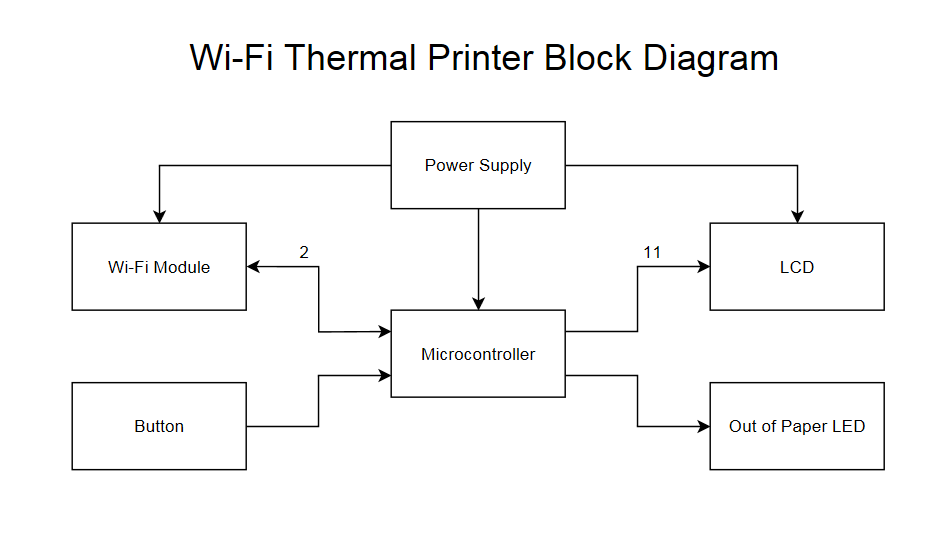


Figure 7 - Circuit Diagram

## C:\Users\User\Desktop\board bot.pngC:\Users\User\Desktop\board top.pngC. Printed Circuit Board – Top and Bottom

Figure 9 – PCB Bottom

Figure 8 – PCB Top

## D. Bill of Materials

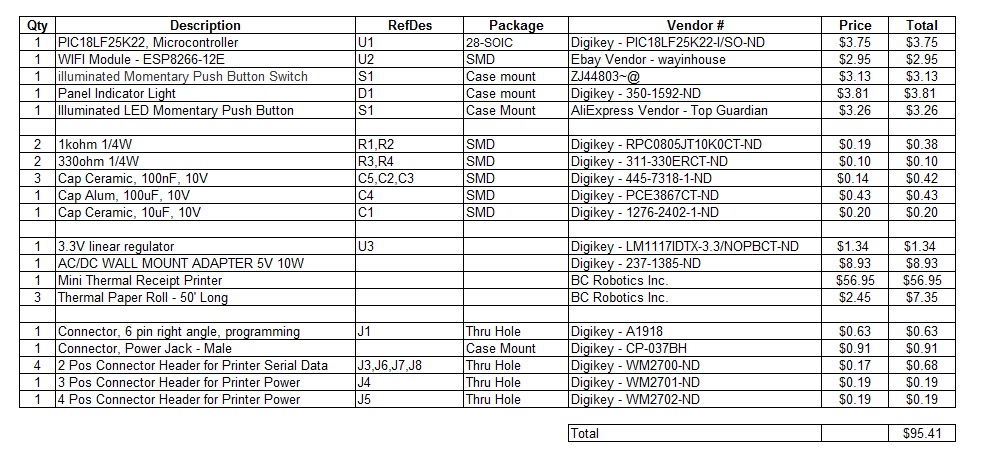


Figure 10 - Bill of Materials

## E. Gantt Chart

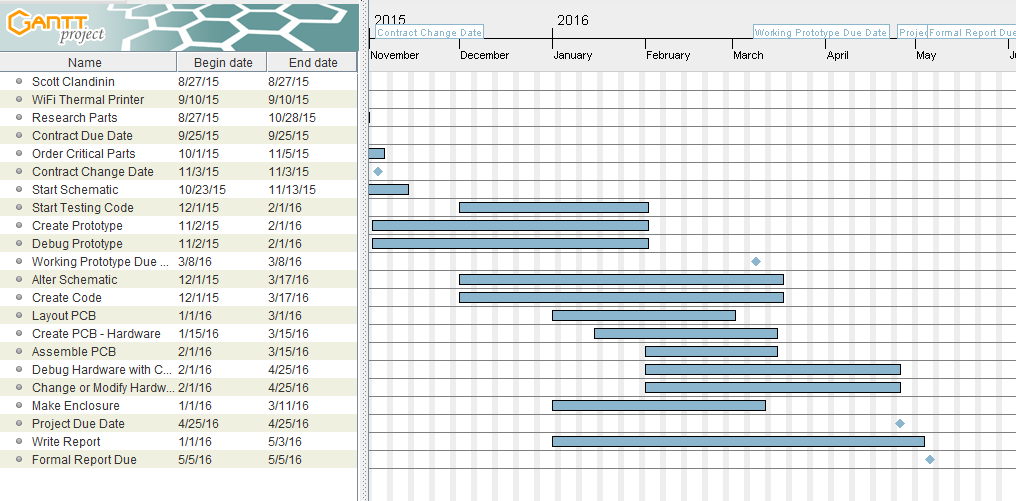


Figure 11 - Gantt Chart

## F. Software

### F.1 Main Code

/\* Scott Clandinin

\* Tech Thesis Project V2

\* Weather data from Open Weather Map:

\* http://api.openweathermap.org/data/2.5/find?q=Saskatoon&units=metric&appid=1324024f8cb94b1444352b7f894fb724

\* Word of the day from Wordnik

\* http://api.wordnik.com/v4/words.json/wordOfTheDay?api\_key=a2a73e7b926c924fad7001ca3111acd55af2ffabf50eb4ae5

\*

\* CSN-A2 thermal receipt printer

\* ESP8266-12 Wi-Fi module

\*

\*/

#include <p18lf25k22.h>

#include "defines.h"

#include "prototypes.h"

#include "delays.h"

#include <stdio.h>

#include <stdlib.h>

//global variables

unsigned char data[255];

unsigned char word[15] = " ";

unsigned char partOfSpeech[10] = " ";

char buff;

//variables to send to LCD

char weather1[14] = "Scott's Wi-Fi";

char weather2[16] = "Weather Station";

char printing[13] = "Now Printing";

char press[15] = "Press to Print";

char fail1[11] = "Connection";

char fail2[7] = "Failed";

char try[10] = "Try Again";

char connected[18] = "Wi-Fi Connected ";

char paperOut[13] = "Out of Paper";

char paperIn[15] = "Paper Replaced";

char connect[14] = "Connecting...";

void main ()

{

int status;

//initializations

port\_init();

lcd\_init();

serial\_init();

// fprintf(\_H\_USART, "AT+UART=19200,8,1,0,0\r\n"); //change default baud rate

// fprintf(\_H\_USART, "AT+CWMODE=1\r\n"); //AP + station mode

fprintf(\_H\_USER, "%c%c%c%c%c", 0x1B, 0x37, 0x07, 0x64, 0x02); //1ms heat time, 20us heat interval

delay ();

sleep\_all();

paperCheck(); //check paper status

send\_command(0x01); //clear display

send\_string(press, 14);

while (1)

{

paperCheck(); //check paper status

delay ();

if (!button) //push button

{

status = wifi\_connect(); //attempt to connect

if (status == 1) //if connected, get the weather and word

{

send\_command(0x01); //clear display

send\_string(connected, 17);

get\_weather(); //get and print weather

get\_wordoftheday(); //get and print word of the day

delaywifi2();

}

else //if connection failed, notify user and sleep

{

send\_command(0x01); //clear display

send\_string(fail1, 10);

send\_command(0xC0); //second line

send\_string(fail2, 6);

delaywifi2();

send\_command(0x01); //clear display

send\_string(try, 9);

delaywifi2();

}

sleep\_all();

send\_command(0x01); //clear display

send\_string(press, 14);

}

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* get\_weather

void get\_weather()

{

char temp[5] = " ";

char wind[5] = " ";

weather\_start(); //connect to server

RCSTA1bits.CREN = 1; //enable receive

wifi\_skip\_data(); //skip headers

weather\_read(); //store data

RCSTA1bits.CREN = 0; //disable receive

fprintf(\_H\_USER, "%c", 0xFF); //wake

fprintf(\_H\_USER, "%c%c", 0x1B, 0x40); //clear print buffer

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x61, 0x01); //justify center

fprintf(\_H\_USER, "-------------------------------\r\n");

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x30); //double size and width

fprintf(\_H\_USER, "Current Weather\r\n");

fprintf(\_H\_USER, "in Saskatoon\r\n\r\n");

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x00); //normal size

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x61, 0x01); //justify center

fprintf(\_H\_USER, "Temperature:\r\n");

get\_temp(&temp); //get the temperature

send\_string(temp, 2);

fprintf(\_H\_USER, "%s", temp);

fprintf(\_H\_USER, " %cC\r\n\r\n", 0xF8);

send\_data(0xDF);

send\_data('C');

fprintf(\_H\_USER, "Wind Speed:\r\n");

get\_wind(&wind); //get the wind speed

fprintf(\_H\_USER, "%s", wind);

fprintf(\_H\_USER, " m/s\r\n\r\n");

fprintf(\_H\_USER, "via openweathermap.org\r\n");

fprintf(\_H\_USER, "--------------------------------\r\n");

fprintf(\_H\_USER, "%c%c", 0x1B, 0x40); //clear print buffer

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* get\_wordoftheday

void get\_wordoftheday()

{

unsigned char definition[120] = " ";

word\_start(); //connect to server

RCSTA1bits.CREN = 1; //enable receive

wifi\_skip\_data(); //skip headers

word\_read(&word); //find word of the day

definition\_read(&definition); //get the definition

partOfSpeech\_read(&partOfSpeech); //get the part of speech of the word

RCSTA1bits.CREN = 0; //disable receive

fprintf(\_H\_USER, "%c", 0xFF); //wake

fprintf(\_H\_USER, "%c%c", 0x1B, 0x40); //clear print buffer

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x30); //double size and width

fprintf(\_H\_USER, "Word of the Day\r\n\r\n");

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x00); //normal size

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x61, 0x00); //justify left

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x08); //bold

fprintf(\_H\_USER, "%s - %s\r\n\r\n", word, partOfSpeech);

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x00); //not bold

fprintf(\_H\_USER, "%s\r\n\r\n", definition);

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x61, 0x01); //justify center

fprintf(\_H\_USER, "via wordnik.com\r\n");

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x61, 0x00); //justify left

fprintf(\_H\_USER, "-------------------------------\r\n");

fprintf(\_H\_USER, "By Scott Clandinin\r\n\r\n\r\n\r\n\r\n");

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* sleep\_all

void sleep\_all()

{

send\_command(0x01); //clear display

send\_string(weather1, 13);

send\_command(0xC0); //second line

send\_string(weather2, 15);

delay();

INTCONbits.INT0IF = 0; //ext interrupt flag low

fprintf(\_H\_USART, "AT+GSLP=5000\r\n"); //send ESP to sleep

delay();

reset = 0; //bring line low to reset

Sleep(); //send micro to sleep

Nop();

//wake

delay();

reset = 1; //wake ESP

}

//misc

#include "port\_init.c"

#include "delays.c"

//LCD

#include "lcd.c"

//USART

#include "wifi\_serial.c"

#include "\_user\_putc.c"

//Printer

#include "printer\_serial.c"

//Parsing

#include "parsing.c"

### F.2 Port Initialization

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* port\_init

void port\_init()

{

OSCCON = 0x70; //16MHz

ANSELA = 0;

ANSELB = 0;

ANSELC = 0;

PORTA = 0;

PORTB = 0;

PORTC = 0b00100000;

TRISA = 0;

TRISB = 0b10000001;

TRISC = 0b10000000;

INTCONbits.INT0IE = 1; //enable external interrupts

}

### F.3 Defines

#define button PORTBbits.RB0

#define reset PORTCbits.RC5

#define LED PORTBbits.RB1

#define DC PORTCbits.RC2

#define RW PORTCbits.RC1

#define EN PORTCbits.RC0

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Configuration Bits Set in code

//You find the definitions in MPlab only

//Help -> PIC18 Config Settings -> PIC18F25K22

#pragma config FOSC = INTIO67 //Internal oscillator block, CLKOUT function on RA6, port function on RA7

#pragma config PLLCFG = OFF //Oscillator multiplied by 4

#pragma config PRICLKEN = ON //Primary clock is always enabled

#pragma config IESO = OFF //Oscillator Switchover mode disabled

#pragma config FCMEN = OFF //Fail-Safe Clock Monitor disabled

#pragma config PWRTEN = OFF //Power up timer disabled

#pragma config BOREN = OFF //Brown-out Reset disabled in hardware only

#pragma config BORV = 190 //VBOR set to 1.90 V nominal

#pragma config WDTEN = OFF //Watch dog timer is always disabled. SWDTEN has no effect.

#pragma config CCP2MX = PORTB3 //CCP2 input/output is multiplexed with RB3

#pragma config PBADEN = OFF //PORTB<5:0> pins are configured as digital I/O on Reset

#pragma config HFOFST = OFF //HFINTOSC output and ready status are delayed by the oscillator stable status

#pragma config MCLRE = EXTMCLR //MCLR pin enabled, RE3 input pin disabled

#pragma config STVREN = OFF //Stack full/underflow will cause Reset

#pragma config LVP = OFF //NO Low Voltage Program, Single-Supply ICSP disabled

#pragma config XINST = OFF //Instruction set extension and Indexed Addressing mode disabled (Legacy mode)

#pragma config DEBUG = ON //Background debugging on RB7 & RB6

//

#pragma config CP0 = OFF //BLOCK 0 NOT CODE PROTECTED

#pragma config CP1 = OFF //BLOCK 1 NOT CODE PROTECTED

#pragma config CP2 = OFF //BLOCK 2 NOT CODE PROTECTED

#pragma config CP3 = OFF //BLOCK 3 NOT CODE PROTECTED

#pragma config CPB = OFF //BOOT BLOCK NOT CODE PROTECTED

#pragma config CPD = OFF //DATA EEPROM NOT CODE PROTECTED

#pragma config WRT0 = OFF //BLOCK 0 NOT WRITE PROTECTED

#pragma config WRT1 = OFF //BLOCK 1 NOT WRITE PROTECTED

#pragma config WRT2 = OFF //BLOCK 2 NOT WRITE PROTECTED

#pragma config WRT3 = OFF //BLOCK 3 NOT WRITE PROTECTED

#pragma config WRTB = OFF //BOOT BLOCK NOT WRITE PROTECTED

#pragma config WRTC = OFF //CONFIGURATION REGISTER NOT WRITE PROTECTED

#pragma config WRTD = OFF //DATA EEPROM NOT WRITE PROTECTED

#pragma config EBTR0 = OFF //BLOCK 0 NOT PROTECTED FROM TABLE READS FROM OTHER BLOCKS

#pragma config EBTR1 = OFF //BLOCK 1 NOT PROTECTED FROM TABLE READS FROM OTHER BLOCKS

#pragma config EBTR2 = OFF //BLOCK 2 NOT PROTECTED FROM TABLE READS FROM OTHER BLOCKS

#pragma config EBTR3 = OFF //BLOCK 3 NOT PROTECTED FROM TABLE READS FROM OTHER BLOCKS

#pragma config EBTRB = OFF //BOOT BLOCK NOT PROTECTED FROM TABLE READS FROM OTHER BLOCKS

### F.4 Function Prototypes

void port\_init();

void sleep\_all();

void get\_weather();

void get\_wordoftheday();

//lcd functions

void lcd\_init();

void send\_command(char command);

void send\_data(char data);

void send\_string(char string[],int length);

char reverse\_bits(char value);

//wifi functions

void serial\_init();

int wifi\_connect();

void wifi\_acknowledge();

void wifi\_skip\_data();

void weather\_start();

void weather\_read();

void word\_start();

void word\_read(char word[]);

void definition\_read(char definition[]);

void partOfSpeech\_read(char partOfSpeech[]);

//parsing code

int find\_temp();

void get\_temp(char temp[]);

int find\_wind();

void get\_wind(char wind[]);

//printer functions

int \_user\_putc (char c);

void printerTest();

void paperCheck();

char printerRead();

//delay functions

void delay();

void delaylcd();

void delaywifi();

void delaywifi2();

### F.5 Wi-Fi

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* serial\_init

void serial\_init()

{

//USART 1 - wifi module

TXSTA1bits.SYNC = 0;

TXSTA1bits.BRGH = 1;

TXSTA1bits.TX9 = 0;

TXSTA1bits.TXEN = 1;

RCSTA1bits.SPEN = 1;

RCSTA1bits.RX9 = 0;

//default baud rate

// SPBRG1 = 0x08; //115200 baud rate

// SPBRGH1 = 0x00; //115200 baud rate

//deisred baud rate

SPBRG1 = 0x33; //19200 baud rate

SPBRGH1 = 0x00; //19200 baud rate

//ESP boot mode baud rate

// SPBRG1 = 0x0C; //74880 baud rate

// SPBRGH1 = 0x00; //74880 baud rate

//USART 2 - Printer

TXSTA2bits.SYNC = 0;

TXSTA2bits.BRGH = 1;

TXSTA2bits.TX9 = 0;

TXSTA2bits.TXEN = 1;

RCSTA2bits.SPEN = 1;

RCSTA2bits.RX9 = 0;

SPBRG2 = 0x33; //19200 baud rate

SPBRGH2 = 0x00; //19200 baud rate

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* wifi\_acknowledge

//check that a command to the ESP was successful

void wifi\_acknowledge()

{

char buffer = 'a';

RCSTA1bits.CREN = 1;

while (buffer != 'K') //wait for connection acknowledgment

{

while (!PIR1bits.RC1IF);

buffer = RCREG1;

}

RCSTA1bits.CREN = 0;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* wifi\_connect

//connect to wireless AP

int wifi\_connect()

{

char buffer = 'a';

char buffer2 = 'a';

send\_command(0x01); //clear display

send\_string(connect, 13);

delay();

fprintf(\_H\_USART, "AT+CWJAP=\"SASKPOLYTECHPublic\",\"\"\r\n");

RCSTA1bits.CREN = 1;

while (buffer != 'K') //wait for connection acknowledgment

{

while (!PIR1bits.RC1IF);

buffer = RCREG1;

}

while (buffer2 != 'K' && buffer2 != ':') //wait for connection acknowledgment

{

while (!PIR1bits.RC1IF);

buffer2 = RCREG1;

}

RCSTA1bits.CREN = 0;

if (buffer2 == 'K')

{

return 1;

}

else

{

return 0;

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weather\_start

//begin the connection for weather retrieval

void weather\_start()

{

fprintf(\_H\_USART, "AT+CIPSTART=\"TCP\",\"api.openweathermap.org\",80\r\n");

wifi\_acknowledge(); //wait for connection acknowledgment

fprintf(\_H\_USART, "AT+CIPSEND=127\r\n");

wifi\_acknowledge(); //wait for connection acknowledgment

fprintf(\_H\_USART, "GET /data/2.5/find?q=Saskatoon&units=metric&appid=1324024f8cb94b1444352b7f894fb724 HTTP/1.0\r\nHost: api.openweathermap.org\r\n\r\n\r\n");

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* word\_start

//begin the connection for word retrieval

void word\_start()

{

fprintf(\_H\_USART, "AT+CIPSTART=\"TCP\",\"api.wordnik.com\",80\r\n");

wifi\_acknowledge(); //wait for connection acknowledgment

fprintf(\_H\_USART, "AT+CIPSEND=127\r\n");

wifi\_acknowledge(); //wait for connection acknowledgment

fprintf(\_H\_USART, "GET /v4/words.json/wordOfTheDay?api\_key=a2a73e7b926c924fad7001ca3111acd55af2ffabf50eb4ae5 HTTP/1.0\r\nHost: api.wordnik.com\r\n\r\n\r\n");

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* wifi\_skip\_data

//skips beginning data sent by server about unwanted server request details

//until a curly bracket appears, denoting the start of the JSON data

void wifi\_skip\_data()

{

char buffer1;

buffer1 = 'a';

while (buffer1 != '{') //skip to the start of the JSON data

{

while (!PIR1bits.RC1IF);

buffer1 = RCREG1;

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weather\_read

//store the weather until carriage return appears,

//meaning end of data string

void weather\_read()

{

int i;

for (i = 0; i < 255; i++)

{

while (!PIR1bits.RC1IF)

{

Nop();

}

data[i] = RCREG1;

if (data[i] == 0x0D)

{

return;

}

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* word\_read

void word\_read(char word[])

{

int i,j;

char buffer1, buffer2;

//wait for 'w' to appear

while (buffer1 != 'w')

{

while (!PIR1bits.RC1IF)

{

Nop();

}

buffer1 = RCREG1;

}

//skip next 6 characters

for (i = 0; i < 6; i++)

{

while (!PIR1bits.RC1IF)

{

Nop();

}

buffer1 = RCREG1;

}

//record word until " appears

for (j = 0; j < 10; j++)

{

while (!PIR1bits.RC1IF)

{

Nop();

}

buffer2 = RCREG1;

if (buffer2 == '"')

{

return;

}

word[j] = buffer2;

}

return;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* definition\_read

void definition\_read(char definition[])

{

int i,j;

char buffer1, buffer2;

//wait for ']' to appear

while (buffer1 != ']')

{

while (!PIR1bits.RC1IF)

{

Nop();

}

buffer1 = RCREG1;

}

//skip next 25 characters

for (i = 0; i < 25; i++)

{

while (!PIR1bits.RC1IF)

{

Nop();

}

buffer1 = RCREG1;

}

//record word until " appears

for (j = 0; j < 255; j++)

{

while (!PIR1bits.RC1IF)

{

Nop();

}

buffer2 = RCREG1;

if (buffer2 == '"')

{

return;

}

definition[j] = buffer2;

}

return;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* partOfSpeech\_read

void partOfSpeech\_read(char partOfSpeech[])

{

int i,j;

char buffer1, buffer2, buffer3;

//wait for ',' to appear twice

while (buffer1 != ',')

{

while (!PIR1bits.RC1IF)

{

Nop();

}

buffer1 = RCREG1;

}

while (buffer2 != ',')

{

while (!PIR1bits.RC1IF)

{

Nop();

}

buffer2 = RCREG1;

}

//skip next 16 characters

for (i = 0; i < 22; i++)

{

while (!PIR1bits.RC1IF)

{

Nop();

}

buffer1 = RCREG1;

}

//record word until " appears

for (j = 0; j < 30; j++)

{

while (!PIR1bits.RC1IF)

{

Nop();

}

buffer3 = RCREG1;

if (buffer3 == '"')

{

return;

}

partOfSpeech[j] = buffer3;

}

return;

}

### F.6 LCD

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* lcd\_init

void lcd\_init()

{

delaylcd();

send\_command(0x38); //function set, 8 bit, 2 line

send\_command(0x0C); //turn display on, cursor off

send\_command(0x06); //entry mode, cursor increment and no display shift

send\_command(0x01); //clear display

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* send\_command

void send\_command(char command)

{

char value;

DC = 0; //D/C set to command

RW = 1; //R/W starts high

delaylcd();

RW = 0; //set R/W for write

delaylcd();

EN = 1; //enable the chip select

delaylcd();

value = reverse\_bits(command);

PORTA = value;

delaylcd();

EN = 0; //enable the chip select

delaylcd();

RW = 1; //R/W starts high

delaylcd();

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* send\_data

void send\_data(char data)

{

char value;

DC = 1; //D/C set to data

RW = 1; //R/W starts high

delaylcd();

RW = 0; //set R/W for write

delaylcd();

EN = 1; //enable the chip select

delaylcd();

value = reverse\_bits(data);

PORTA = value;

delaylcd();

EN = 0; //enable the chip select

delaylcd();

RW = 1; //R/W starts high

delaylcd();

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* send\_string

//sends a string to the LCD

void send\_string(char string[],int length)

{

int i;

for (i = 0; i < length; i++)

{

send\_data(string[i]);

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* reverse\_bits

//reverses data and command bytes before sending them to

//the LCD. The micro data pins were laid out on the PCB in the

//opposite orientation

char reverse\_bits(char value)

{

char reversed = 0;

int i;

for(i=0; i<=7; i++)

{

reversed<<=1;

reversed|=(value&1);

value>>=1;

}

return reversed;

}

### F.7 Thermal Printer

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* printerTest

//testing the different fonts and functions of the printer

void printerTest()

{

fprintf(\_H\_USER, "%c%c%c%c%c", 0x1B, 0x37, 0x07, 0x64, 0x02); //1ms heat time, 20us heat interval

fprintf(\_H\_USER, "Lorem ipsum dolor sit amet, \r\n\r\n");

//justification

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x61, 0x02);

fprintf(\_H\_USER, "Right justified\r\n");

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x61, 0x01);

fprintf(\_H\_USER, "Center justified\r\n");

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x61, 0x00);

fprintf(\_H\_USER, "Left justified\r\n");

//font test

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x00);

fprintf(\_H\_USER, "Font A test\r\n");

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x01);

fprintf(\_H\_USER, "Font B test\r\n");

//bold test

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x08); //bold

fprintf(\_H\_USER, "Bolded text\r\n");

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x00); //not bold

fprintf(\_H\_USER, "Not bolded\r\n");

//double height and width

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x30); //double size and width

fprintf(\_H\_USER, "double height and width\r\n");

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x21, 0x00); //normal size

fprintf(\_H\_USER, "Normal size\r\n");

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* paperCheck

//check the status of the printer and loop while the paper is out

void paperCheck()

{

char status;

status = printerRead();

if (status != 0x24)

{

return;

}

else

{

send\_command(0x01); //clear display

send\_string(paperOut, 12);

}

while (status == 0x24)

{

LED = 0;

delay ();

delay ();

delay ();

LED = 1;

delay ();

delay ();

delay ();

status = printerRead();

}

LED = 0; //turn off LED

send\_command(0x01); //clear display

send\_string(paperIn, 14);

delaywifi2();

delaywifi2();

send\_command(0x01); //clear display

send\_string(press, 14);

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* printerRead

//request the status from the printer

char printerRead()

{

char status;

RCSTA2bits.CREN = 1; //receive enable

fprintf(\_H\_USER, "%c%c%c", 0x1B, 0x76, 0x04); //request paper status

while (!PIR3bits.RC2IF)

{

Nop();

}

status = RCREG2;

RCSTA2bits.CREN = 0; //receive disable

return status;

}

### F.8 Weather Parsing

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* find\_temp

int find\_temp()

{

char letter1, letter2, letter3, letter4;

int i;

for (i = 0; i < 255; i++)

{

letter1 = data[i];

letter2 = data[i+1];

letter3 = data[i+2];

letter4 = data[i+3];

if (letter1 == 't' && letter2 == 'e' && letter3 == 'm' && letter4 == 'p')

{

return i;

}

}

return 0;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* get\_temp

void get\_temp(char temp[])

{

int location;

int i = 0;

location = find\_temp();

location = location + 6;

while (data[location] != ',')

{

temp[i] = data[location];

i++;

location++;

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* find\_wind

int find\_wind()

{

char letter1, letter2, letter3, letter4, letter5;

int i;

for (i = 0; i < 255; i++)

{

letter1 = data[i];

letter2 = data[i+1];

letter3 = data[i+2];

letter4 = data[i+3];

letter5 = data[i+4];

if (letter1 == 's' && letter2 == 'p' && letter3 == 'e' && letter4 == 'e' && letter5 == 'd')

{

return i;

}

}

return 0;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* get\_wind

void get\_wind(char wind[])

{

int location;

int i = 0;

location = find\_wind();

location = location + 7;

while (data[location] != ',')

{

wind[i] = data[location];

i++;

location++;

}

}

### F.9 Delay Functions

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* delay

void delay ()

{

int i;

for (i = 0; i < 10000; i++)

{

Delay1TCY();

Delay1TCY();

Delay1TCY();

Delay1TCY();

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* delaywifi

//short wifi delay

void delaywifi ()

{

int i;

for (i = 0; i < 10; i++)

{

delay();

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* delaywifi2

//longer wifi delay

void delaywifi2 ()

{

int i;

for (i = 0; i < 20; i++)

{

delay();

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* delaylcd

//time between LCD commands

void delaylcd ()

{

int i;

for (i = 0; i < 999; i++)

{

Delay1TCY();

}

}

### F.10 USART 2 Enable

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \_user\_putc.c

/\* this default version should not do anything. it is entirely a

placeholder symbol. To keep code size at a minimum, it is declared

without a return value or parameters. The caller will still clean up

the stack frame correctly.

When using the \_H\_USER stream, the function will be implemented in

application code with the prototype:

int \_user\_putc (char c);

\*/

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \_user\_putc

//function to allow fprintf to work on Tx2

int \_user\_putc (char c)

{

while(!PIR3bits.TX2IF);

TXREG2 = c;

return c;

}