**Design Review 1: The SmartMirror**

Chris Rectenwald, Teague Kohlbeck, and Benny Richmond

EE 41440- Senior Design 2

Professor Mike Schafer

**Table of Contents**

**Introduction**

**System Requirements**

**Project Description**

Overall System

System Block Diagram

On/Off Subsystem

Display Subsystem

User Interface Subsystem

Speaker Subsystem

Internet Subsystem

Interface Requirements

**System Integration Requirements**

**User/Installation Manual**

**To-Market Design Changes**

**Conclusions**

**Appendices**

**Introduction**

**System Requirements**

The microcontroller used to control the mirror and the LCD displays will need to be powered by a 5V power supply. This will not need to be portable as the mirror will be stationary after its initial installation. A 3.3V regulator will need to be included for some components. The embedded system used in the mirror will need to be able to interface with a WiFi interface as well as multiple LCD displays. The WiFi will need to be able to work within a home, so it should be sensitive enough to pick up the signal from a home WiFi router. The system will need to be able to take user input to program the user's location, initial SSID information, and switch what data is being displayed on the auxiliary screen. There will need to be communication from a motion sensor, which will need to sense motion up to ~10 feet in front of the mirror, to determine if a person is present and the system needs to be woken from its sleep state. The displays must be able to show all relevant text and icons, and refresh quickly enough to allow for regular information updates. Overall, all the code must be compact enough to fit and run on the microcontroller we plan to use. The mirror will be around 2’ by 1.5’, so all the hardware will need to fit within this profile. The entire system will also need to be mounted on a wall, so there will need to be a frame built that can support this weight.

**5. Project Description**

**5.1 Overall System**

The overall system has been broken down into the 5 subsystems: The On/Off Subsystem, Internet Subsystem, User Interface Subsystem, Speaker Subsystem, and Display Subsystem. The connectivity of the entire system can be seen in the figure below. Although each subsystem works independently as they have been tested with either a Sparkfun ESP8266 Dev Thing, an Arduino Uno or a Sparkfun ESP32 Dev Module, we will need to consider whether or not we can run this system with seamless synchronization with multiple microcontrollers or a single microcontroller. If multiple devices does not work out, we might lean towards using a board with a similar design to the Sparkfun ESP32 Dev Module as it has 4 SPI lines and 2 I2C lines which are more than Sparkfun ESP8266 Dev Thing 2 SPI lines and 1 I2C line. The Sparkfun ESP32 Dev Module also has more SRAM than the Sparkfun ESP8266 Dev Thing, so it should be able to perform the tasks needed for full system functionality. However we need to make sure the pins for the ESP32 Dev Module can function for our system as well as have enough memory, so the SmartMirror reaches its capabilities. We will take these considerations when designing our own board for the SmartMirror.



**Figure: Overall System Flow Diagram**

**5.2 Operation and Design of On/Off Subsystem**

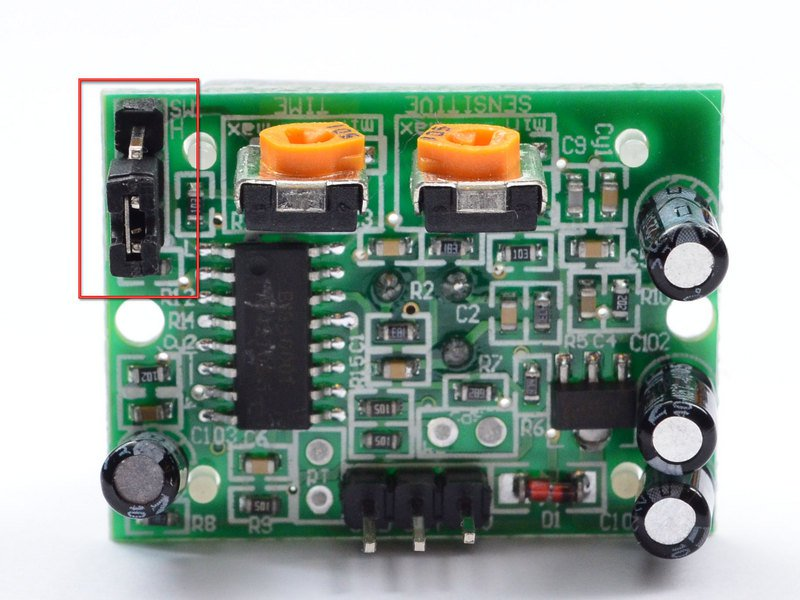


**Figure: Flow Diagram of On/Off Subsystem**

This system overall deals with determining whether or not there is a person in front of the mirror or not. As of right now the group plans to use HC-SR501 PIR Motion Detector to detect a human movement. We chose the PIR Motion Detector over other options such as a photocell because of a PIR Motion Detector is not too aesthetically displeasing, while triggering a high or a low voltage rather than current. We plan to interface the PIR Motion Detector with the Arduino Uno but could be interfaced with an ESP microcontroller. All of these choices would suffice as their are examples in the Arduino IDE to help us get through any potential roadblocks along the way.

Moving forward with this system we will need to consider how to effectively put a current-draining resistor on the mirror in order to get rid of static voltages. We also need to consider the time-delay and sensitivity adjustments as seen on top of the figure below that are available with this PIR Motion Detector as well as a repeat or single trigger option that is in the red box in order to provide the most pleasant user experience.

So far we have tested the PIR sensor in the Arduino IDE by digitally reading the voltage level from the middle prong at the bottom of figure below and outputting whether there is motion detected accordingly. After seeing that the voltage level changed based off if one’s hands had passed the capacitors on the PIR Motion Detector or not, we concluded that the PIR Motion Detector works.



**Figure: PIR Motion Detector**

**5.3 Internet Subsystem**



**Figure: Flow Diagram of Internet Subsystem**

The internet subsystem deals with being able to retrieve information such as weather, time, news or any other applicable information from the internet to be displayed on the mirror. An ESP microcontroller or similar device will be the device pulling the data over a WiFi signal and communicate via I2C/SPI to the LCDs on the mirror.

Our choice for considering both of these device is because of their ability to connect to the internet while still have the capability of communicating with I2C/SPI to the mirror. We chose to use the Arduino IDE for this subsystem, as it is flexible in ways to retrieve information with either using get http functions or parsing with javascript. For instance in order to retrieve the time, we are able to connect to a Network Time Protocol that would give us the Greenwich Mean Time, and would translate from the time accordingly based on the user’s location.

We also wanted to retrieve the local weather data, so we decided to use Wunderground’s API for any location that is based off the user’s inputted zip code. Wunderground provides a webpage of the data relevant to the user’s needs including the location, time, and weather in javascript. Arduino allows us to use the “ArduinoJson.h” library that allows one to easily parse data from character arrays using javascript. Although ideally we plan for the zip code to be read from the user’s setting on a website, to test the subsystem we hard coded the zip code to be one of South Bend’s zip codes 46617. An illustration of the software program for this subsystem, Pull\_Data.ino, can be seen in the Figure below.



**Figure: Flow Diagram of Software for Internet Subsystem**

**5.4 User Interface**

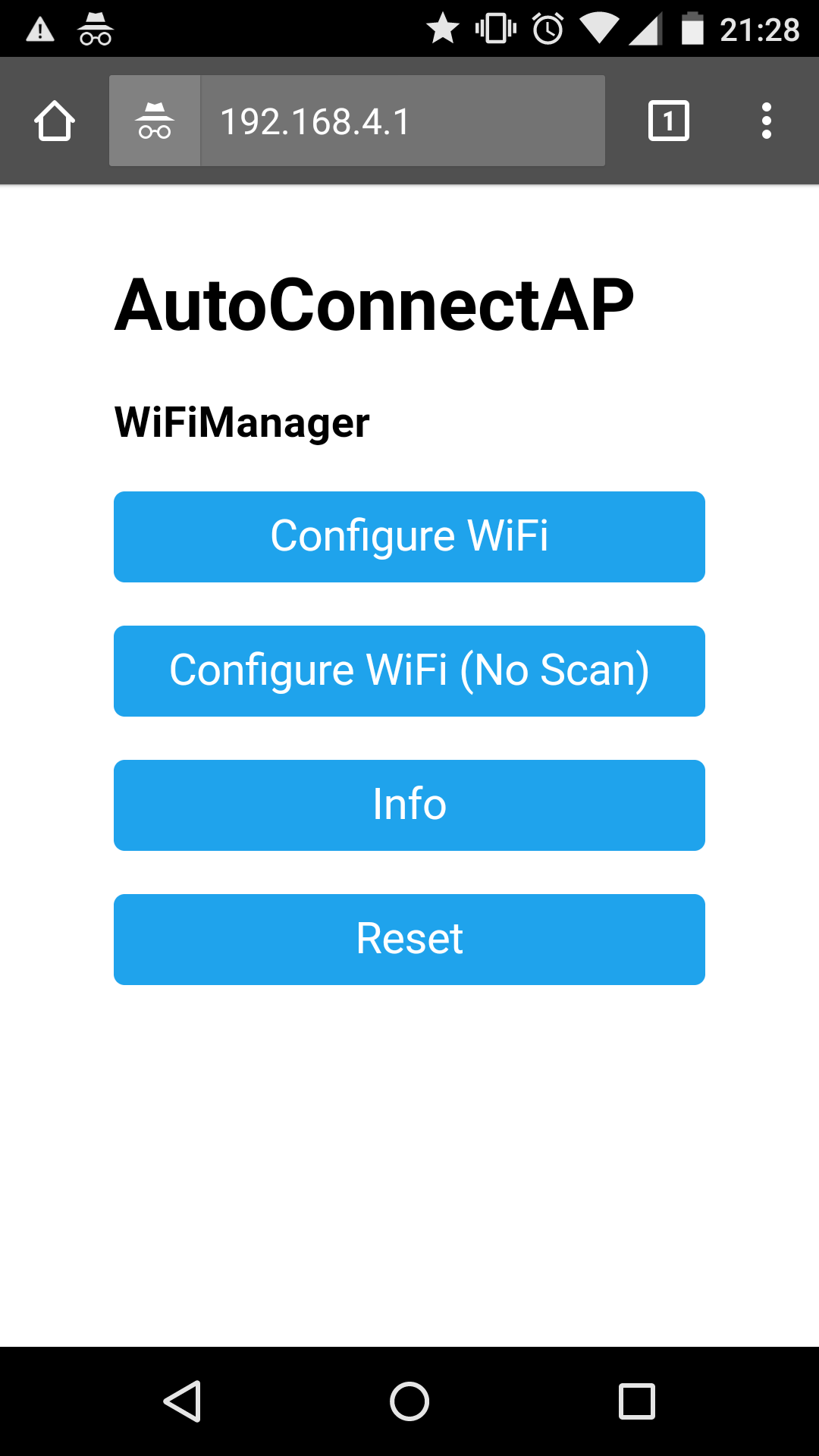
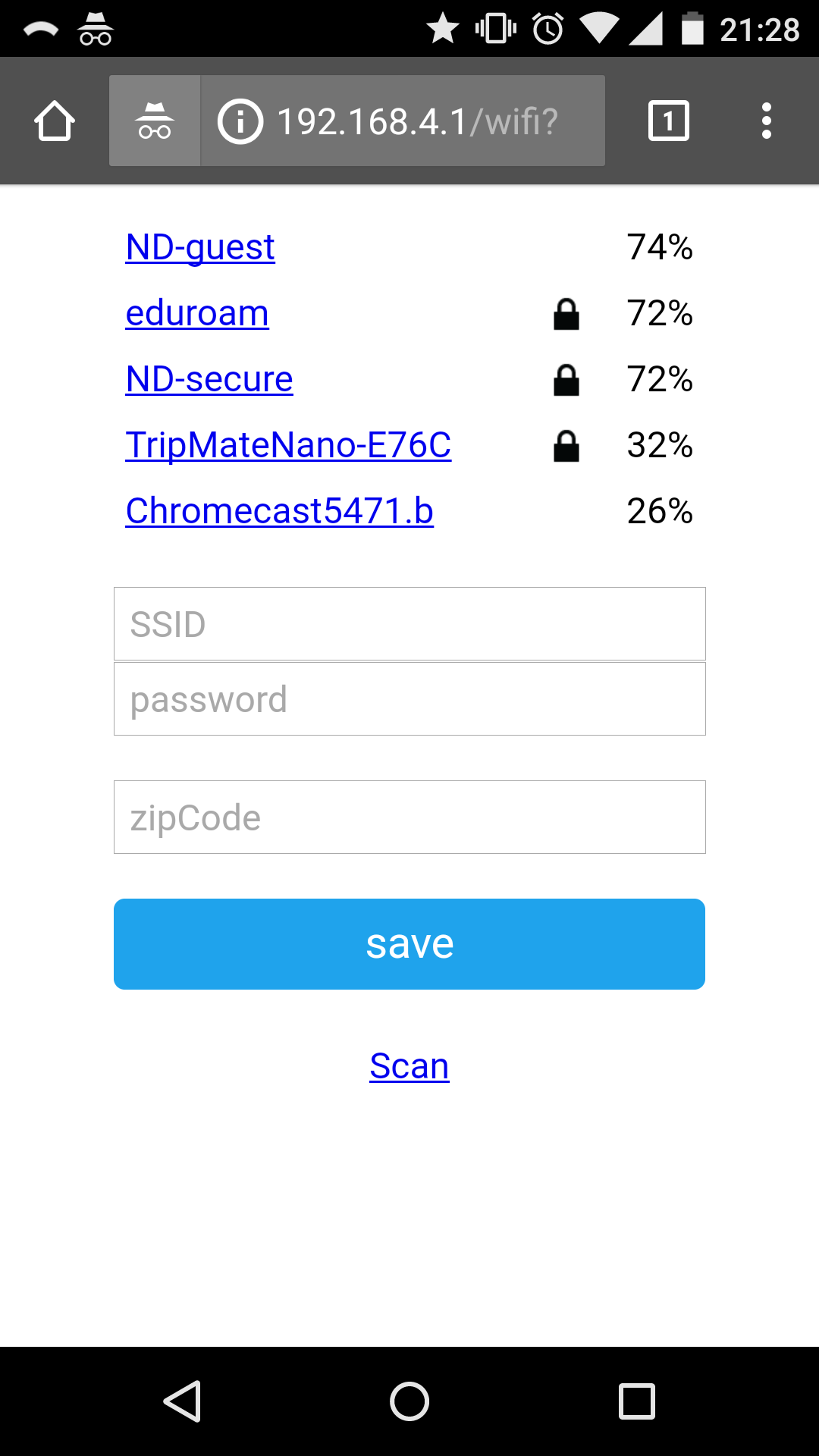


**Figure: Flow Diagram of User Interface Subsystem**

The user interface subsystem allows the user to update the settings of the mirror to connect to their home wifi and display data relevant to their geographical location. The ESP microcontroller must be configured to run as an access point server upon initial boot up as well as when the user desires to edit the settings of the mirror, as well as store the settings in the non-volatile memory of the chip.

The ESP microcontroller was chosen due to its ability to be configured as an access point as well as its ability to connect to the WiFi. The user is required to enter the ssid and password to allow the chip to connect with the WiFi as well as the user’s zip code. The zip code was chosen as a parameter as it will allow for the weather and time for the user’s location to be pulled by the internet subsystem and reduces the risk for the incorrect location to be input. The system is set to act as an access point upon powering the board on. After being powered on, it will check to see if there are valid credentials to connect to WiFi. If there are, then the chip will move on to the main function, if not, it will create an access point to input WiFi credentials. This was chosen as the mirror will be stationary and thus the information should remain the same after the initial boot up. If the location changes, the chip will not be able to connect to the network and will generate an access point to adjust the settings.

The ESP8266 was programmed using the Arduino IDE and utilizing the “ArdunoJson.h”, “DNSServer.h”, “ESP8266WebServer.h”, “WiFiManager.h”, “ESP8266.h”, and “FS.h” libraries. These were chosen as they are well documented and allow for custom html form data to be added, allowing for a custom textbox to be added to allow the zip code to be input. The system is set to take the data input as a char value. When access point mode is initiated, the user interface will be available at the chips default IP address of 192.168.4.1. Once the WiFi successfully connects, the chip will move onto the main function. Additionally, these allow for a pleasant and user friendly interface to be accessed at the default ip of the chip, as shown below.

**Figure: User Interface Web page**

To test the system, the code was uploaded to the chip and the web page was accessed from a mobile device. After the WiFi and zip code credentials were input, the board was powered off and then powered back on. Then the info was checked to ensure that the WiFi credentials had been saved.



**Figure: Flow Diagram of Software for User Interface Subsystem**

**Display Subsystem:**

This subsystem takes information retrieved by the Internet Subsystem, creates the appropriate textual outputs, and sends the desired information via SPI to each of the tft LCD screens. The settings used to organize the display of data will be based on the user’s last input to the user interface subsystem. The system will continually exchange data with the internet subsystem to update the displays regularly.

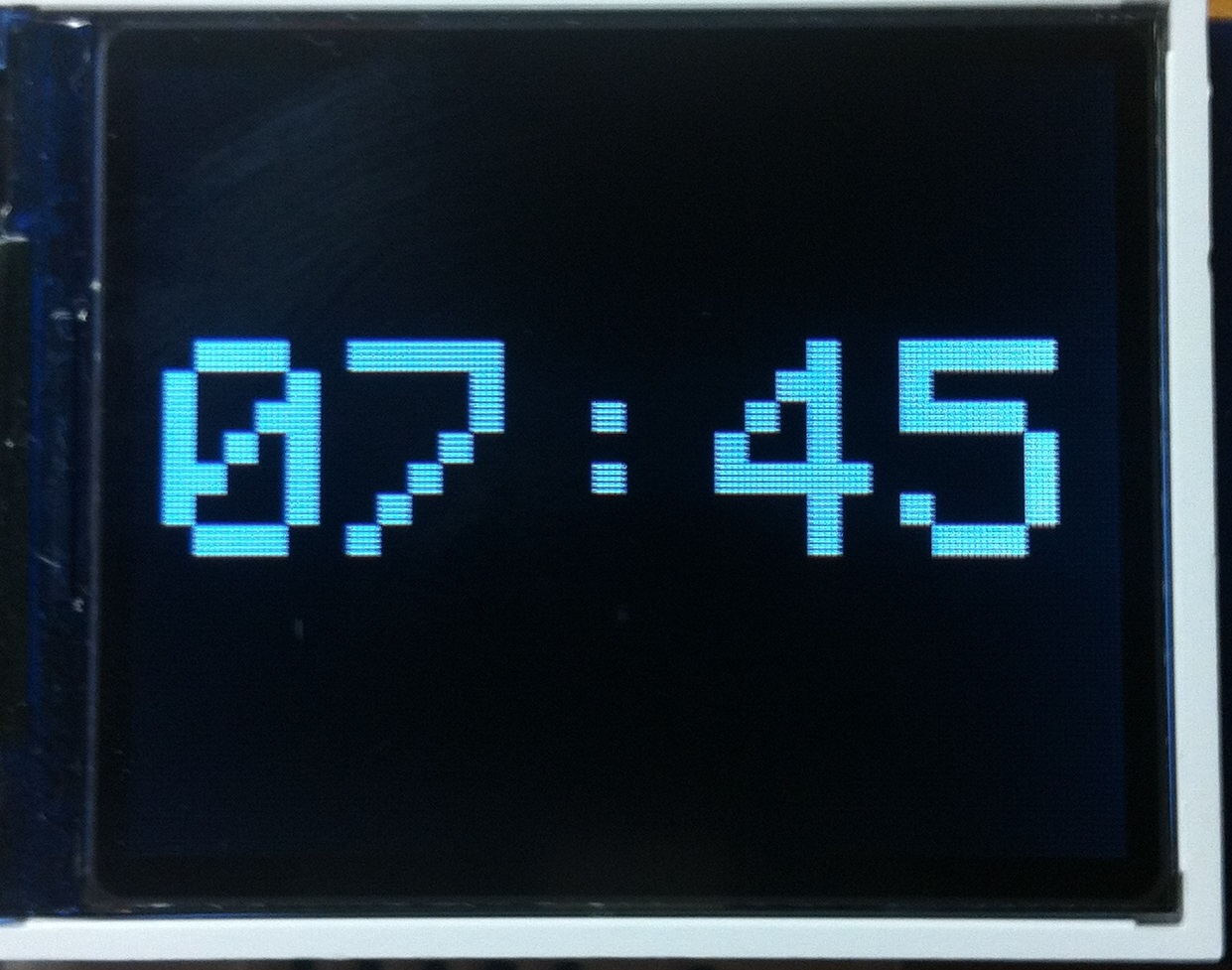
There will be four display screens in this subsystem. All are Adafruit brand tft LCD displays, chosen for their ease of use and flexibility. The first screen is a 1.8” display, and will be used to show the current time. The second screen will be another 1.8” display that will show an icon relating to the current forecast. The third screen will be a 2.8” display, and will be used to show the daily weather. The fourth screen will be another 2.8” display, and will be used to show further information. One of the 1.8” displays was purchased with a breakout board and header pins included for ease of use, but the other three displays will only have surface mount cables.

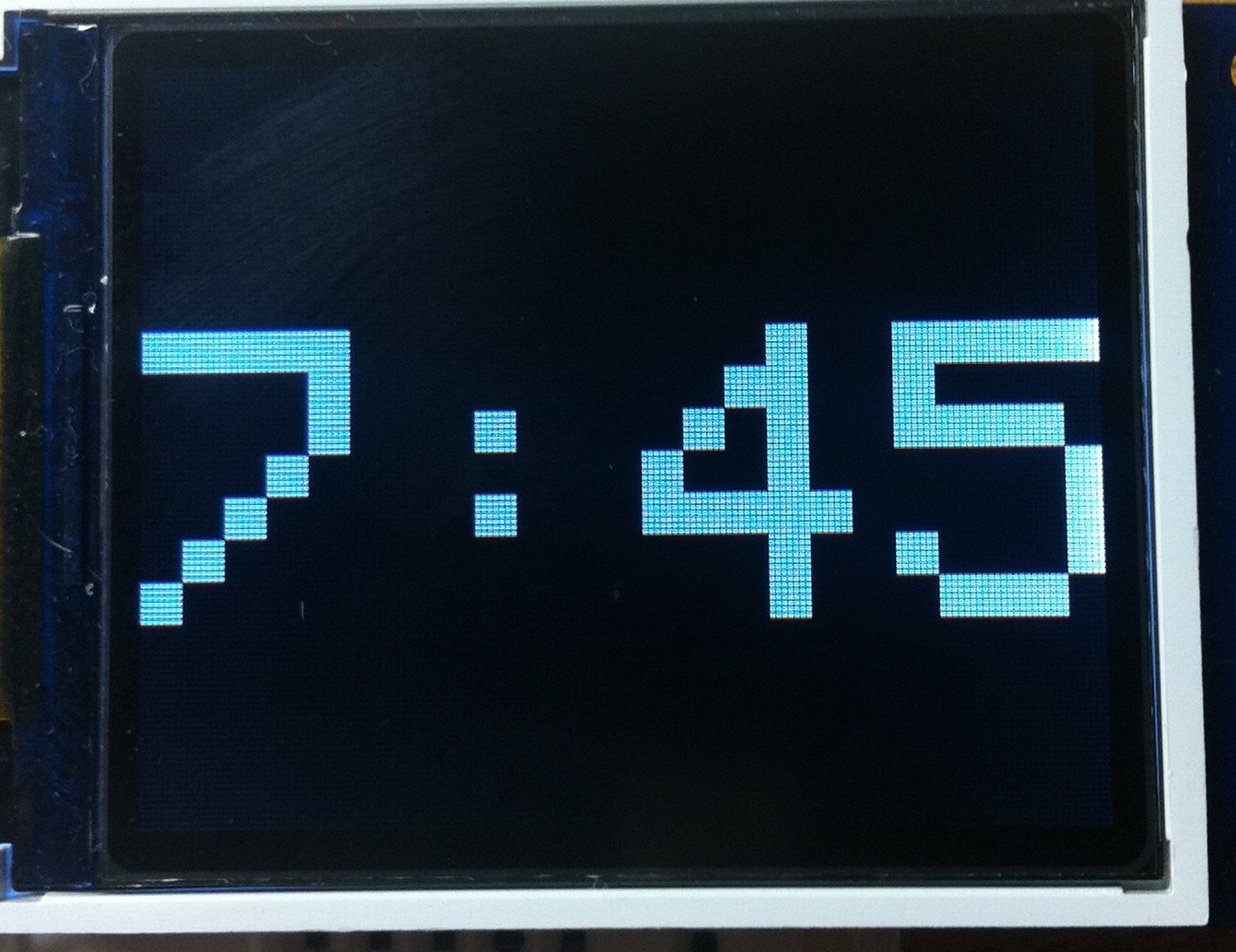
The displays will be controlled from the ESP microcontroller. Connection with the display hardware will be done with the ST7735 library and ILI9341 library for the 1.8” and 2.8” displays respectively, and graphics functionality will be achieved with the Adafruit GFX library. All three libraries have been modified to reduce the memory requirements of the microcontroller, as many of the functions included are not used. Each display has five wires running to it: power, ground, chip select, MOSI, and clock. The only unique wire for each is the chip select, all others are shared. The microcontroller sets which board will be communicated with via the chip select pin. Diagrams of the subsystem connections are included below.



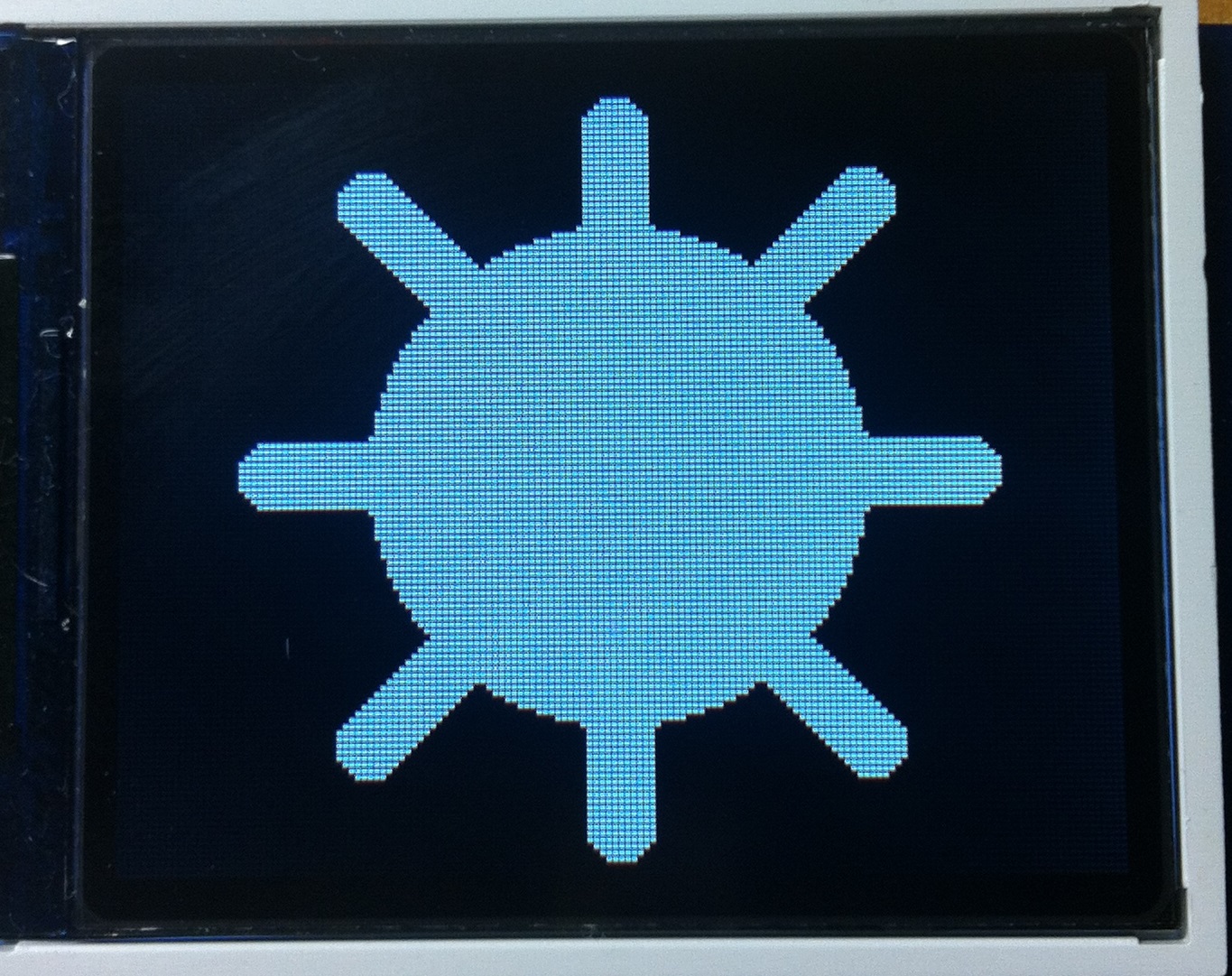


Display 1 is set up to receive a float variable of the current time from the internet subsystem. The float is but in a useable buffer, and a colon is added to make the time easier to understand.The time will be available in either a 12 or 24 hour format, depending on the setting the user chose during setup. Time will always come into the subsystem in 24 hour format, but if 12 hour format is required the system will update the value accordingly. Additionally, when the time is only three digits long, the text will automatically be scaled up and repositioned to allow for the largest and most visible information possible. Shown below are displays of a full 4 digit time, a three digit time with padding, and a three digit time with size increased.



Display 2 is set up to receive an integer from 1 to 5, where each number represents a current weather condition. The possibilities are sunny, cloudy/overcast, raining, storming, and snowing. An icon depicting the condition will be shown on display 2. To save space on the microcontroller and increase execution speed, the icons were drawn manually instead of being loaded from bmp files. This was done by positioning various elementary shapes into buffers using the adafruit gfx library. Circles, triangles, lines, and individual pixels were all utilized. Examples of each icon can be seen below (take note that the coloring looks slightly off in photos, but the images are pure black and white).

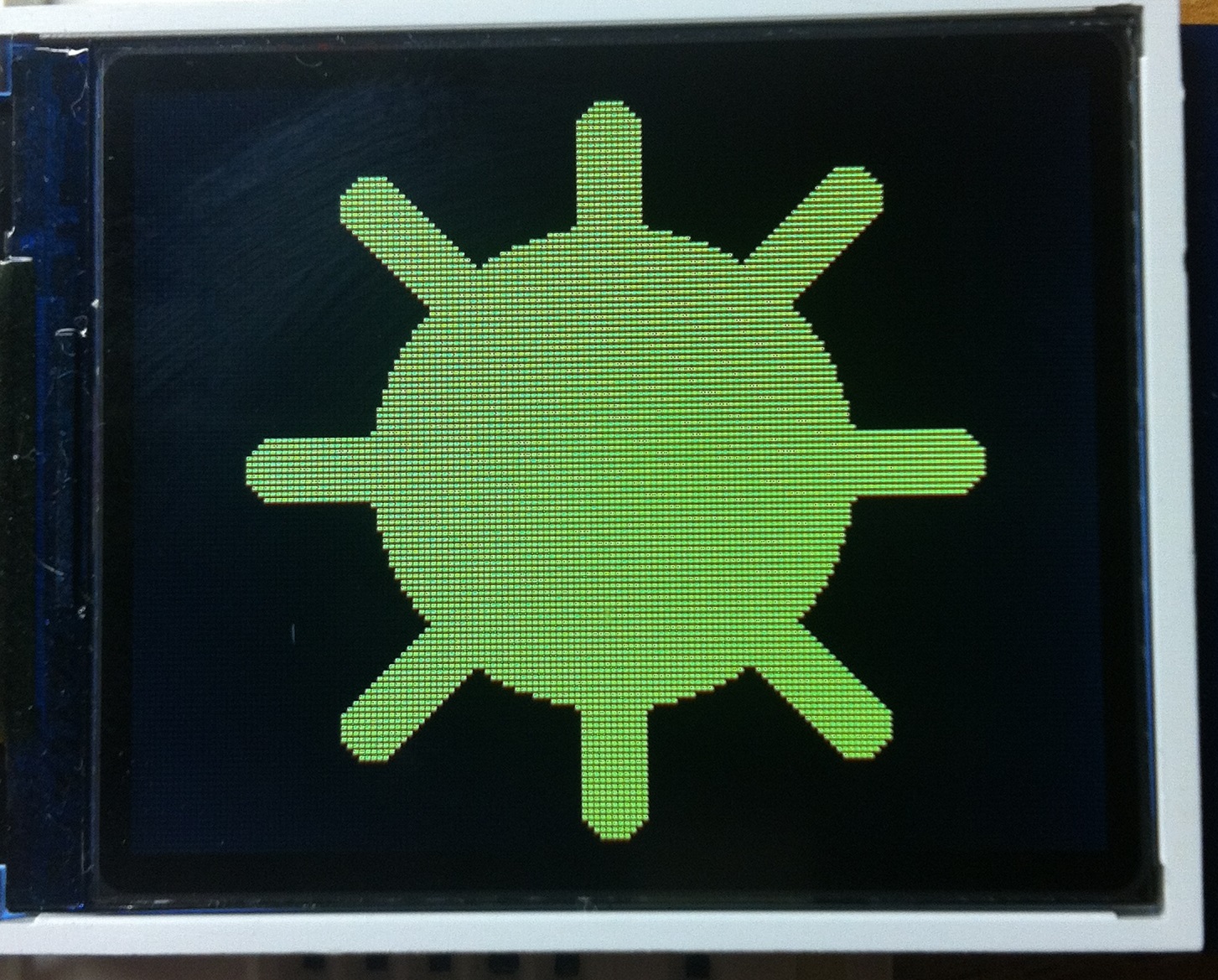
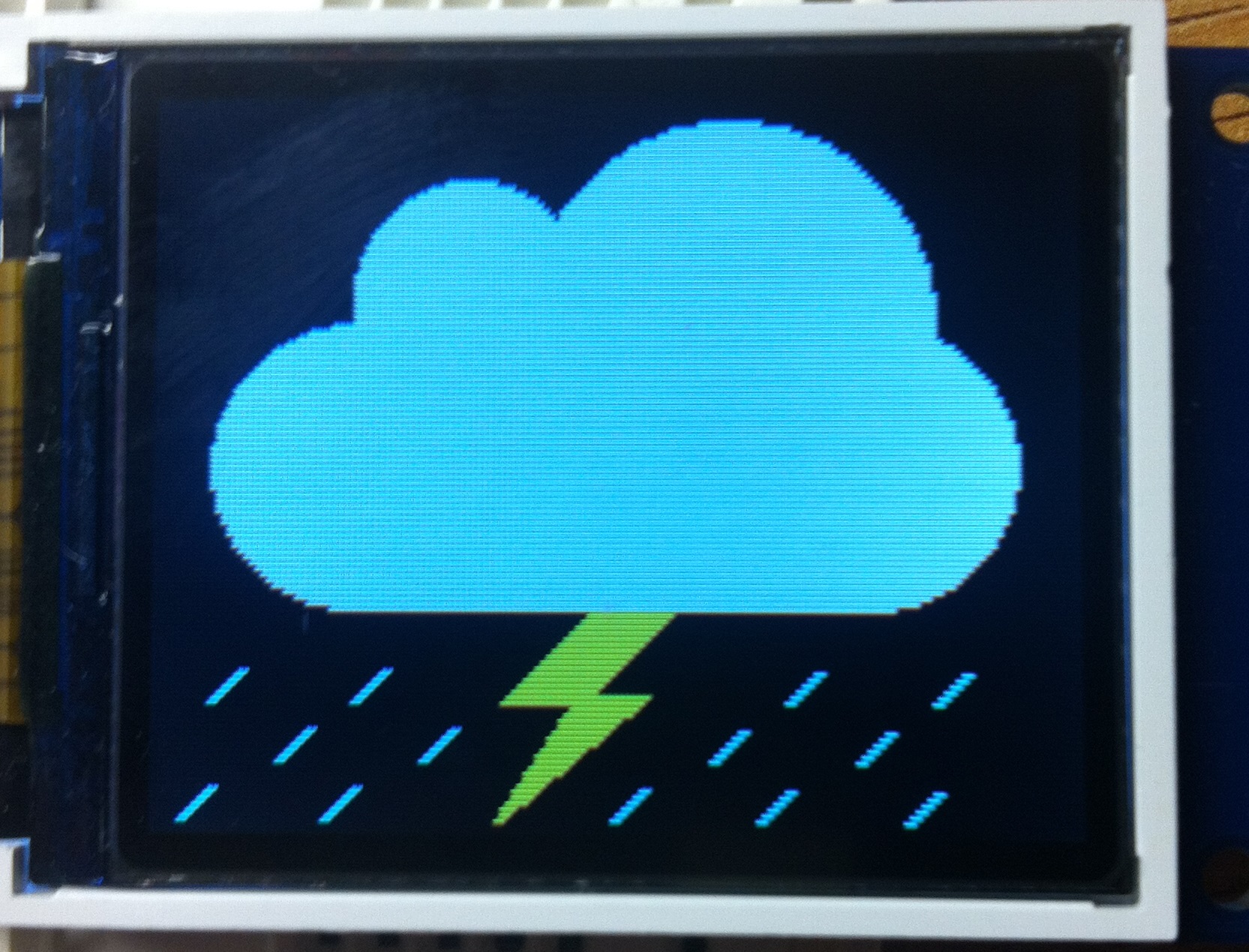
 



Display 3 is set up to receive text information from the internet subsystem in the form of character arrays. This information will be put up on the display as it comes in, and will be updated as much as possible. Because this is a larger screen, it allows for more information to be shown at once. At a minimum, the temperature, humidity, and the daily forecast will be shown. The top left and top right will show the temperature and humidity respectively. If the forecast is the only other information to be shown, it will be along the bottom of the screen. If there is additional information to be shown, it will scroll along the bottom of the screen, entering from the right and exiting to the left. The scrolling feature is achieved with reasonable speed by only updating ‘cells’ on the display where changes occur. The character array is processed into small sections, which are sent to the display backed with black coloring so that they can overwrite current characters. This makes for smooth scrolling and readable text.

Display 4 is able to receive and display any extra information as needed from the internet subsystem. Based on the style of information the text can be large or small, scrolling or motionless, and static or changing. Some information that might be displayed includes sports scores, stock information, an rss feed, or a news ticker.

It is possible for all displays to show the text and images in color. It is unlikely that the user will want this functionality, as color does not show through the mirror as well, but it will nonetheless be available. Examples can be seen below, where the sun has been colored yellow and the stormcloud has been colored grey, blue, and yellow for the cloud, rain, and lightning respectively.

**Speaker Subsystem**

The speaker subsystem allows for the user to be able to play music through the mirror via a bluetooth connection. Once the mirror has connected to the bluetooth device (likely a phone), it will continuously receive data from the device and will output music accordingly.

To enable a simpler and cheaper mirror, a premade bluetooth speaker was used. Instead of running the bluetooth connection and control information through the microcontroller, the speaker handles the majority of it, leaving little for the microcontroller. The TaoTronics Wireless Pocket Boombox was chosen for its price and performance. The speaker has been disassembled to allow for modifications, and to ensure it would fit behind the mirror glass. A diagram of subsystem connections is included below.



There were two main modifications to the TaoTronics speaker. First, the speaker drivers themselves were removed in order to have more flexibility in mounting position. Testing was done to see if larger drivers would produce better sound, but it was determined the internal amplifier was not powerful enough. The second modification was adding hardware to make the system controllable by the microcontroller. To do this, wires were added across the contacts of the various surface mount buttons on the speaker. In this way, either the ESP microcontroller can control the features or the buttons can be run externally (allowing for better buttons and more straightforward construction). An image of the disassembled device can be seen below (green wires are for power, red/blue are for pairing).



For control of the buttons, either the microcontroller can automatically connect the wires, or a button can be added externally for user access. When the microcontroller is used, it must be enabled for a precise amount of time to achieve the desired function, rather than reset the device. The following times were determined for successful operation: pairing button: 1-1.5 seconds, on button: 1.2-2 seconds, off button: 3-3.5 seconds.

One additional consideration that needs to be made for the speaker subsystem is whether a flexible membrane needs to be added to allow pressure in the system to equalize. Three such membranes were included in the speaker enclosure, but it is as of now unknown whether this will be necessary or not. If it is included, the existing membranes will simply be attached to the frame built for the mirror.