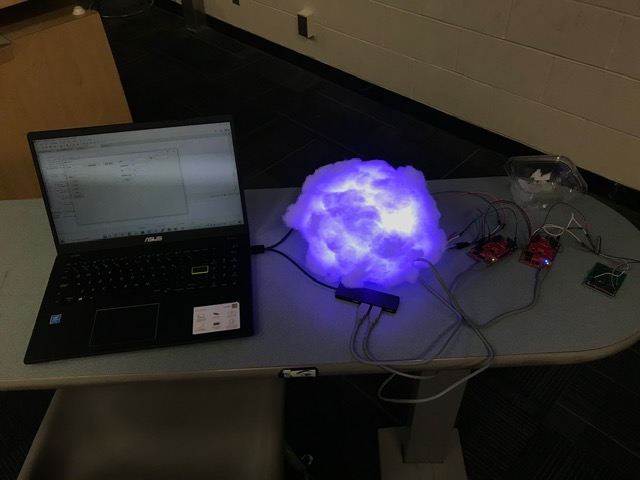
**MECH 423 Final Project: Indoor Cloud Lighting System**



Members: Kiran Prakash, Sophie Kenny

Emails: [kiran\_prakash@hotmail.co.uk](mailto:kiran_prakash@hotmail.co.uk),

sophiekenny916@gmail.com

Date: 7/12/2022

Abstract

This device is a lighting system that is connected to sensors and a GUI. The sensors are intended to acquire data of the weather outside and light up the cloud accordingly, yielding a more diffused, natural-looking light. The cloud lights up based on the type of weather identified by the sensors.

The GUI acquires and outputs data from the sensors and also accepts input from a user to set a certain weather type, instead of finding it from the sensors. The firmware is run on 2 MSP430s; one uses many of its modules to control the flow of data between the sensors, LEDs, and GUI. The other MSP430 is used only to light up the LED strip. The firmware uses the UART module to accept and output data from the GUI. It uses the ADC module to accept analog data from the sensors and output 8-bit values. Finally, it uses Timer B1 to continuously accept data from the sensors, to check if the GUI has sent any data, to identify the type of weather, and to output signals to the LED circuit to turn them on.

The hardware consists of the sensor circuit, the LED strip, the MSP430, and a laptop to run the GUI. The sensor circuit comprises several sensors assembled within voltage dividers to output a large range of signals, these ranges are described below. The LED strip comprises the LEDs, bit addressable by their WS2811 ICs, which are the lights. This strip can light up in many ways, but is used in this project to fill the cloud with a single color for each type of weather.

Objectives

The goal of this project is to create a cloud-like lighting system that has LEDs that change based on the weather outside. The device will change color based on the type of weather. For example, if it is raining, snowing, sunny, or cloudy. We will design and build a system which appears like a cloud and houses the LEDs. The sensors will be connected but placed in a separate module to be attached in an outdoor environment to observe the weather using sensors (temperature, light, and humidity). Once the data is obtained from the sensors, it will be processed by the MSP430, which will output different patterns of colors to the LEDs. These changes will be based on thresholds or regions for the temperature, brightness, and humidity that will be calibrated from real data. The device will also have manual settings if users want to contrive certain permutations or additional custom conditions (ex. a starry sky or an aurora). The final product will be a lighting system that is user adjustable, but capable of creating a more natural atmosphere in the room that it is in by bringing the sky indoors with real time data.

The following table outlines the tasks that were and were not accomplished.

Table 1 - Tasks Accomplished and Not Accomplished

| Tasks Accomplished | Tasks Not Accomplished |
| --- | --- |
| FR1: Find and design, temperature sensor, humidity sensor, photoresistor, and LED interface circuits. | FR5: Develop motor and driver electronics. Motor not implemented due to time constraints |
| FR2: MSP430 code to measure temperature, pressure, moisture and brightness values from the sensors | FR6: MSP430 code to make the clouds swirl and rotate if it's raining/stormy. Requires use of the motor which was not used due to time constraints |
| FR3: C# interface for sensor calibration and testing | Data could not be calibrated to output an actual temperature, just the corresponding 8-bit value |
| FR4: MSP430 code to change the colors of the LEDs based on weather conditions determined from the temperature, humidity, and light sensor values. MSP430 code to create various color patterns for different weather | Did not create custom weather types and corresponding custom LED patterns. |
| FR7: C# GUI used to change LEDs based on manual settings or sensor data |  |

Rationale

This project is interesting and different from other lighting mechanisms because it will make rooms more beautiful and natural looking with lighting that is dynamic and changes based on the weather. It also functions as an artful but intuitive weather station and clock. It is different from just lighting a room one color or having very artificial lighting colors because it simulates the outside world and changes throughout the day.

Summary of Function Requirements (FRs)

The table below outlines the requirements that were completed.

Table 2 - Functional Requirements

| Functional Requirements | % Effort | Responsible Person |
| --- | --- | --- |
| FR#1: Find and design, temperature sensor, humidity sensor, photoresistor, and LED interface circuits. | 20% | Kiran/Sophie |
| FR#2: MSP430 code to measure temperature, pressure, moisture and brightness values from the sensors | 20% | Kiran |
| FR#3: C# interface for sensor calibration and testing | 10% | Kiran |
| FR#4: MSP430 code to change the colors of the LEDs based on weather conditions determined from the temperature, humidity, and light sensor values. MSP430 code to create various color patterns for different weather | 30% | Sophie |
| FR#7: C# GUI used to change LED colours based on manual settings or sensor data | 20% | Kiran |

Functional Requirement 1 - Find and design, temperature sensor, humidity sensor, photoresistor, and LED interface circuits.

Approach & Design

This FR mainly pertains to electrical analysis. Sensors were chosen to provide an analog output (vs output from an I2C/SPI data line, for example) for simpler integration with the system. The circuits are designed such that they output a large range of voltages for their operating ranges. The final circuit design is shown below, which simply implements voltage dividers to yield an output for each sensor.

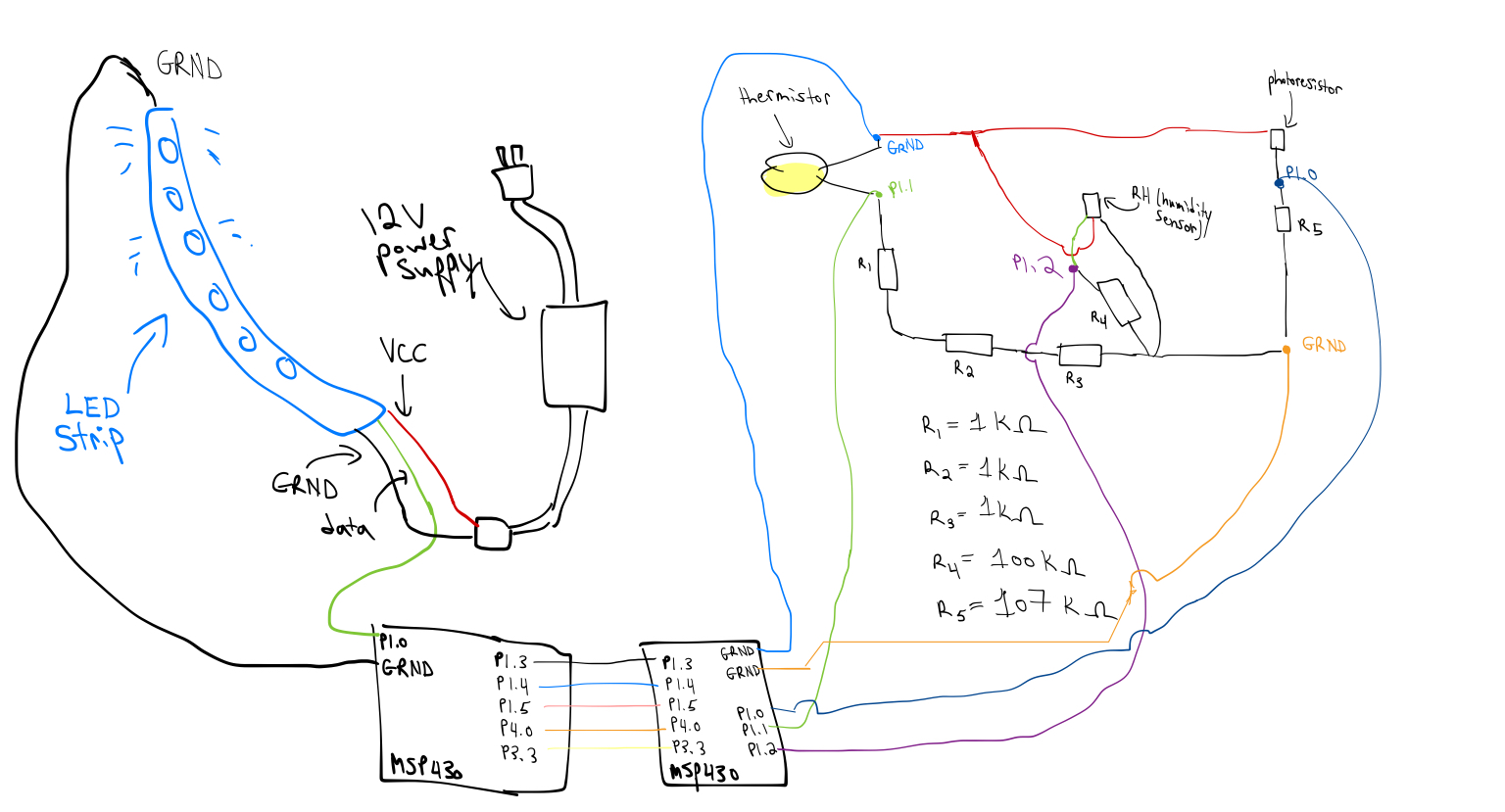


Figure 1 - Circuit Diagram For Sensors

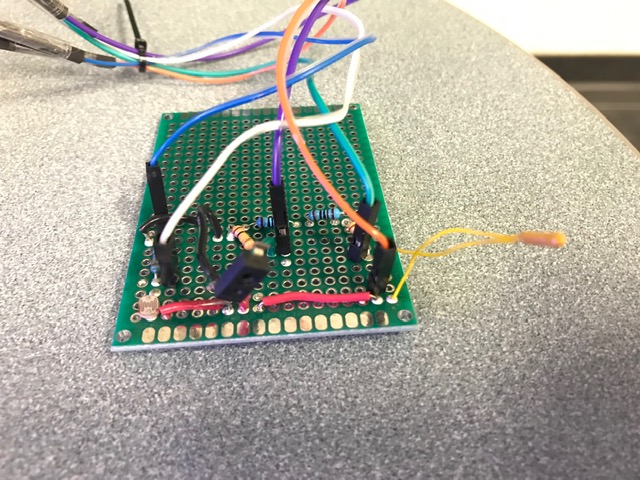


Figure 2 - Final Circuit Protoboard

Inputs and Outputs

Inputs:

* VCC to supply the sensors and the voltage dividers = 3.6V
* Measurand values for each sensor (ie. light, RH, and temperature)

Outputs:

* Analog outputs of each sensor/divider.

Parameters

The parameters used for this are the resistor values, which are fine tuned to yield a maximum analog output range.

Testing and Analysis

The circuit is shown in the figure above, this yields large ranges of analog voltages and corresponding 8-bit values; they are shown below. While other resistors could have been used to maximize the range for each sensor, not all values of resistors were available during the build of this circuit. Resistors were found from a scrap bin.

Table 3 - Sensor Signal Values

| Sensor | Min Signal (V; 8-Bit) | Room Signal (V; 8-Bit) | Max Signal (V; 8-Bit) |
| --- | --- | --- | --- |
| Photoresistor | 0.08; 6 | 2.54; 180 | 3.32; 235 |
| Thermistor | 0.56; 40 | 1.06; 75 | 2.40; 170 |
| RH Sensor | 0.78; 55 | 1.13; 80 | 2.47; 175 |

Functional Requirement 2 - MSP430 code to measure temperature, pressure, moisture and brightness values from the sensors

Approach & Design

Since the sensors output analog voltages, the MSP will read the analog values from its ADC. The MSP430 will cycle between different analog inputs and read each as an 8-bit value. This is much less complex than setting up an SPI/I2C module and communicating with the sensors, if they had such an output.

Inputs and Outputs

Inputs

* Analog inputs from each sensor, ranging from 0-3.6V

Outputs

* 8-bit value value for temperature
* 8-bit value value for RH
* 8-bit value value for brightness

Parameters

* DAQ rate, determined by setting Timer B.1 frequency (TB1CCR0)

Testing and Analysis

The firmware for this FR uses UART to send the 8-bit values to the GUI as a packet. It also uses the ADC to convert the analog sensor data into a readable digital value for the MSP. Naturally, the firmware configures 3 pins as analog (A0, A1, A2) and it utilizes timer B1 to periodically acquire data from the ADC and send them to the GUI. The ranges of 8-bit values found are shown in the testing subsection of FR1.

Functional Requirement 3 - C# interface for sensor calibration and testing

Approach & Design

The sensors need to be calibrated and circuits designed to reach a maximal output range. Since the weather changes slowly, we used different mechanisms to contrive repeatable environmental conditions. With these conditions, the 8-bit sensor values can be connected to their corresponding environmental values. Based on these environmental conditions, different weather types can be made. A C# interface was developed to show the sensor outputs (see Figure 3 below) that communicated with the MSP430 to obtain the sensor values.

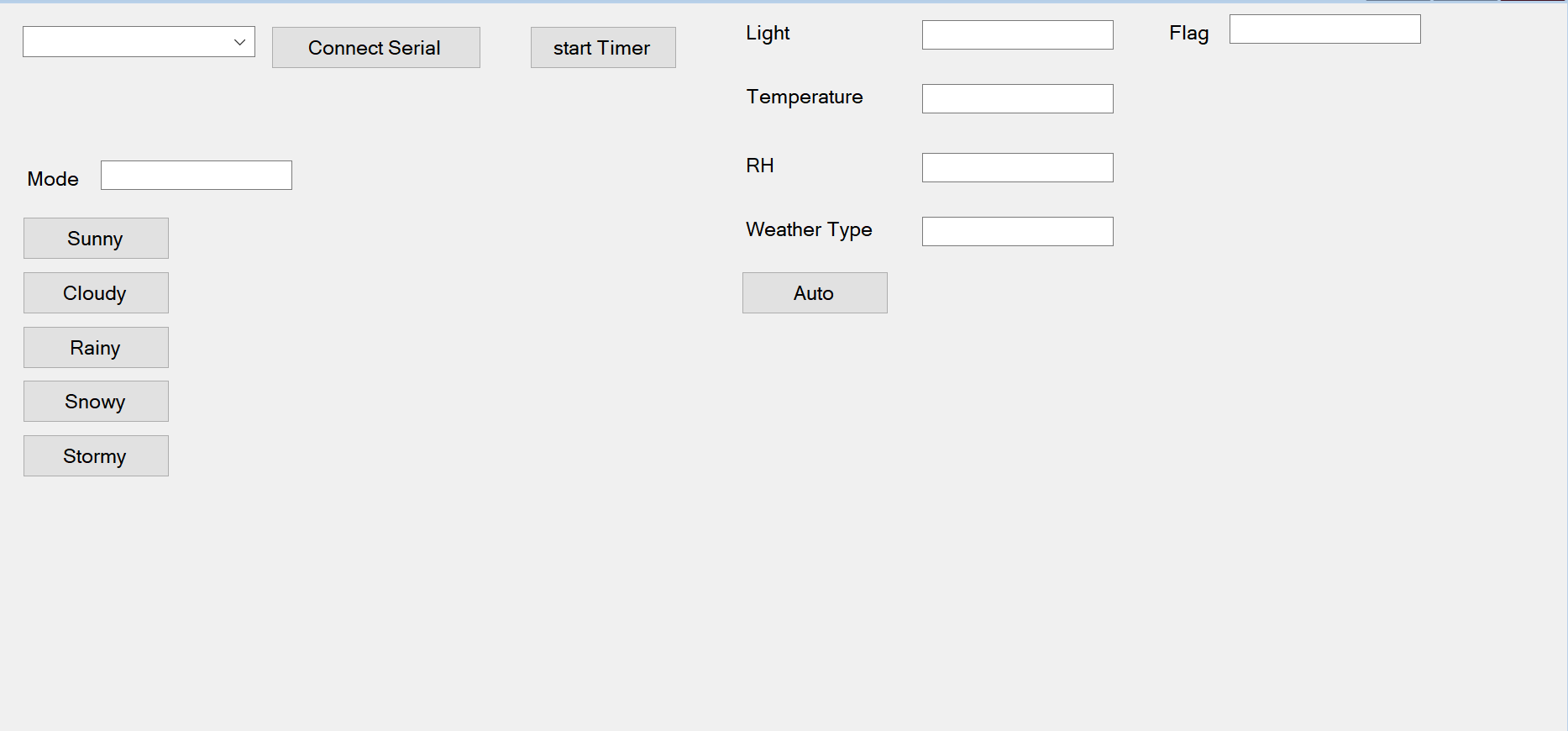


Figure 3 - C# GUI for Data Visualization

Temperature: A heat gun is used to generate hot temperatures (at the lowest setting 50 Celsius, blowing ~2 inches from the thermistor). The room environment simulates medium temperatures (~20 Celsius). A bath of cold water/ice for the thermistor to be submerged in simulates low temperatures (<5 Celsius)

RH: A heat gun is used to dry the RH sensor. Placing and holding a finger on the RH sensor replicates intermediate values. Moist towelettes are placed on the sensor to emulate humid environments.

Light: A finger is placed on the photoresistor to emulate dark environments, taken off ~ 2 inches to emulate medium-light, and the finger completely removed for bright environments.

Inputs and Outputs

Inputs

* Environmental values created by mechanisms described above

Outputs

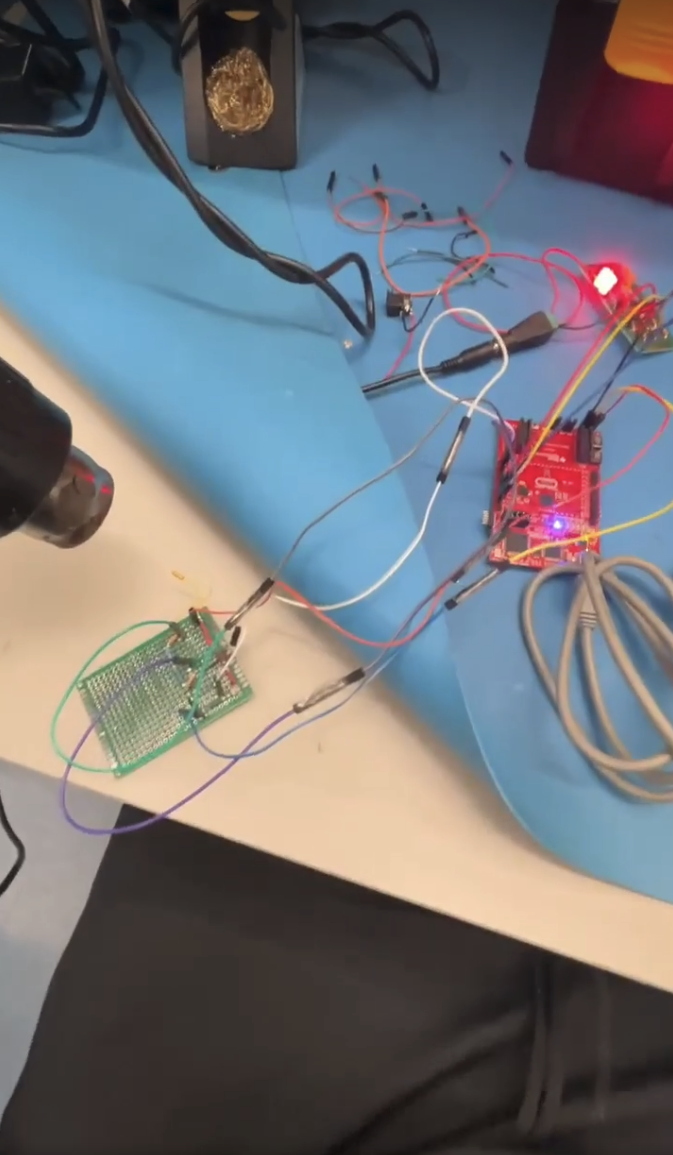
* 8-bit values representing the aforementioned environmental values on GUI

Parameters

* Calibration constants used to convert the 8-bit values to environmental values

Testing and Analysis

The environments were emulated according to the mechanisms described above. The weather types were successfully identified by the MSP430 in a repeatable fashion. This repeatable fashion confirms that the mechanisms used to contrive environments are effective. An example of this test for sunny weather is shown in the figure below.

Figure 4 - Sunny lighting (red LEDs) are outputted based on hot temperature (and reduced RH) from heat gun, and the photoresistor being exposed.

Functional Requirement 4

Approach & Design

With the sensor values calibrated and known, ranges of values are defined. These are used to define certain conditions required for each type of weather, according to the table below. The MSP should also be able to generate the color patterns and light up the LEDs, according to the second table below. One MSP430 took the data from the sensors and the other MSP sent the color pattern to the LED strip using bit banging. A series of bits were sent to determine the led color for the RGB light (8 bits for each of R, G and B for a total of 24 bits). The codes for sending zeros and ones using bit banging were calculated by finding the time that a low or high signal needed to be sent for a zero or a one on the LED datasheet. The signal was then sent by setting a pin to high/low and using nop() commands to keep the pin at that state for a certain number of clock cycles (one nop was one clock cycle and the time per cycle was calculated based on a 16MHz frequency). from the LED datasheet.The first MSP would determine the weather condition and send a high signal on a certain output pin to the other MSP, which would register which weather condition was being triggered.

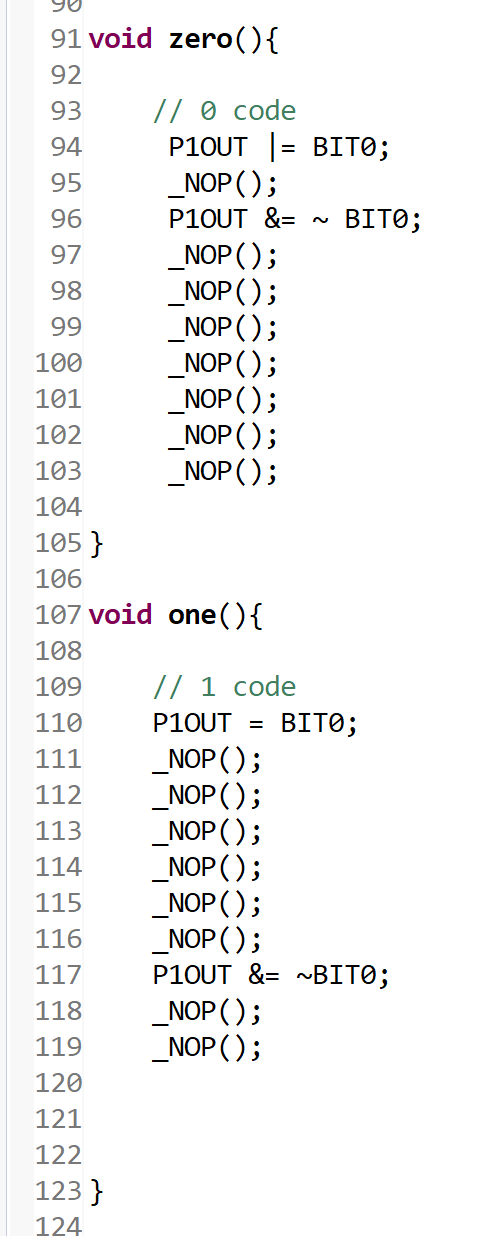


Figure 5 - Example Code for Activating LEDs

Table 4 - Weather Condition Trigger Conditions

| Weather | Photoresistor Condition | Thermistor Condition | RH Condition |
| --- | --- | --- | --- |
| Sunny | Bright | Hot | - |
| Cloudy | Medium | Medium | - |
| Rainy | Medium | Medium | High |
| Snowy | Medium | Cold (Below 0) | Medium |
| Stormy | Low | Cold (Above 0) | High |

Table 5 - Weather Condition Colours

| Weather Type | LED Colour | LED Mode |
| --- | --- | --- |
| Sunny | Red | Constant |
| Cloudy | Yellow, Blue | Constant |
| Rainy | Blue | Constant |
| Snowy | Yellow | Constant |
| Stormy | Blue | Flashing |

Inputs and Outputs

Inputs

* 8-bit values acquired from sensors

Outputs

* Type of weather
* LED color pattern

Parameters

* Thresholds to define ranges of values for each sensor

Testing and Analysis

The environments were emulated according to the mechanisms described above. The weather types were successfully identified by the MSP430 in a repeatable fashion. Each environment was tested repeatedly, and showed consistent LED outputs. An example of this test for sunny weather is shown in the figure in FR3.

Functional Requirement 5

Approach & Design

The GUI is built to read the sensor data from the MSP and display it, along with the associated type of weather. Additionally, if the user wishes to not use the sensors to change the LEDs, they may manually set the type of weather using the GUI, which stops the MSP from reading and sending data values and just lights up the LEDs accordingly. With this manual setting, further custom color patterns can be created. These settings are set using buttons on the GUI and it outputs values using textboxes.

Inputs and Outputs

Input

* User input to manually set the weather type, or have it set automatically

Output

* MSP detecting manual or auto setting
* LED color pattern

Parameters - None

Testing and Analysis

The FW uses UART to send readings and weather types as a packet to the GUI, and to receive single characters from the GUI to choose between manual and auto settings. Timer B1 is also used to acquire these data, and to create the color pattern. The FW has a UART interrupt which simply sets the mode (manual/auto), which is then analyzed in the timer interrupt.

System Evaluation

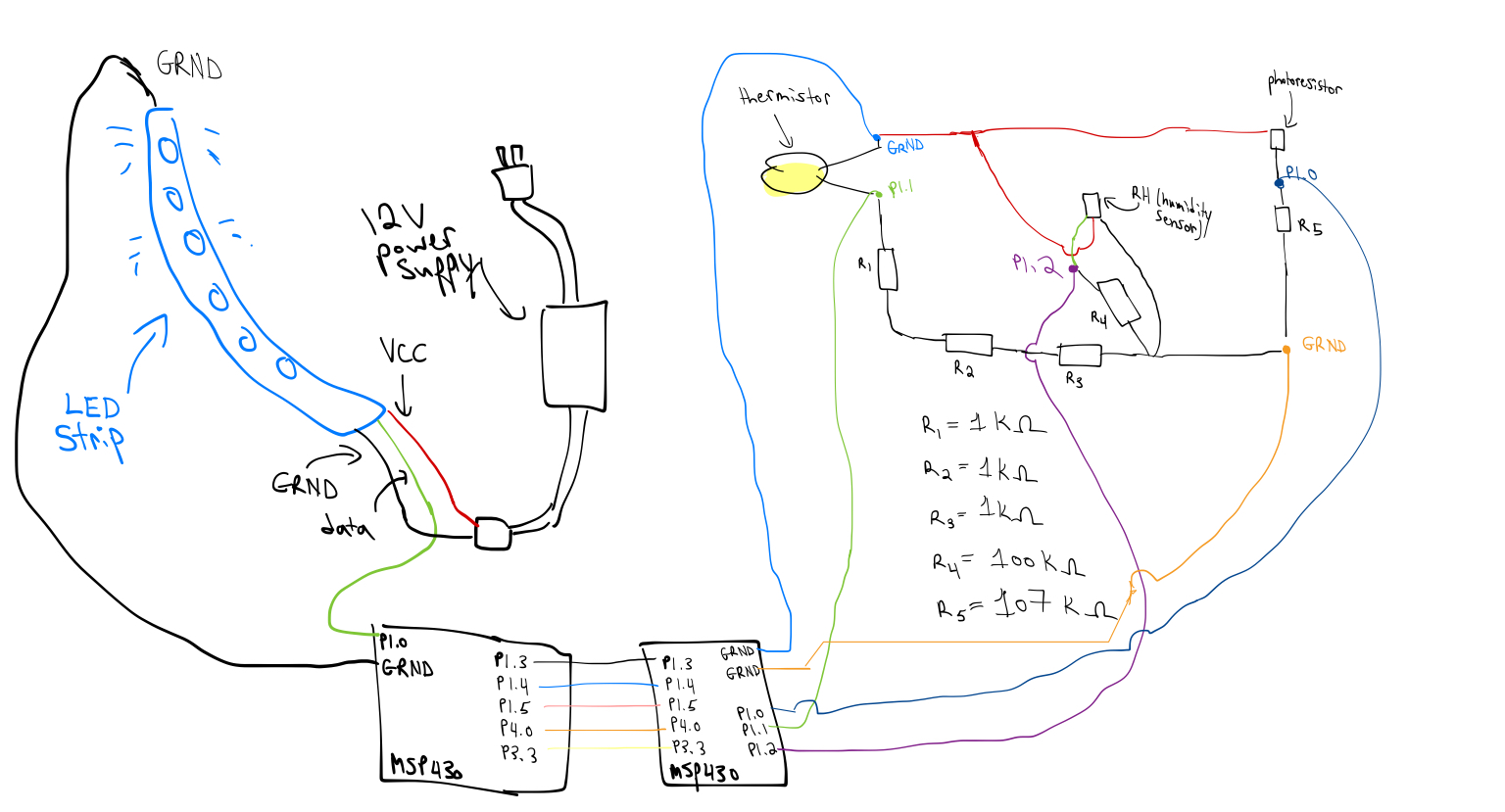


Figure 6 - Circuit Diagram of Entire Final System

As outlined above, the GUI was created to be able to set weather conditions manually and output sensor data for system testing purposes. To test that the sensor and LED system was working, we manually set the output with the GUI to see if the LEDs lit up with the expected color combination. Additionally, we determined that the sensors were outputting and being read correctly by imposing weather conditions on them (ie heating the temperature sensor) and checking to see if the readings changed as expected. Then, to combine the whole system, we set the system to automatic mode and imposed weather conditions on the sensors to verify that certain sensor conditions lead to a certain LED output. Shown in images below are tests using the GUI and sensors to change the LED colors.

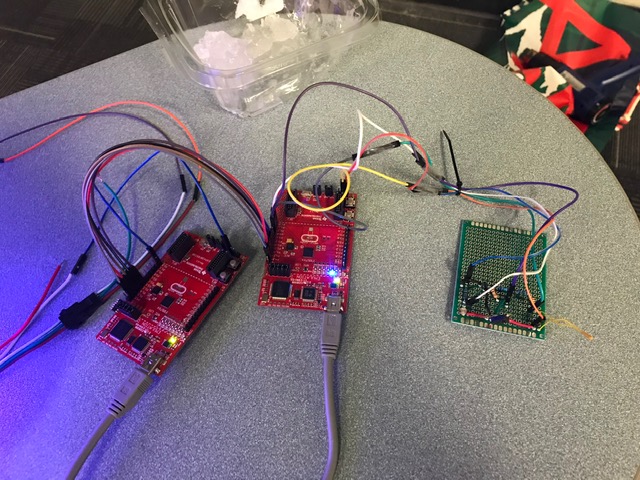


Figure 7 - MSP and Protoboard Setups

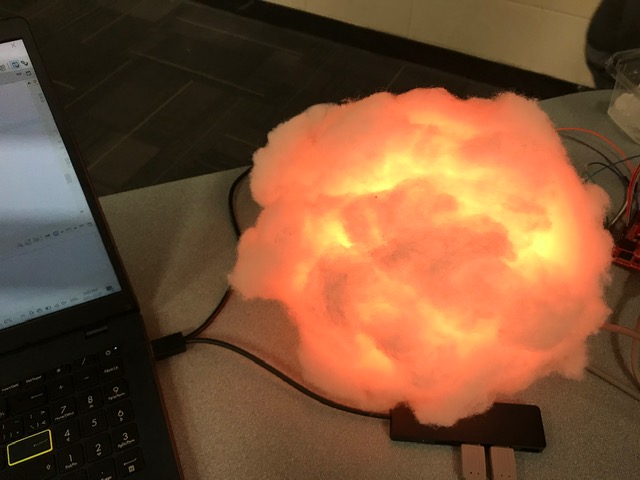


Figure 8 - Sunny(left) and Cloudy(right) LED Settings



Figure 9 - Rainy(left) and Snowy(right) LED Settings



Figure 10 - Stormy LED Settings

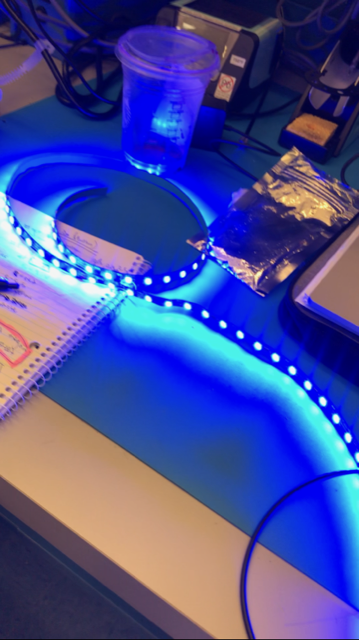


Figure 11 - LED Colour Testing to Verify Functionality

Reflections

For our project, what worked well was its conceptual simplicity. The communication was either from the sensors to the LEDs and user, or from the user to the LEDs, so it was simple and no issues with data synchronization were present. Also, there were no moving parts so no mechanical system had to be assessed. Using light materials such as the lantern and cotton to construct the actual light was also ideal. The real-time clock also would have worked well, but could not be implemented due to time constraints.

What could have been improved is as follows. Only 1 MSP430 is needed; a second one was used due to time constraints. One MSP would need to run correctly with the 16MHz clock. Furthermore, the real-time clock should be implemented. A pre-programmed hardware module could have been used and simply connected to the MSP for quick access to real time. Moreover, an LED strip with a much better documented library for the firmware should have been used as learning how the LEDs worked became the most difficult thing to figure out. More planning for that was needed. Moreover, 3D printing the housing for the sensor circuitry is acceptable for typical outdoor weather conditions, but becomes much less reliable when trying to contrive weather conditions. The heat gun would melt and deform the PLA. An acrylic housing would be better for this, although still somewhat susceptible to these issues.

The 3 most useful things learned from MECH 423 are as follows. Firstly, learning to read IC datasheets, although not completely, was incredibly useful in industry for learning how more complex circuitry operates. Reading datasheets generally has become much easier to do, which is an integral skill for any electronic position. Secondly, learning how an MCU operates (with its different modules, and how to configure them) was very useful since most of the programming learned in the mechatronics program is high-level. This knowledge bridges the gap between our knowledge of hardware and software. Thirdly, learning how to communicate using UART was a useful introduction to communication protocols and packets. Again, this replaces some of the high-level abstraction from software with clearer illustrations of what is sent during communication. Although the MSP430 does not support this, learning SPI and I2C during lectures is recommended.

As mechatronics engineers, our prime repertoire of knowledge pertains to control systems. These systems are themselves the bridges between different fields of physics (magnetic, electrical, mechanical, thermofluid, etc) and we are suited to translate the information from one field to another. This also means that we are not experts in specific fields (other than control systems). Learning more about the specific fields is one goal, and another is to learn completely different fields of knowledge.  
  
Firstly, learning more about semiconductors and digital logic circuits would help solidify our understanding of ICs. There are several online resources and courses that can be taken for this. Secondly, learning ML would vastly enable and improve applications of other fields of knowledge. For example, our project could have used a training set of data to more robustly calibrate the sensors based on outdoor conditions. Since this is a computer science topic, there are countless online resources to learn this at our own pace. Thirdly, as an engineer, there are not many opportunities in school to understand the economics of a project. Topics would include the resources required and available for each task, and estimating the value that each task brings. These topics would allow us to use our engineering prowess to make businesses on sound innovations.