EECS 113 Final Project:
Smart Weather Meter

Ryan M. Morrison
Jonathan Lam
Esteban Gomez
Group 3

Introduction

The purpose of this project is to create internet-enabled weather reporting device. The device reports various data related to current weather conditions such as temperature, wind, and precipitation. Additionally, the device provides an irrigation scheduler which determines the amount of daily water requirements and initiates an irrigation pump at determined intervals. In order to accomplish this, datum fetched from the California Irrigation Management System (CIMIS) and local sensors are leveraged. The user-interface is designed to deliver information clearly and with logical organization. The underlying device software is designed with expandability, efficiency and automation in mind.

The system is designed to continuously run, displaying each screen of data in five second cycles. For a particular screen, data is updated when appropriate (see descriptions below). The screens can be manually cycle-selected by pressing a button. When running, the system automatically checks for updated CIMIS daily and hourly reports and updates data accordingly.

Project Development

Software Implementation

The software package is designed with help from the wiringPi LCD library. This library allows for interfacing of HD44780 variants with the Raspberry Pi. Functions from this library are used to perform tasks such as writing to the LCD.

The project-specific source file structure is designed in the following manner:

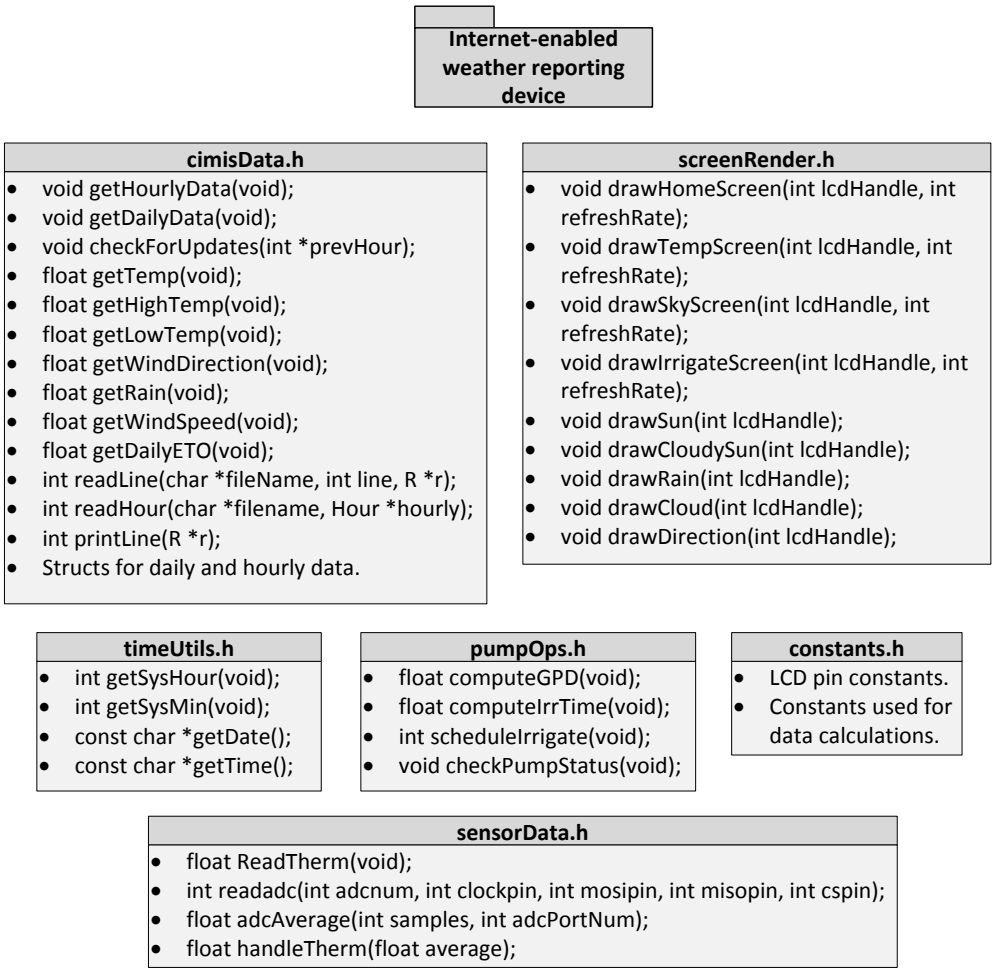

Figure 1: Project sketch of software structure with relevant functions.

The breakdown of each function is as follows:

1. **timeUtils.h**

   (a) `int getSysHour(void)` and `int getSysMin(void)`
   Convenience functions used to ascertain the current system hour and minute. These are used throughout the program for various tasks such as scheduling and updating data.

   (b) `const char *getDate()` and `const char *getTime()`
   Convenience functions used to ascertain a character representation of the system date and time. These are used to write the date and time to the LCD home screen.

2. **sensorData.h**

   (a) `float ReadTherm(void)`
   Ascertains a float representation of the temperature reading of the thermistor. This function calls the following functions and is called by the function that writes the temperature screen to the LCD.

   (b) `int readadc(int adcnum, int clockpin, int mosipin, int misopin, int cspin)`
   Ascertains an integer value as output from the ADC (in this case the thermistor reading). This is called by an averaging filter.

   (c) `float adcAverage(int samples, int adcPortNum)`
   This function serves as a 10-point averaging filter for ADC readings.

   (d) `float handleTherm(float average)`
   Converts the averaging filter output into degrees Fahrenheit.

3. **screenRender.h**

   (a) `void drawHomeScreen(int lcdHandle, int refreshRate)`
   Function to render the home screen to the LCD. It uses convenience functions from `timeUtils.h` to get the date and time. The refreshrate parameter determines the duration of the screen render with an internal loop.

   (b) `void drawTempScreen(int lcdHandle, int refreshRate)`
   Function to render the temperature screen to the LCD. It uses various functions from `sensorData.h` and `cimisData.h` to get the local, predicted high, and predicted low temperatures. Special defined LCD characters are used to render the graphic aspects of the screen. The refreshrate parameter determines the duration of the screen render with an internal loop.

   (c) `void drawSkyScreen(int lcdHandle, int refreshRate)`
   Function to render the weather conditions screen to the LCD. It uses various functions from `cimisData.h` to get the hourly wind and precipitation data points. Special defined LCD characters are used to render the graphic aspects of the screen. The refreshrate parameter determines the duration of the screen render with an internal loop.

   (d) `void drawIrrigateScreen(int lcdHandle, int refreshRate)`
   Function to render the weather conditions screen to the LCD. It uses various functions from `cimisData.h` and `timeUtils.h` to compute the ETo, gpd, and tpd data points. Special defined LCD characters are used to render the graphic aspects of the screen. The refreshrate parameter determines the duration of the screen render with an internal loop.

   (e) `void drawSun(int lcdHandle)`, `drawCloudySun(int lcdHandle)`, `drawRain(int lcdHandle)`, `drawCloud(int lcdHandle)`, and `drawDirection(int lcdHandle)`
   Convenience functions use by the above functions to render the various graphic elements to the LCD. These functions make large use of the wiringPi library.

4. **pumpOps.h**

   (a) `float computeGPD(void)` and `computeIrrTime(void)`
   Functions used to compute the gallons per day value and the daily percentage of irrigation time.

   (b) `int scheduleIrrigate(void)`
   Function used to determine the total seconds for each irrigation period.

   (c) `void checkPumpStatus(void)`
   This function calls the compute gpd function and the time functions to determine if it is time for irrigation. If so, the green LED is activated, if not, the red LED is activated.

5. **cimisData.h**

   (a) `void getHourlyData(void)` and `getDailyData(void)`
   Convenience functions used to fetch data from the CIMIS' daily and hourly reports. These functions are derived from the template file scanner file provided by the TA.

   (b) `void checkForUpdates(int *prevHour)`
   This function uses convenience functions from `timeUtils.h` to check the time and update CIMIS reports on a predetermined time schedule.

   (c) `float getTemp(void)`
   This function fetches the most current hourly temperature from CIMIS.

   (d) `float getHighTemp(void)` and `float getLowTemp(void)`
   These functions fetch previous 7-day high and low temperatures from CIMIS and averages them.

   (e) `float getWindDirection(void)`, `getRain(void)`, `getWindSpeed(void)`
   These functions fetch the most current hourly data from CIMIS for wind direction, wind speed, and precipitation.

   (f) `float getDailyETO(void)`
   This function fetches the previous day's ETo value and it is used to compute the gallons per day value.

   (g) `int readLine(char *fileName, int line, R *r)` and `readHour(char *filename, Hour *hourly)`
   Modified versions of the filescan template provided by the TA. Readline fetches data from CIMIS daily reports and readhour fetches data from CIMIS hourly reports. Both store all values in respective data structures.

Each of these functions are developed on a per-screen basis. For example, the `drawHomeScreen` function is developed first, with any dependencies for that being developed at the same time. For reasons of space, specific details of these functions are omitted.

Home Screen

This screen displays the current month, date, and year on the top row of the LCD. The current time is displayed on the bottom row, updated on the second. This is the first screen displayed in the screen cycle.

Figure 2: Home screen with current date and time.

Temperature Screen

This screen displays the following data:

- Thermometer icon— this provides a quick visual cue to indicate the type of data being displayed, in addition to indicating the unit of measure (°F).
- Current temperature—this provides a local measurement of temperature via the onboard thermistor. Data is updated on the second.
- Daily predicted high—this provides a predicted high temperature for the day as determined from CIMIS daily data. Data is updated on the day with screen refreshes.
- Daily predicted low—this provides a predicted low temperature for the day as determined from CIMIS daily data. Data is updated on the day with screen refreshes.

To ascertain the local temperature, a thermistor is used with an mcp3008 ADC. These devices are interfaced as follows:

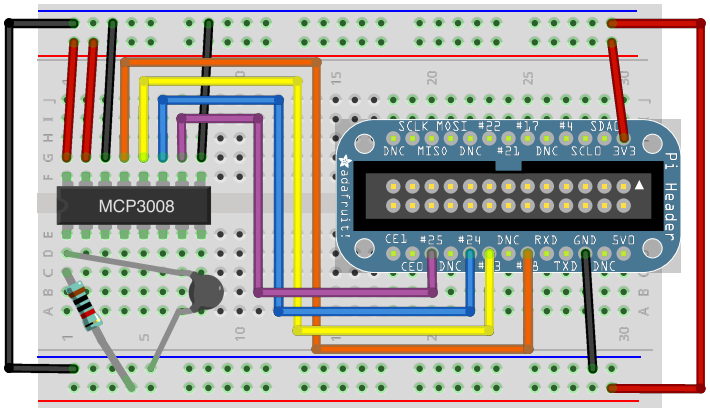

Figure 3: Fritzing diagram of the physical connection of thermistor and ADC.

The thermistor is connected to a 10kΩ pull-up resistor. From the circuit, the variable resistance of the thermistor is calculated.

$$V_{out} = V_{iadc} = V_{oadc} = \frac{adcRead(3.3)}{1023} \qquad (1) \qquad\qquad V_{out} = \left(\frac{R_t}{10\text{k}\Omega + R_t}\right) 3.3 \qquad (2)$$

$$\frac{adcRead(3.3)}{1023} = \left(\frac{R_t}{10\text{k}\Omega + R_t}\right) 3.3$$

$$\frac{adcRead}{1023} = \left(\frac{R_t}{10\text{k}\Omega + R_t}\right)$$

$$adcRead(10\text{k}\Omega) + R_t adcRead = 1023 R_t$$

$$R_t = \frac{adcRead(10\text{k}\Omega)}{1023 - adcRead}$$

The variable resistance equation is then used in the code to create a function that takes *adcRead*, the average of 10 samples, and outputs the resistance of the thermistor as heat sources are applied/removed.

The high and low temperatures are calculated by averaging the previous 7-day highs and lows from CIMIS.

1 **It is noted that thermistor results are unreliable. This is possibly rectified by improving the filter.**

Figure 4: Temperature screen with thermometer icon, current high, and current low temperatures.

Weather Condition Screen

This screen displays the following data:

- Wind direction and speed—the wind speed is displayed in mph and determined from CIMIS hourly data; the wind direction is displayed with an 8-point arrow icon and is determined from CIMIS hourly data. Both data points are updated on the hour with screen refreshes.
- Sky conditions—this provides a visual cue to indicate the amount of cloud cover in the sky and if there is rain present. Four scenarios are possible (all are updated on the hour with screen refreshes):
- Full sun—this icon is displayed for temperatures greater than/equal to 80°.
- Half-sun with clouds—this icon is displayed for temperatures less than 80° and greater than/equal to 68°.
- Full cloud—this icon is displayed for temperatures less than 68°.
- Cloud with rain—this icon is displayed when rainfall percentage is greater than 0.5%.
- Rainfall—the percentage of rainfall is displayed (inches) and is determined from CIMIS hourly data. Data is updated on the hour with screen refreshes.

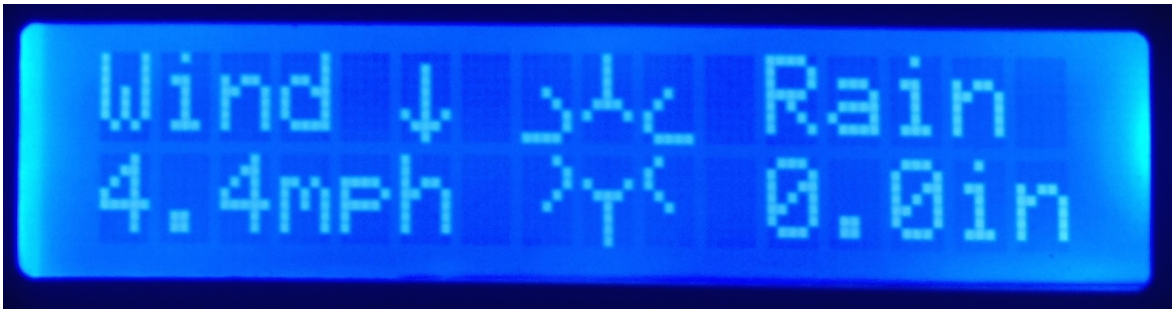

Figure 5: Weather conditions screen with wind speed and direction, current cloud cover icon, and precipitation.

Irrigation Schedule Screen

This screen displays the following data:

- Water drip icons—this provides a quick visual cue to indicate the type of data being displayed.
- Gallons per day—as determined from CIMIS daily data and software functions, this value displays the total amount of water (gallons) to be irrigated for the current day. Data is updated daily with screen refreshes.
- Time per day—Gallons per day—as determined from CIMIS daily data and software functions, this value displays the percentage of daily time allotted to irrigation for the current day. Data is updated daily with screen refreshes.
- Pump status—the ON/OFF status of the pump is displayed.

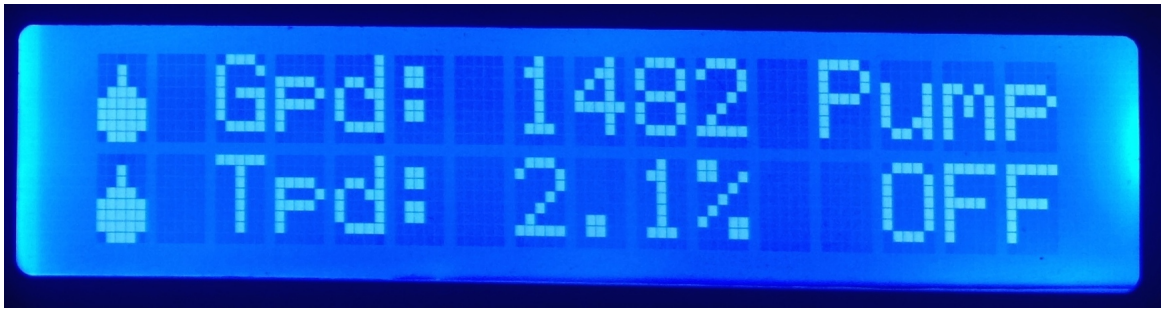

Figure 6: Irrigation schedule screen with icons, gpd, tpd, and pump status.

The following equation is used to determine the gallons per day value:

$$Gpd = \frac{ETo \cdot PF \cdot SF \cdot 0.62}{IE}$$

Where :

- ETo is fetched from CIMIS
- PF is the plant factor (1)
- SF is the square footage of irrigated land (8150)
- IE is the irrigation efficiency (.75)

The time per day ratio is calculated by the following:

$$Tpd = \frac{Gpd}{Gph}$$

Where Gph is the specific gallons per hour for a pump (2900).

System Status LEDs and Screen Select with Button

The system has two LEDs, one green and one red, which indicate the pump status. Illuminated green means the pump is on, illuminated red means pump is off. A push button is supplied to allow manual cycling of the data screens. These components are interfaced as follows:

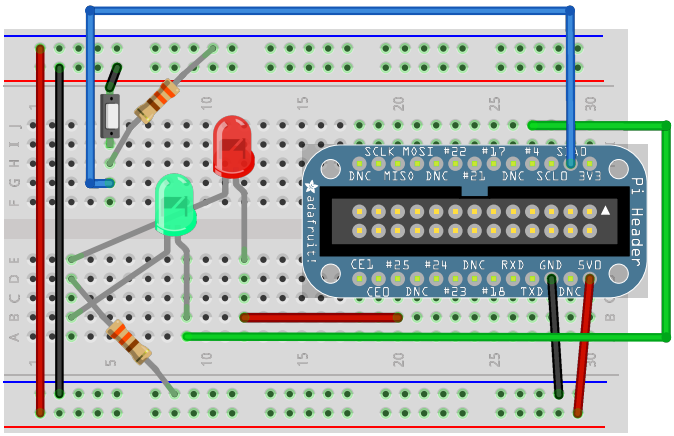

Figure 7: LEDs and push button interfaced with the circuit.

To facilitate the ability to manually select screens, conditionals are placed inside the screen render functions that check if the pin is high or low. The LEDs are controlled from within the `checkPumpStatus` function.

Complete Project

The complete circuit schematic is pictured below:

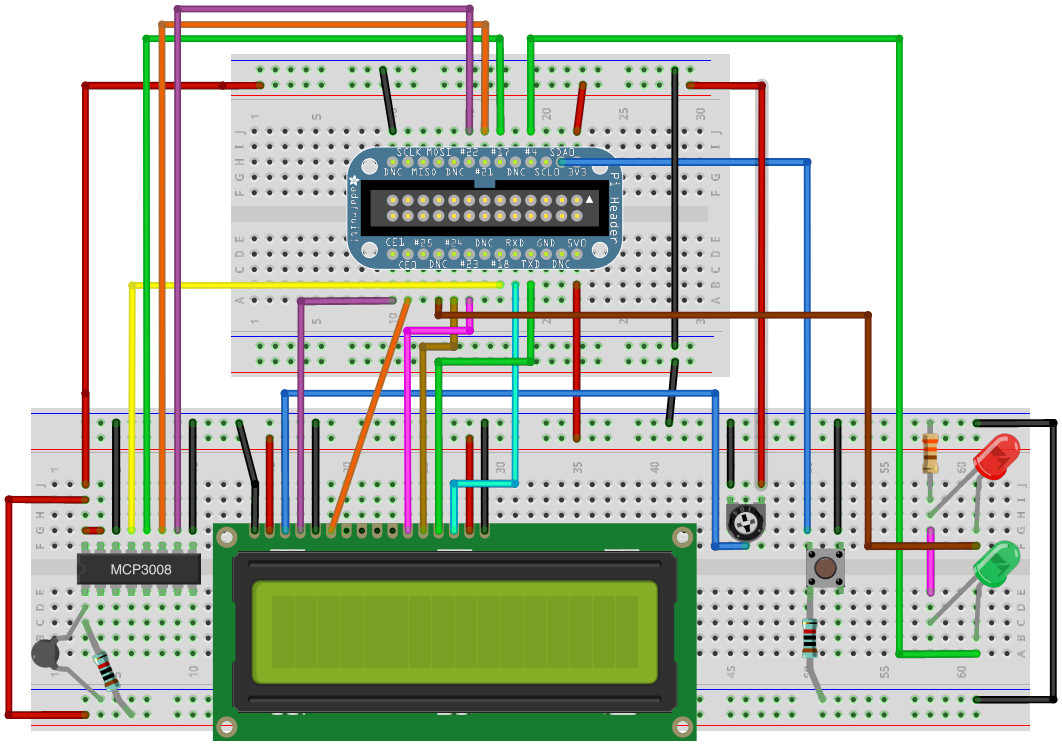

Figure 8: Fritzing diagram of the complete circuit.

The complete physical circuit is pictured below:

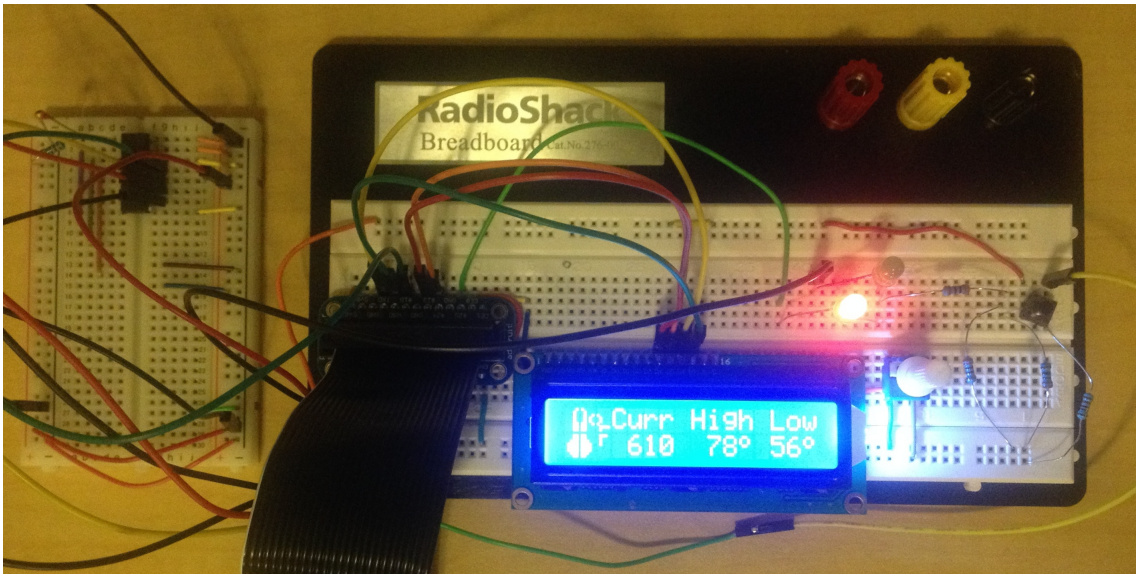

Figure 9: Completed physical circuit.

References

Raspberry Pi website: http://www.raspberrypi.org

WiringPi website: http://wiringpi.com

Wiring RPi and MCP3008 ADC: https://learn.adafruit.com/send-raspberry-pi-data-to-cosm/connecting-the-cobbler-slash-mcp3008-slash-tmp36

LCM1602A LCD data sheet: http://www.adrirobot.it/datasheet/LCD/PDF/LCM1602A.pdf

CIMIS website: http://wwwcimis.water.ca.gov/cimis/infoEtoOverview.jsp;jsessionid=9F4F5B2EC494B1062AD81247587639D0

Notes on irrigation: http://www.irrigationtutorials.com/how-to-estimate-water-useage-required-for-an-irrigation-system/

Reference weather station: http://www.bhiltd.co.uk/html/wireless_weather_station.html

Thermistor tutorial: https://learn.adafruit.com/thermistor/using-a-thermistor

Irrigation pump used for this project: http://www.homedepot.com/p/Wayne-1-1-2-HP-Cast-Iron-Quick-Prime-Lawn-Sprinkler-Pump-WLS150/100017195