CECS 447 Spring 2021 Project 5

An Embedded Weather Quest

By

Jesus Franco

5/5/2021

This project will use the CC3100GetWeather_4C123 project and be tweaked to allow the user to manually input a location to a Serial Terminal and output information onto the Serial Terminal and ST7335R LCD. Also depending on the location weather status, a short animation will be displayed on the ST7335R display.

Introduction:
This project will allow the use of Johnathan Valvano's CC3100GetWeather_4C123 project to be tweaked to create the Embedded Weather Quest. A CC3100 Module BoosterPack will be used to allow the user to access the internet and will allow the user to manually input the city name, city ID, geographic coordinates, or zip code to get location information such as temperatures, humidity, and weather states. The information will be displayed on the ST7335R LCD display for the top half of the display and will display the current weather state animation for the bottom half of the display. If either onboard launchpad switches are pressed, the program will cycle all over again and await new manual inputs.

Port Table:

Name	**I/0**	**Port**	**Description**
ST7735 LCD SCK	Output	PA2	A 50% duty cycle clock generated by the master.
ST7735 LCD SDA	Output	PA5	A I2C bus serial data line.
ST7735 LCD A0	Output	PA6	Data/Command.
ST7735 LCD RESET	Output	PA7	Reset.
ST7735 LCD CS	Output	PA3	Chip enable.

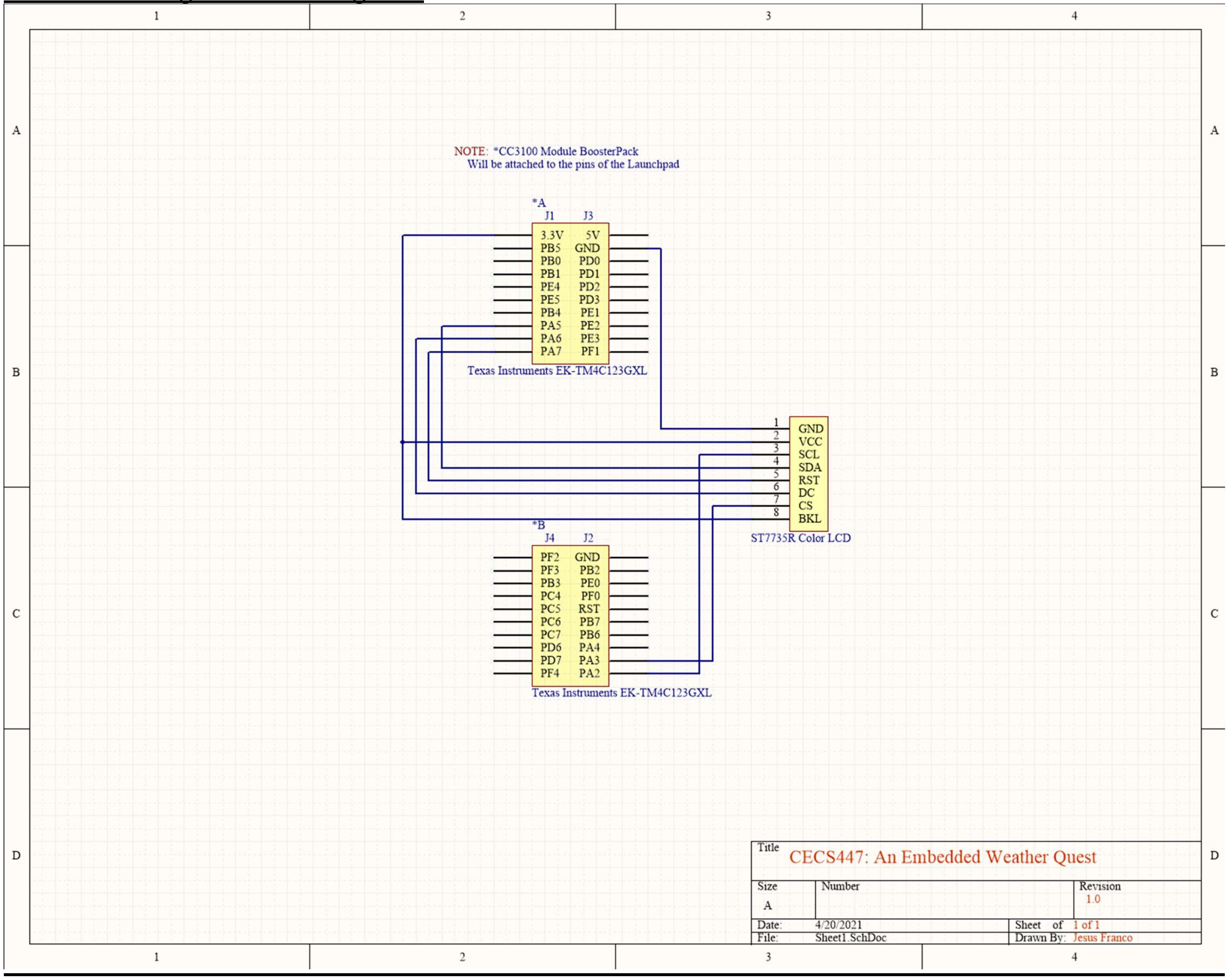
Operation:

To fulfill the operation of this project, there needed to be a deep understanding on how the example code worked. After a quick realization of what the code is doing, the next process is to run the example code. The example code required an access point to access the internet, so my home Wi-Fi was the remaining factor; however, there was a tiny issue. The CC3100 Boost Pack only functions with 2G networks instead of 5G, so that was a quick fix. Next step was to bring in a serial terminal and start running the example code. The example code outputted the received information coming from the requested API to the serial terminal and this is where we can get some progress going. Using the current requested API command, the next step was to parse through the data and receive specific parts of the output such as the city name, current temperature, minimum temperature, maximum temperature, humidity, and weather status. Luckily, this portion was somewhat given to us and is in the main2015.c file. After a bit of tweaking, the information was outputting correctly on the terminal. The next steps were to allow user manual input to allow the APIs to search with conditions such as city name, city ID, geographic coordinates, and zip code. It is logical that I would have to combine strings using the strcat function, but the tricky part was finding the correct commands. After some research, the commands were found at the API website, https://openweathermap.org/current; however, only the initial part of the command had to be tweaked. The next step was to get a APPID which is given after creating an account. Once combining these commands, a string that will hold the command is created. Lastly, all that is missing is the LCD display. A quick main plot display is creating displaying the background and the initial template for the information. Next displaying the required information onto the LCD display is set at the correct location. Finally, the only thing missing would be animations for a clear sky, cloudy sky, and rainy sky. These animations would be placed in the inner while loop of where the code is hanging. This will make it possible for it to run forever until SW1 or SW2 is pressed, which will recycle the whole program and await the user for new manual inputs for location.

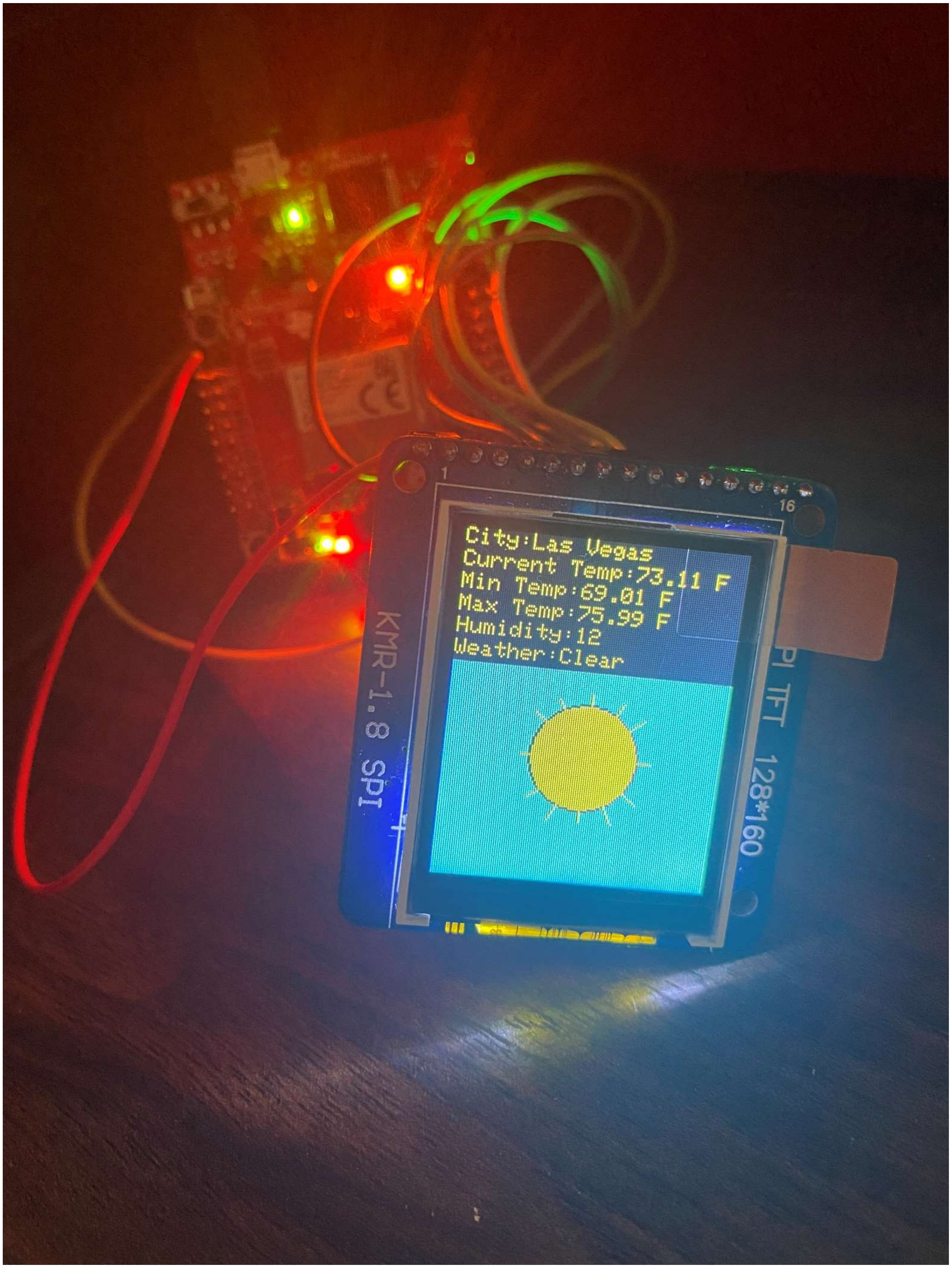
Theory:

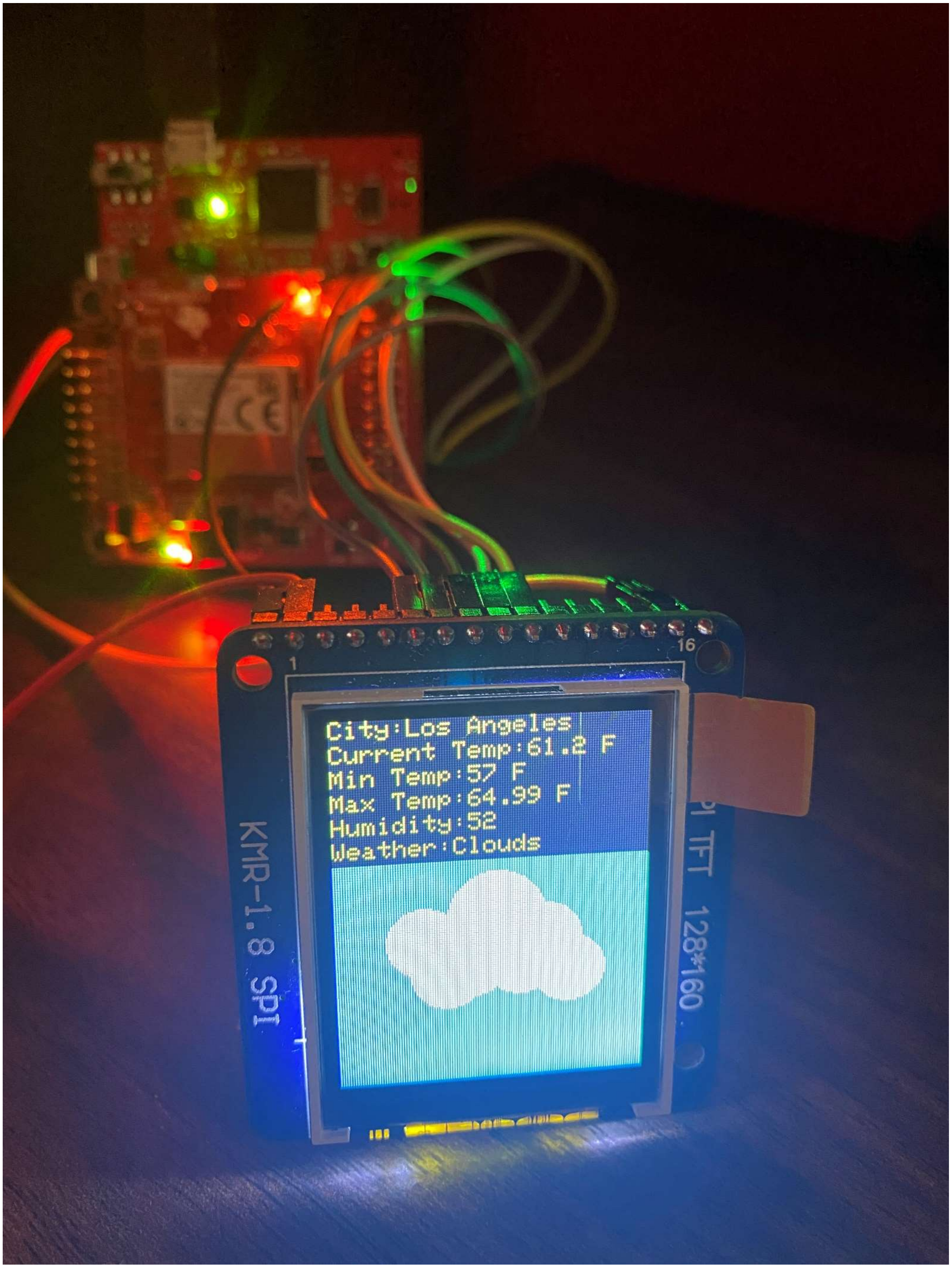
In theory, this project required a lot of work. To get started on this project, there needed to be an understand of what exactly was going on. Firstly, I had to run the example code to see what exactly will output on the Serial Terminal. At first, there was nothing showing up and that was because the CC3100 Booster Pack required a 2G Wi-Fi connection and I was connecting it to my 5G network. After this quick fix, I noticed that the output was a huge string outputting the current information about Long Beach. This was because the REQUEST variable was set in respect to the Long Beach location. Firstly, I had to allow myself to use the information given and output it in a specific order on the Terminal. I had to parse through the huge string to search and extract the required information. This was thankfully given in the example code, main2015.c, but it required a bit of tweaking to get a cleaner set of information. The next step was to allow myself to manually input the data. I outputted menu on the terminal to allow the user to see possible methods of searching for locations with manual inputs that consists of the city name, city ID, geographic coordinates, and zip code. This would all be done using the strcat function to combine strings and the requested api commands were found in this website, https://openweathermap.org/current. There did need to be some tweaking to create the correct REQUEST command, but that website and some trial-and-error work was enough to figure it out. I also required an APPID but that was given by creating an account. After many trial and error attempts, I managed to get correct information through the serial terminal from manual inputs. The next steps were to implement the ST7335R LCD screen. A quick static plot was created to display an empty template with missing information. That is because the missing information is the information that was parsed through earlier. This will be displayed on the top half of the LCD display. For the bottom half, there will be a quick animation to determine if the location's weather status is either a clear sky, cloudy sky, or rainy sky. This is the most tedious work to create animated drawings to express those weather conditions. These animations would be in the final inner loop of the program where the loop's sole purpose is to hand there forever until SW1 or SW2 is pressed. If SW1 or SW2 is pressed, then the program will recycle and await new manual inputs for a new location search.

Hardware Design/ Circuit Diagram:

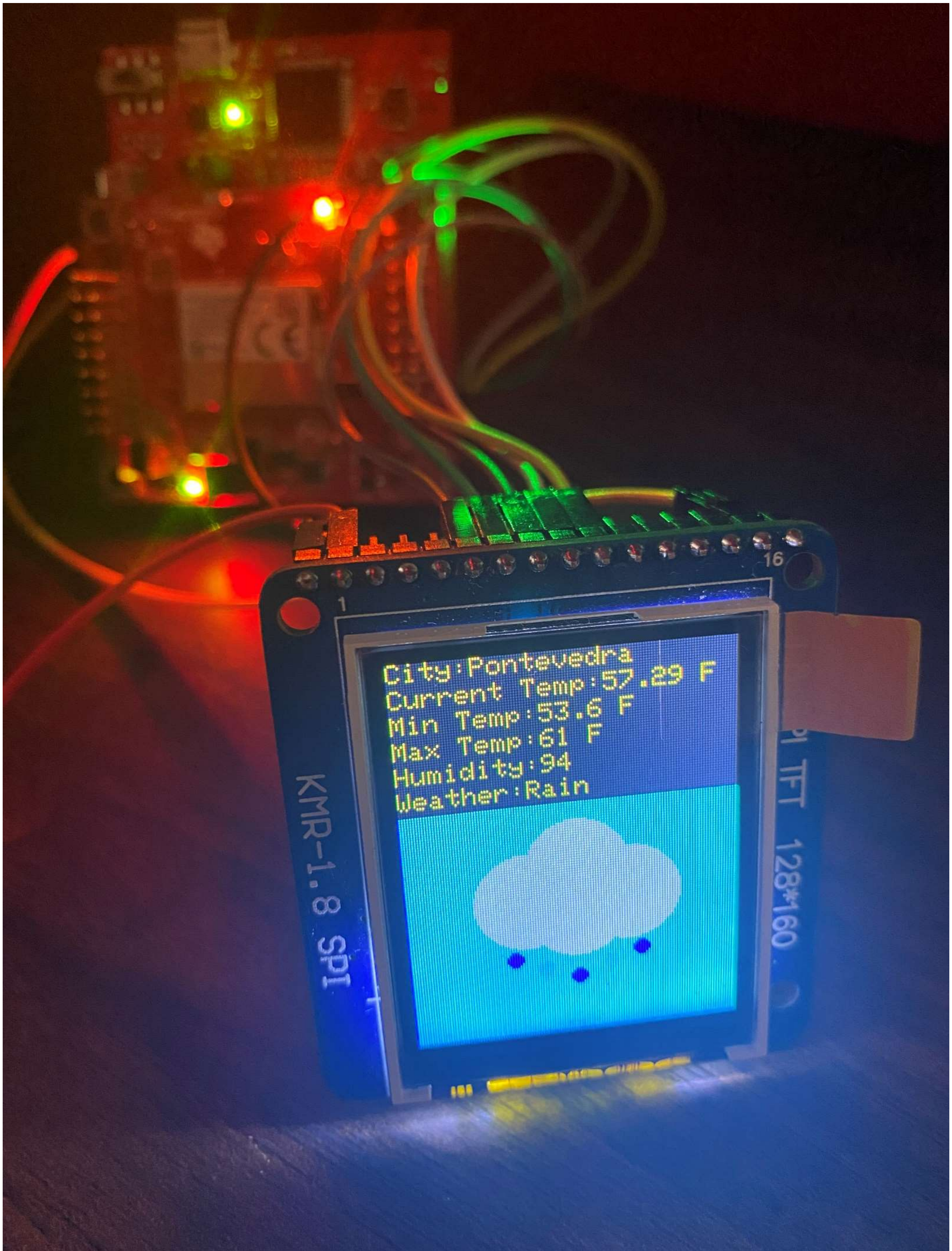


Photographs of Hardware System:





```
City:Los Angeles
Current Temp:61.2 F
Min Temp:57 F
Max Temp:64.99 F
Humidity:52
Weather:Clouds
```


City:Pontevedra
Current Temp:57.29 F
Min Temp:53.6 F
Max Temp:61 F
Humidity:94
Weather:Rain
KMR-1.8 SPI
SPI TFT 128*160
1
16

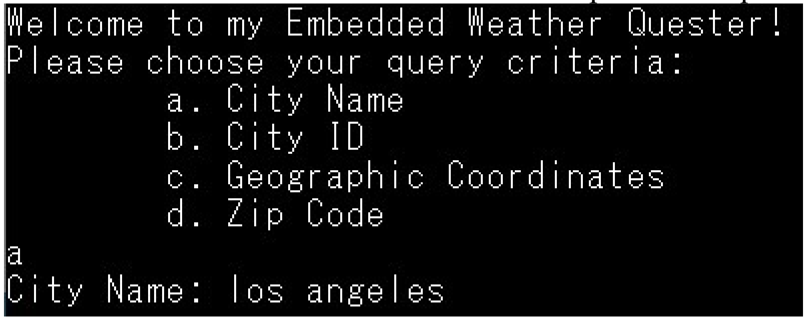
Software Design:

```
int main(void){int32_t retVal;  SlSecParams_t secParams;
      // String arrays for prompted manual input
      char string_city[20], string_city_id[20], string_lat[20], string_lon[20],
string_zip[20];

  char *pConfig = NULL; INT32 ASize = 0; SlSockAddrIn_t  Addr;
  initClk();        // PLL 50 MHz
  UART_Init();      // Send data to PC, 115200 bps
  LED_Init();       // Initialize LaunchPad I/O
      ST7735_InitR(INITR_REDTAB); // Initialization for ST7735R LCD
      SysTick_Init(); // Initialization for SysTick

  UARTprintf("Weather App\n");
  retVal = configureSimpleLinkToDefaultState(pConfig); // set policies
  if(retVal < 0)Crash(4000000);
  retVal = sl_Start(0, pConfig, 0);
  if((retVal < 0) || (ROLE_STA != retVal) ) Crash(8000000);
  secParams.Key = PASSKEY;
  secParams.KeyLen = strlen(PASSKEY);
  secParams.Type = SEC_TYPE; // OPEN, WPA, or WEP
  sl_WlanConnect(SSID_NAME, strlen(SSID_NAME), 0, &secParams, 0);
  while((0 == (g_Status&CONNECTED)) || (0 == (g_Status&IP_AQUIRED))){
    _SlNonOsMainLoopTask();
  }
  UARTprintf("Connected\n");
  while(1){
            ST7735_Main_Plot(); // Main static LCD Plot
            clear_flag = 0; // initially set clear sky flag off
            cloud_flag = 0; // intiially set cloud sky flag off
            rain_flag = 0;  // initially set rainy sky flag off

            // Menu options for the Serial Terminal
            UARTprintf("Welcome to my Embedded Weather Quester!\n");
            UARTprintf("Please choose your query criteria:\n");
            UARTprintf("      a. City Name\n");
            UARTprintf("      b. City ID\n");
            UARTprintf("      c. Geographic Coordinates\n");
            UARTprintf("      d. Zip Code\n");\
            // Awaiting for manual Char value from user
            switch(UARTgetc()){
                  case 'a': // Set up API command for City Name
                        UARTprintf("\nCity Name: ");
                        UARTgets(string_city,20);
                        strcat(TEMP, REQUEST_CITY);
                        strcat(TEMP, string_city);
                        strcat(TEMP, APPID);
                        break;
                  case 'b':// Set up API command for City ID
                        UARTprintf("\nCity ID: ");
                        UARTgets(string_city_id,20);
                        strcat(TEMP, REQUEST_CITY_ID);
                        strcat(TEMP, string_city_id);
                        strcat(TEMP, APPID);
                        break;
                  case 'c':// Set up API command for Geographical Coordinates
                        UARTprintf("\nLatitude: ");
                        UARTgets(string_lat,20);
                        UARTprintf("\nLongitude: ");
                        UARTgets(string_lon,20);
                        strcat(TEMP, REQUEST_GEO);
                        strcat(TEMP, string_lat); //lat
                        strcat(TEMP, "&lon=");
                        strcat(TEMP, string_lon); //lon
                        strcat(TEMP, APPID);
                        UARTprintf("\n");
                        UARTprintf(TEMP);
                        UARTprintf("\n");
                        break;
                case 'd':// Set up API command for Zip Code
                        UARTprintf("\nZIP Code: ");
                        UARTgets(string_zip,20);
                        strcat(TEMP, REQUEST_ZIP);
                        strcat(TEMP, string_zip);
                        strcat(TEMP, APPID);
                        UARTprintf("\n");
                        UARTprintf(TEMP);
                        UARTprintf("\n");
                        break;
        }// end switch

    strcpy(HostName,"api.openweathermap.org");
    retVal = sl_NetAppDnsGetHostByName(HostName,
             strlen(HostName),&DestinationIP, SL_AF_INET);
    if(retVal == 0){
      Addr.sin_family = SL_AF_INET;
      Addr.sin_port = sl_Htons(80);
      Addr.sin_addr.s_addr = sl_Htonl(DestinationIP);// IP to big endian
      ASize = sizeof(SlSockAddrIn_t);
      SockID = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, 0);
      if( SockID >= 0 ){
        retVal = sl_Connect(SockID, ( SlSockAddr_t *)&Addr, ASize);
      }
      if((SockID >= 0)&&(retVal >= 0)){
                        // Set requested API command
        strcpy(SendBuff,TEMP);
        sl_Send(SockID, SendBuff, strlen(SendBuff), 0);// Send the HTTP GET
        sl_Recv(SockID, Recvbuff, MAX_RECV_BUFF_SIZE, 0);// Receive response
        sl_Close(SockID);
        LED_GreenOn();
        UARTprintf("\r\n\r\n");
        UARTprintf(Recvbuff);  UARTprintf("\r\n");
      }
    }
        // Clear all manual input strings
        for(int j = 0; j<20; j++){
                string_city[j] = '\0';
                string_city_id[j] = '\0';
                string_lat[j] = '\0';
                string_lon[j] = '\0';
                string_zip[j] = '\0';
        }
        // Clear TEMP REQUEST string
        for(int j = 0; j<200; j++){
                TEMP[j] = '\0';
        }
        // Output all information on Serial Terminal and LCD
        Outputs();
        UARTprintf("\n");
        // While loop to be left hanging if SW1 or SW2 is not pressed
        // if SW1 or SW2 is pressed, the program will cycle all over again
        // for new manual inputs
    while(Board_Input()==0){
                if(clear_flag) ST7735_Clear_Plot();             // Display clear sky
animation
                else if(cloud_flag) ST7735_Cloud_Plot();// Display cloud sky animation
```

```
                  else if(rain_flag) ST7735_Rain_Plot();    // Display rain sky animation
            }; // wait for touch
    LED_GreenOff();
  }
}
```
NOTE: Only Main function is showed. The rest of the code is far too long. Check code attached.

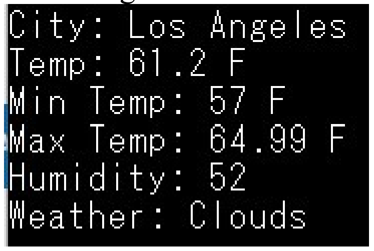
Requirements:

1. Build an embedded system with Wifi capability. Include laptop serial terminal and color LCD.
   - Check images above.
2. A user interface for the serial terminal to accept user's request:



```
Welcome to my Embedded Weather Quester!
Please choose your query criteria:
        a. City Name
        b. City ID
        c. Geographic Coordinates
        d. Zip Code
a
City Name: los angeles
```

3. Following information are required to display on serial terminal:



```
City: Los Angeles
Temp: 61.2 F
Min Temp: 57 F
Max Temp: 64.99 F
Humidity: 52
Weather: Clouds
```

4. Display information on color LCD:
   - Check images above.
5. Animation for weather conditions: sunrise (clear sky), cloudy (cloudy sky), and rainy (rainy sky).
   - Check images above.
   - Check animations in attached videos.

Conclusion:

This project required a lot of time and research to accomplish. The CC3100 is very useful, and I am very appreciative for having a project that uses this component because it may come of use in the senior project courses. Overall, this project is very fulfilling after it has been accomplished, and it seems like a great ending project for my final embedded system course. The project was very tedious, and I managed to get lost a lot in the code since it was so massive, but it was still a lot of fun since it helped me understand a bit more about APIs, IoT, and gave a much more broader idea on how to communicate between two devices.