Assignment: Final Project Writeup
Course: ECE523
Authors: Aryaman Ghura, Kavya Manchanda, Binh Minh Nguyen

Problem

The final project tasks us with creating a local weather station using the Thingy52 and an ePaper display. It requires three contexts, each using the past hour's collected data to display different forms of information. The below sections explain our method of collecting data and displaying the same in the three contexts.

Data Collection

The data is collected every WEATHER_MILLIS time units as defined for the variable.

```
103    //update intervals for ePaper display and sensor data
104    #define EPAPER_MILLIS 600000        // update every 10 min
105    #define WEATHER_MILLIS 600000       // update every 10 min
106    #define BUFFERSIZE 7                // number of values collected each hour
```

In the current code, we are updating the data every 10 minutes, and updating the display every 15 minutes. Since the buffer needs to collect data for every hour, the buffer size is set to 7 to every the data every 10 minutes until current time for the hour. We justified recording data every 10 minutes, since weather does not change very quickly, and any significant change would only be observed in 10 minutes time. Similarly, the ePaper display updates data every 15 minutes since we considered 4 updates per hour is both enough for the user given weather does not immediately change, and saves power consumption since updating the display is power inefficient and computationally expensive.

To record and calculate the required data for each context, we call the calculate_statistics() function every WEATHER_MILLIS time units in the following section of main.c:

```
669  static void weather_timeout_handler(void * p_context)
670  {
671      NRF_LOG_INFO("Getting new weather readings\r\n")
672
673      //tell the sensors to sample
674      drv_humidity_sample();
675      drv_pressure_sample();
676
677      if (first_hr_values_done == 0){
678          calculate_statistics(loop_i);          // calling calculate_statistics() with range = loop_i if first hr being recorded
679      }
680      else{
681          calculate_statistics(BUFFERSIZE-1);     // calling calculate_statistics() with range = BUFFERSIZE-1 if first hr already recorded
682      }
683
684      //get call back later, debug output from m_environment.c shows new values
685
686  }
```

```
static void calculate_statistics(int range){

    if (first_record_done == 0){                // if first recording reset, reset variables
        first_record_done = 1;
        loop_i = 0;
        range = 0;
    }

    temperature_arr[loop_i] = temperature;      // add current temperature to array
    humidity_arr[loop_i] = humidity;            // add current humidity to array
    pressure_arr[loop_i] = pressure;            // add current pressure to array

    avg_temperature = avg_humidity = avg_pressure = 0.0;    // initialize avg temp humidity, pressure
    for(int i=0; i<=range; i++){                            // calculate avg temp humidity, pressure till latest updated value
        avg_temperature += temperature_arr[i];
        avg_humidity += humidity_arr[i];
        avg_pressure += pressure_arr[i];
    }
    avg_temperature = avg_temperature/(range+1);
    avg_humidity = avg_humidity/(range+1);
    avg_pressure = avg_pressure/(range+1);

    std_temperature = std_humidity = std_pressure = 0.0;    // initialize std temp humidity, pressure
    for(int i=0; i<=range; i++){                            // calculate std temp humidity, pressure till latest updated value
        std_temperature += pow((temperature_arr[i] - avg_temperature), 2)/(range+1);
        std_humidity += pow((humidity_arr[i] - avg_humidity), 2)/(range+1);
        std_pressure += pow((pressure_arr[i] - avg_pressure), 2)/(range+1);
    }
    std_temperature = sqrt(std_temperature);
    std_humidity = sqrt(std_humidity);
    std_pressure = sqrt(std_pressure);

    float avg_rate_of_change = 0.0;         // finding the average rate of pressure change for context3 weather prediction
    int count_avg = 0;                      // variable to count number of rates considered
    for (int i=loop_i; i>0; i--) {          // looping through every variable
        avg_rate_of_change += (pressure_arr[i]-pressure_arr[i-1])/2;
        count_avg++;
    }

    if (first_hr_values_done == 1){         // if first hour values recorded, buffer is full and thus consider all values in order
        for (int i = BUFFERSIZE-1; i>loop_i+1; i--) {
            avg_rate_of_change += (pressure_arr[i]-pressure_arr[i-1])/2;
            count_avg++;
        }
        avg_rate_of_change += (pressure_arr[0]-pressure_arr[BUFFERSIZE-1])/2;   // considering change between last value of last hr and first value of current hr
        count_avg++;
    }


    if (count_avg==0){                      // case for first value when data not enough
        strcpy(barometer_pred, "Not enough data for Prediction");
    }

    else{                                   // assessing barometer prediction based on avg_rate_of_change and current pressure
        avg_rate_of_change = avg_rate_of_change/count_avg;
        if ((avg_rate_of_change < 0) && (pressure > 1023)){
            strcpy(barometer_pred, "Warmer and Cloudier Weather");
        }
        else if ((avg_rate_of_change >= 0) && (pressure > 1023)){
            strcpy(barometer_pred, "No Change in Weather");
        }
        else if ((avg_rate_of_change < 0) && (pressure <= 1023) && (pressure >= 1009)){
            strcpy(barometer_pred, "Precipitation");
        }
        else if ((avg_rate_of_change >= 0) && (pressure <= 1023) && (pressure >= 1009)){
            strcpy(barometer_pred, "Precipitation going away");
        }
        else if ((avg_rate_of_change < 0) && (pressure < 1009)){
            strcpy(barometer_pred, "Storm within 24 Hours");
        }
        else if ((avg_rate_of_change >= 0) && (pressure < 1009)){
            strcpy(barometer_pred, "Storm going away");
        }
    }

    if (first_record_done == -1){           // if first recording, retake first recording
        first_record_done = 0;
    }

    if (loop_i == (BUFFERSIZE-1)){          // if last buffer recording recorded, first hour complete
        first_hr_values_done = 1;
    }

    loop_i = (loop_i+1)%BUFFERSIZE;         // next buffer to be recorded
}
```

It uses multiple global variables as given below to store the past hours data, current data, barometer predictions, current recording, as well as various checks for special cases, particularly when the buffer is not completely full in the first hour. Furthermore, it may be noted that the code instructs the Thingy52 to re-record the first recorded value during start since we observed it to inaccurately record 0 for all data when starting up.

The global variables as mentioned above are given below:

```
120   int first_hr_values_done = 0;                                  // special case if full hrs values not yet collected
121   int first_record_done = -1;                                    // special case for the first recording
122   int loop_i = 0;                                                // latest updated value in the buffer
123   float temperature = 0.0;                                       // for updating current temperature
124   float avg_temperature = 0.0;                                   // for updating average temperature
125   long double std_temperature = 0.0;                             // for updating std dev of temperature
126   float temperature_arr[BUFFERSIZE];                             // array for hourly temperature readings
127   float humidity = 0.0;                                          // for updating current humidity
128   float avg_humidity = 0.0;                                      // for updating average humidity
129   long double std_humidity = 0.0;                                // for updating std dev of humidity
130   float humidity_arr[BUFFERSIZE];                                // array for hourly humidity readings
131   float pressure = 0.0;                                          // for updating current pressure
132   float avg_pressure = 0.0;                                      // for updating average pressure
133   long double std_pressure = 0.0;                                // for updating std dev of pressure
134   float pressure_arr[BUFFERSIZE];                                // array for hourly pressure readings
135   char barometer_pred[50] = "Not enough data for Prediction"; // barometer prediction for context 3
```

Context 0

For context 0, we are tasked with recording the "current data and historic data statistics for the last hour, each represented as a line of text." The code for the same is given below:

```
244   // Display temperature, humidity and pressure data and statistics
245   static void write_context_zero(void)
246   {
247       NRF_LOG_INFO(NRF_LOG_COLOR_CODE_GREEN"Context 0\r\n");
248       sprintf(display_string_buffer,"Display %i", display_context);
249       Paint_Init(image, EPD_get_bufwidth() * 8, EPD_get_bufheight());
250       SetRotate(ROTATE_0);
251       Paint_Clear(UNCOLORED);
252
253       DrawStringAt(8, 2, "Temperature (C)", &Font12, COLORED);                    // print cur, avg, std temperature in C
254       sprintf(display_string_buffer,"Cur: %.2f", temperature);
255       DrawStringAt(8, 17, (const char*)&display_string_buffer[0], &Font12, COLORED);
256       sprintf(display_string_buffer,"Avg: %.2f", avg_temperature);
257       DrawStringAt(8, 32, (const char*)&display_string_buffer[0], &Font12, COLORED);
258       sprintf(display_string_buffer,"StDev: %.2Lf", std_temperature);
259       DrawStringAt(8, 47, (const char*)&display_string_buffer[0], &Font12, COLORED);
260
261       DrawStringAt(8, 77, "Humidity (%)", &Font12, COLORED);                      // print cur, avg, std humidity in %
262       sprintf(display_string_buffer,"Cur: %.2f", humidity);
263       DrawStringAt(8, 92, (const char*)&display_string_buffer[0], &Font12, COLORED);
264       sprintf(display_string_buffer,"Avg: %.2f", avg_humidity);
265       DrawStringAt(8, 107, (const char*)&display_string_buffer[0], &Font12, COLORED);
266       sprintf(display_string_buffer,"StDev: %.2Lf", std_humidity);
267       DrawStringAt(8, 122, (const char*)&display_string_buffer[0], &Font12, COLORED);
268
269       DrawStringAt(8, 153, "Pressure (hPa)", &Font12, COLORED);                   // print cur, avg, std pressure in hPa
270       sprintf(display_string_buffer,"Cur: %.2f", pressure);
271       DrawStringAt(8, 168, (const char*)&display_string_buffer[0], &Font12, COLORED);
272       sprintf(display_string_buffer,"Avg: %.2f", avg_pressure);
273       DrawStringAt(8, 183, (const char*)&display_string_buffer[0], &Font12, COLORED);
274       sprintf(display_string_buffer,"StDev: %.2Lf", std_pressure);
275       DrawStringAt(8, 198, (const char*)&display_string_buffer[0], &Font12, COLORED);
276
277       Display(image);
278   }
```

It simply prints the data from the global variables that record current, average, and standard deviation for the past hour's temperature, humidity, and pressure. This data is constantly updated every WEATHER_MILLIS time units as mentioned above, when the calculate_statistics() function is called.

Context 1

For context 1, we are tasked with creating a bar graph that represents the last hour's data for temperature. The code for the same is given below:

```
// Display bar graph for the last hour's temperature data
static void write_context_one(void)
{
    NRF_LOG_INFO(NRF_LOG_COLOR_CODE_GREEN"Context 1\r\n");
    sprintf(display_string_buffer,"Display %i", display_context);
    Paint_Init(image, EPD_get_bufwidth() * 8, EPD_get_bufheight());
    Paint_Clear(UNCOLORED);

    DrawStringAt(40, 3, "Temp(C)", &Font12, COLORED);                       // label y-axis as Temperature in C
    SetRotate(ROTATE_90);

    int x_axis_length = 190;
    int y_axis_length = 75;

    DrawStringAt(5, 10, "Past Hour Temperature vs Time", &Font12, COLORED);     // print graph title
    DrawVerticalLine(35, 25, y_axis_length, COLORED);                       // print y-axis
    DrawHorizontalLine(35, 100, x_axis_length, COLORED);                    // print x-axis
    DrawStringAt(25, 95, "0", &Font12, COLORED);                            // label y-axis range
    DrawStringAt(18, 60, "15", &Font12, COLORED);                           // label y-axis range
    DrawStringAt(18, 25, "30", &Font12, COLORED);                           // label y-axis range
    DrawStringAt(80, 117, "Time Past (min)", &Font12, COLORED);             // label x-axis as Time Past in minutes

    char *bar_bin_labels[] = {"0m", "10m", "20m", "30m", "40m", "50m", "60m"};  // x-axis bin labels
    int bar_width = (int)(190/BUFFERSIZE);                                  // find size for each bin bar

    int bin_no = 0;
    for (int i = (loop_i-1); i >=0; i--) {                                      // loop backwards from last recorded value to print past hr temps
        int bar_height = (int)((temperature_arr[i]/35)*y_axis_length);          // calculate bar height based on range
        int x1_start = (int)(35 + bar_width*bin_no + bar_width/3);
        int x2_end = (int)(35 + bar_width*(bin_no+1));
        DrawFilledRectangle(x1_start, 100, x2_end, 100-bar_height, COLORED);    // print bin

        DrawStringAt(x1_start, 104, bar_bin_labels[bin_no], &Font12, COLORED);  // print bin label
        bin_no = bin_no+1;
    }

    if (first_hr_values_done == 1){                                             // continue looping backwards from last recorded value for past hours values
        for (int i = BUFFERSIZE-1; i>=loop_i; i--) {
            int bar_height = (int)((temperature_arr[i]/35)*y_axis_length);      // calculate bar height based on range
            int x1_start = (int)(35 + bar_width*bin_no + bar_width/3);
            int x2_end = (int)(35 + bar_width*(bin_no+1));
            DrawFilledRectangle(x1_start, 100, x2_end, 100-bar_height, COLORED);    // print bin

            DrawStringAt(x1_start, 104, bar_bin_labels[bin_no], &Font12, COLORED);  // print bin label
            bin_no = bin_no+1;
        }
    }

    Display(image);
}
```

The code makes use of the various functions of the epdpaint.c file found in the source/drivers/waveshare_epd_2in3_v3/ directory. The various static values used as coordinates have been derived after testing on the ePaper display to accurately print data. It may be noted that the write_context_one() updates every bin to accurately represent it's label. For the first hour, it only displays bins for the past minutes that have been recorded.

Context 2

For context 2, we are tasked with calculating storm predictions for the next 24 hours. We do this using the instructions given on the provided site: https://education.seattlepi.com/effects-atmospheric-pressure-pump-efficiency-4657.html. The given Inch of Mercury (inHg) threshold for different predictions given in this link is converted to hectoPascal (hPa) for correct comparison with the collected data using the conversions provided on this site: https://wiki.ivao.aero/en/home/training/documentation/Pressure_conversion_table. It may be noted that these predictions are made in the calculate_statistics() function as given above and the prediction is updated onto the barometer_pred global variable. These predictions are

made based on the past hours data, and thus no new buffer is required. For the first hour, the prediction is made based on the available data. Code for context 2 is given below:

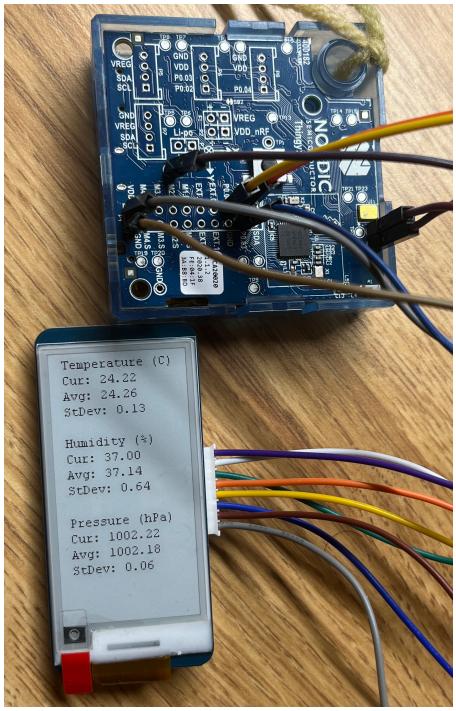
```
// Display barometric data
static void write_context_two(void)
{
    NRF_LOG_INFO(NRF_LOG_COLOR_CODE_GREEN"Context 2\r\n");
    sprintf(display_string_buffer,"Display %i", display_context);
    Paint_Init(image, EPD_get_bufwidth() * 8, EPD_get_bufheight());
    Paint_Clear(UNCOLORED);
    SetRotate(ROTATE_90);

    DrawStringAt(18, 25, "Barometer Prediction:", &Font12, COLORED);
    DrawStringAt(18, 50, barometer_pred, &Font12, COLORED);                // print barometer prediction
    Display(image);
}
```

Results

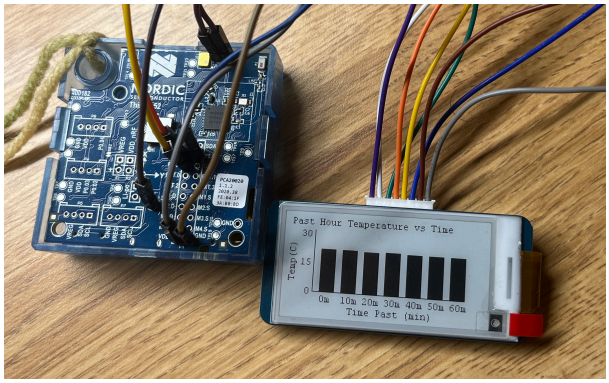
The resulting ePaper display values are given as follows: It may be noted EPAPER_MILLIS was changed to 30 seconds and WEATHER_MILLIS was changed to 5 seconds as requested in the project description file. This would not be an ideal case in reality since weather is highly unlikely to significantly change in 5 seconds and its going to be both computationally expensive and energy-wise inefficient for Thingy52 to be recording values and updating the ePaper display to quickly.

Please note that we do not change the Buffer Size in this demo to hold the hour's data since the buffer would have a size of 720 which is extremely memory inefficient for the Thingy52 and the ePaper display would have the bars extremely crammed and indistinguishable due to size restrictions.

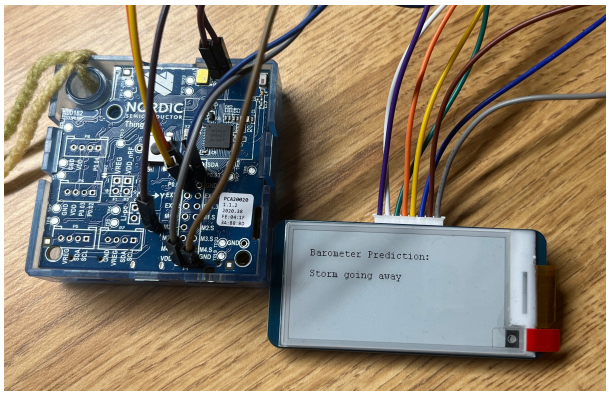
Context 0:



Context 1:



Context 2:



The above predictions are not very accurate since the average pressure calculated indoors was taken at ~1002hPa which according to the provided website predicts there to be a storm. We have not changed the threshold for an indoor environment from the one given on the website since the project description file only requests a proof-of-concept for the storm prediction.

A demonstration of context 1 with EPAPER_MILLIS = 15 min and WEATHER_MILLIS = 10 min is also given below to show fewer bins being displayed in the first hour due to limited data: