**ELEC 291 Section 20C**

Lab 4

L2C

Team 2A

*Student name Student number Contribution percentage*

Andy Ruan 36863141 33.3%

Kevin Wong 32105132 33.3%

Clarence Su 36387132 33.3%

Contribution Summary:

Andy Ruan worked on the first iterations of the circuit and Fritzing schematics. He coded for debouncing, and serial output logic on the Arduino. He contributed to receiving and processing the data in Matlab as a strip chart.

Kevin Wong worked on the code to take in sensor data in Processing and displaying it like a strip chart.

Clarence Su finalized circuit and Fritzing schematic. He also contributed to and finalized the Arduino and Matlab code.

**Introduction and motivations**

This report outlines the process in which Team 2A used to complete lab 4 of ELEC 291-20C. It is organized in the following manner: First, it will describe the lab and its sub-components. Then next will conclude on lesson learned from the lab. In addition, four appendices are included providing a Fritzing breadboard schematic of the circuit used and all the code used during the lab respectively.

The objective of this lab was to read sensor values from multiple types of sensors connected to the Arduino and output this data through the serial. In addition, this data is analyzed and displayed through both MATLAB and Processing.

**Lab Description**

Lab 4 was divided into two parts with part two subdivided by IDE:

1. A simple weather monitoring system
2. Interfacing Arduino with MATLAB and Processing
   1. Processing
   2. MATLAB

Successful completion of both parts resulted in a circuit consisting of an Arduino, a push button switch, a DHT11 humidity and temperature sensor, a LM35 analog temperature sensor, and a photocell. The ambient environment around the circuit can then be monitored through three methods of interface: serial, Processing, and MATLAB.

In part one, our team only encountered minor issues while wiring such as reversed pins on the LM35, DHT11 to analog instead of digital, and loose connections to the Arduino. While the DHT11 datasheet was poor, the lab handout was enough for us to figure out the proper circuit for functionality. We did however learn to appreciate proper datasheets more after stumbling through the DHT11 sheet. Overall, each circuit component was quite simple.

On the software side, we had already had prior experience with debouncing from the matrix keypad in the previous lab. The Arduino program was mostly a combination of this and similar code from lab one. Our team decided against rewriting the DHT library as we thought that it was not time efficient use of time.

A problem the team had was with the switch. The switch would not change to a different state when the button was pressed; therefore, values were not displaying on the Serial Monitor properly. To fix the problem, the team added a debounce time similar to the numeric keypad in Lab 3.

In part two, the team wrote code to graph data from the various sensors in Processing. A design consideration was either plotting all the graphs or simply plotting one graph. The team decided to plot all graphs on one screen with different colors. However, the photocell’s data had much bigger values compared to all other sensor data so the team decided to display the data in decalux.

A bug that was originally left unnoticed in the Arduino code was that when we outputted the photocell reading which was the last value to print, we decided to save space by using println(photocell) instead of printing the photocell value and then printing a ‘\n’ character. The resulted in an offset to the next line’s values being printed and resulted in our Processing code being unable to read the last value of every line, the photocell reading.

A problem with the Processing was graphing in real time. The graph generated by drawing lines in Processing was working but it lacked user-friendly interface. Therefore, the team took a few hours to try and implement a graph with a better user interface and more informative but failed. As a result, the team decided to enhance the working line graph by adding X and Y axes.

In MATLAB, our team started by studying the example code given from StripChart.m and the code to read one value from a serial input. We understood immediately that we would require a loop to continuously read values from the Arduino’s serial output and then update the plot used in the strip chart. However, for an easier time debugging, the team started by attempting to read display the values read from the serial to the MATLAB console. Our console quickly filled up with numbers; some numbers were valid lines of data while other time they were just garbage. After altering the code to also display a count, we realized that the loop was reading data too quickly for the amount of data that the Arduino was providing. To remedy this, the team put in a conditional statement inside the loop to continue to read data until it was valid. Valid data would always be in the form of a 1x4 matrix so we discarded all other matrix sizes. Following successful data parsing, we decided to graph the photocell readings with the strip chart using the example in StripChart.m as a guide. It can however be shown that we read all the data and can easily edit the code to graph a different value. Due to other commitments, our team decided not to implement any thing more complex than the basic strip chart. We decided to graph the photocell values as it was relatively easier to manipulate its value compared to the next easiest option which was breathing directly onto the DHT11. After noticing some stalling with the strip chart due to doing nothing while there was no valid data, our team decided that on the final iteration of the MATLAB code, we would update the graph with the previous reading if the data was not valid to preserve responsiveness.

**Conclusions**

In conclusion, this lab allowed students to learn about Serial events on Arduino. Students could extract data from Arduino sensors and use external software such as MATLAB and Processing to graph the data points for the user. Serial events make projects more interesting due to all the existing software tools to enhance projects. For instance, students could have used Python and all of its libraries to analyze the data and graph as well.

The team learned that proper documentation simplifies tasks. The team spent a lot of time trying to understand the DHT11; that time could have been spent on other issues such as working on the report or Processing. The team also became more efficient in reading documentation. Learning MATLAB and Processing for the first time under time constraints, it was critical to filter out unnecessary details about coding syntax and focus on aspects that helped the project.

**References**

[1] http://arduining.com/2013/08/05/arduino-and-processing-graph-example

**Appendix I**

Below is a Fritzing schematic of a circuit similar to the one used in lab four. Note that although the pin assignments match the actual circuit, the exact sensor placement may be different to provide for a better view in the Fritzing.

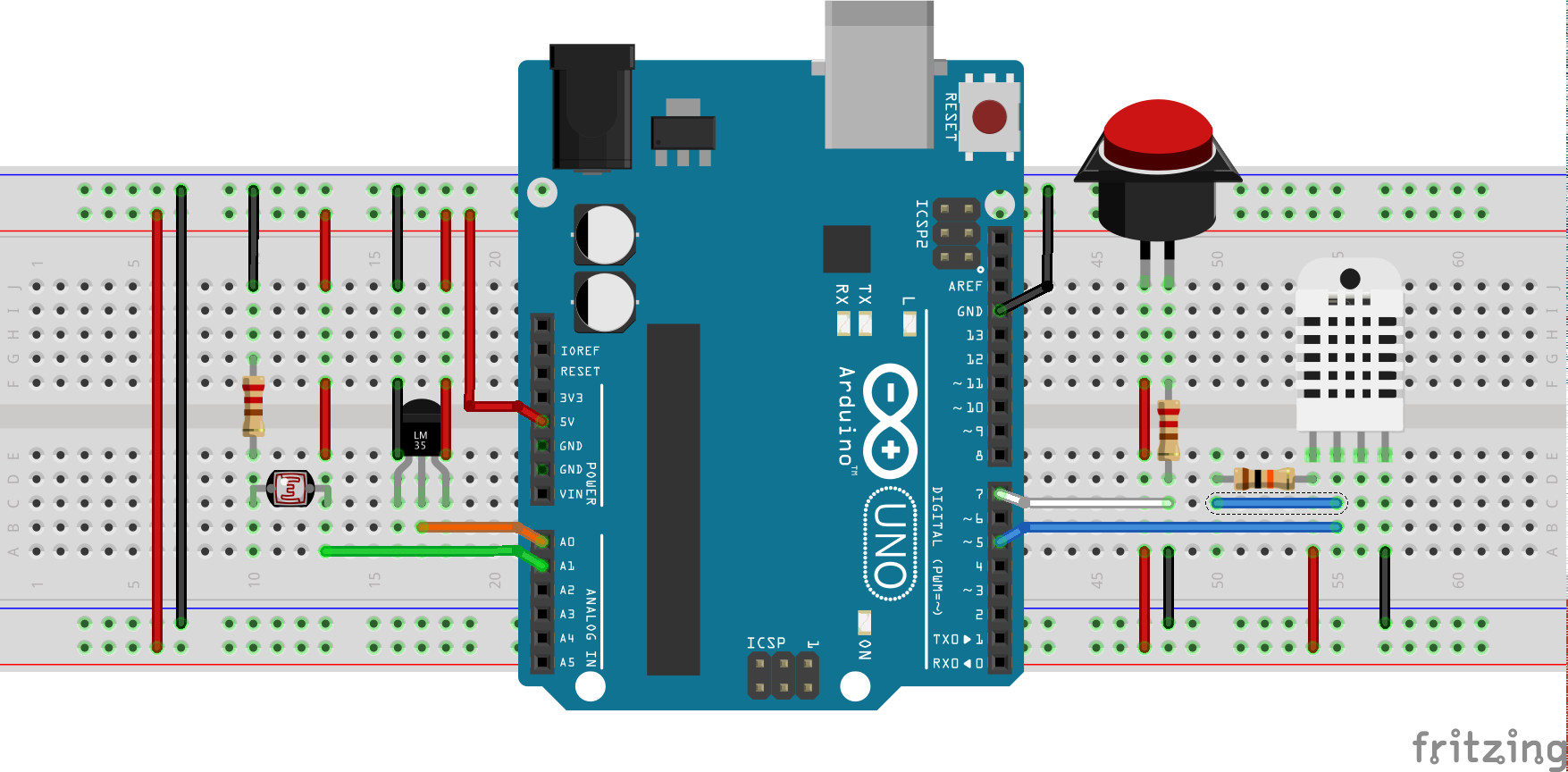


Figure 1: Fritzing schematic of circuit used in lab

**Appendix II**

The following Arduino code was used for Lab 4:

#include <DHT.h>

// Pin assignments

const int DHTPIN = 5;

const int DHTTYPE = DHT11;

const int TEMPPIN = A0;

const int PHOTOPIN = A1;

const int BUTTONPIN = 7;

const int DEBOUNCEDELAY = 1000;

// Global variables

int ButtonState;

int prevButtonState = LOW;

unsigned long timeSinceDebounce=0;

int state;

int readingDataTime =1000;

const int carlibrateValue = 15; //calculated from the data

DHT dht(DHTPIN,DHTTYPE); // initialize the DHT

void setup() {

Serial.begin(9600);

pinMode(BUTTONPIN, INPUT);

dht.begin();

}

void loop() {

// For debouncing

int buttonIn = digitalRead(BUTTONPIN);

if( buttonIn != prevButtonState ) { // update the timeSinceDebounce if push button is fluctuating due to noise

timeSinceDebounce = millis();

}

if( (millis() - timeSinceDebounce) > DEBOUNCEDELAY ) { // if button has been in the same state for long enough

if( buttonIn != ButtonState ) { // and button is in a different state from the previous actual button state

ButtonState = buttonIn; //save the buttonIn to ButtonState

delay(5);

//switch the state, if state is 1, change it to 0, if 0 change it to 1

if(state == 1){

state=0;

timeSinceDebounce = millis(); //update the debounce time

}

else{

state=1;

timeSinceDebounce = millis();

}

}

}

// // Read data from the sensors

int dhth = dht.readHumidity();

int dhtt = dht.readTemperature();

int lm35 = analogRead(TEMPPIN);

int light = analogRead(PHOTOPIN);

int calibratedLm35 = lm35 - carlibrateValue; //calibrate the data using a statistic difference

// Task 1: Output the raw data for analysis by processing/matlab

if( state == 0 ) {

Serial.print(dhth);

Serial.print(",");

Serial.print(dhtt);

Serial.print(",");

Serial.print(calibratedLm35);

Serial.print(",");

Serial.print(light);

Serial.print(",");

Serial.print(0);

Serial.print("\n");

delay(readingDataTime);

}

else if(state ==1 ) {

// Task 2: Output the data to the serial monitor with captions and units

Serial.print("DHT Humidity: ");

Serial.print(dhth);

Serial.print(" RH");

Serial.print("\t");

//partial pressure of water vapor to the equilibrium vapor pressure of water at the same temperature

Serial.print("DHT Temperature: ");

Serial.print(dhtt);

Serial.print(" degree");

Serial.print("\t");

Serial.print("LM35 Temperature: ");

Serial.print(calibratedLm35);

Serial.print(" degree");

Serial.print("\t");

Serial.print("Photocell: ");

Serial.print(light);

Serial.print(" lx");

Serial.print("\n");

delay(readingDataTime);

}

prevButtonState = buttonIn; // store the previous button state for the next loop

}

**Appendix III**

The following MATLAB code was used for Lab 4:

% Set up serial communications

s1 = serial('COM3', 'BaudRate', 9600, 'Parity', 'none', 'DataBits', 8, 'StopBits', 1, 'FlowControl', 'none');

fopen(s1);

x=1:1000; % initiate the x-axis values

y = zeros(1000, 1); % initiate the y-axis values

ylim([0 1000]); % fix the y-axis

hLine = plot(x,y);

ylabel({'Light Intensity', '(in lx)'}); % label the y-axis

StripChart('Initialize', gca); % initiate the strip chart

for i = 1:100 % run loop 100 times

val = fscanf(s1); % get a line of data from the serial

results = sscanf(val, '%f,%f,%f,%f', [1,4]); % parse line for values

[r,c] = size(results);

if c ~= 4 % check that data is valid (1x4 matrix)

y(1:i) = prevVal; % if not valid, update strip chart with the previous reading (to keep the chart responsive)

StripChart('Update', hLine, y(1:i));

xlabel({'Time', '(in 500ms)'}); % continuously update the x-axis label because StripChart overwrites

else

disp(results); % display the data in the MATLAB console, if valid data

y(1:i) = results(4); % write the photocell value into y-matrix for the graph

prevVal = results(4); % store the most recent reading

StripChart('Update', hLine, y(1:i)); % update the graph with new y-data

xlabel({'Time', '(in 500ms)'}); % continuously update the x-axis label because StripChart overwrites

end

end

fclose(s1);

**Appendix IV**

The following Processing code was used for Lab 4:

import processing.serial.\*;

Serial myPort; // The serial port

int inByte; // Variable for converting serial port string to float for DHT Humidity

int tempDHTByte; // Variable for DHT Temperature

int tempLM35Byte; // Variable for LM35 Temperature

int photocellByte; // Variable for Photocell Light Data

//Last height and x position for for DHT Humidity.

int xPos = 1; // horizontal position of the graph

int lastxPos=1;

int lastheight=0;

//Last height and x position for tempDHTByte

int xPos2 = 1;

int lastxPos2=1;

int lastheight2=0;

//Last height and x position for tempLM35Byte

int xPos3 = 1;

int lastxPos3=1;

int lastheight3=0;

//Last height and x position for photocellByte

int xPos4 = 1;

int lastxPos4=1;

int lastheight4=0;

PFont font;

void setup () {

// set the window size:

size(600,250);

smooth();

//Loading font for text

font = loadFont("AppleSymbols-24.vlw");

textFont(font);

// List all the available serial ports

printArray(Serial.list());

// Select port for computer

myPort = new Serial(this, Serial.list()[1], 9600);

// A serialEvent() is generated when a newline character is received :

myPort.bufferUntil('\n');

background(0); // set inital background:

}

void draw () {

// Writing text for Y-axis

textFont(font,20);

float x = 20;

float y = 150;

textAlign(CENTER,BOTTOM);

pushMatrix();

translate(x,y);

rotate(-HALF\_PI);

text("DATA",50,5);

popMatrix();

// Y-Axis Units

textFont(font,12); // change font to 12

text("100", 15, height - 100);

text("50", 15, height - 48);

text("0", 13, height -4);

//Change font back to size 20

textFont(font,24);

//Writing text for X-axis

text("Time (s)", 200, height-4);

text("Sensor Data vs Time", 300, 50); //Text for title

fill(127,34,255); //Fill in color for legend

text("DHT Humidity in RH" , 300, 70);

fill(255,50,245); //Fill in color for legend

text("DHT Temperature in Degrees Celsius" , 300, 90);

fill(127,250,55); //Fill in color for legend

text("LM35 Temperature in Degrees Celsius" , 300, 110);

fill(107,20,45); //Fill in color for legend

text("Photocell in dalx" , 300, 130);

fill(100,123,32); //Fill in color for legend

strokeWeight(4); //stroke wider

stroke(200,34,255); // stroke color

line(2, 0, 2, height); // Draws vertical line for graph

line(0, height-2, width, height-2); // Draws horizontal line for graph

// DHT Humidity graph

stroke(127,34,255); //stroke color

line(lastxPos, lastheight, xPos, height - inByte);

lastxPos= xPos; // Determines last x position for last line

lastheight= int(height-inByte); // Determines last height of last line

// DHT Temperature graph

stroke(255,50,245); //stroke color

line(lastxPos2, lastheight2, xPos2, height - tempDHTByte);

lastxPos2= xPos2; // Determines last x position for last line

lastheight2= int(height-tempDHTByte); // Determines last height of last line

// LM35 Temperature graph

stroke(127,250,55); //stroke color

line(lastxPos3, lastheight3, xPos3, height - tempLM35Byte);

lastxPos3= xPos3; // Determines last x position for last line

lastheight3= int(height-tempLM35Byte); // Determines last height of last line

//Photocell graph

stroke(107,20,45); //stroke color

line(lastxPos4, lastheight4, xPos4, height - photocellByte);

lastxPos4= xPos4; // Determines last x position for last line

lastheight4= int(height-photocellByte); // Determines last height of last line

// at the edge of the window, go back to the beginning:

if (xPos >= width || xPos2 >= width || xPos3 >= width || xPos4 >= width ) {

xPos = 0;

lastxPos= 0;

xPos2 = 0;

lastxPos2= 0;

xPos3 = 0;

lastxPos3= 0;

xPos4 = 0;

lastxPos4 = 0;

background(0); //Clear the screen.

}

else {

// increment the horizontal position:

xPos++;

xPos2++;

xPos3++;

xPos4++;

}

}

void serialEvent (Serial myPort) {

// get the ASCII string:

//String inString = myPort.readStringUntil('\n');

String inString = myPort.readString();

String[] sensorData;

//inString = trim(inString);

if (inString != null) {

//inString = trim(inString); // trim off whitespaces.

sensorData = split(inString, ',');

inByte = Integer.parseInt(sensorData[0]); //DHT Humidity

tempDHTByte = Integer.parseInt(sensorData[1]); // Variable for DHT Temperature

tempLM35Byte = Integer.parseInt(sensorData[2]); // Variable for LM35 Temperature

photocellByte = Integer.parseInt(sensorData[3]); // Variable for Photocell Light Data

photocellByte = photocellByte / 10;

println("Humidity: " + str(inByte));

println("DHT Temperature: " + str(tempDHTByte));

println("LM Temperature: " + str(tempLM35Byte));

println("Photocell: " + str(photocellByte));

}

}