**Lab 3: Climate Control & Room Occupancy Prototype**

Eric Pires

John Lamanuzzi

Richard Harrison

ECE 388

Team 3

Lab Date: September 28, 2016

Due Date: October 12, 2016



**Table of Contents**

[Introduction 2](#_Toc463904971)

[Methods and Testing Procedures 3](#_Toc463904972)

[DHT22 Temperature and Humidity Sensor 3](#_Toc463904973)

[PIR Sensor 3](#_Toc463904974)

[LCD 4](#_Toc463904975)

[Push Button 5](#_Toc463904976)

[Relay 5](#_Toc463904977)

[Climate Control and Room Occupancy Prototype 6](#_Toc463904978)

[Results 7](#_Toc463904979)

[DHT22 Temperature and Humidity Sensor 7](#_Toc463904980)

[LCD 7](#_Toc463904981)

[Push Button 8](#_Toc463904982)

[Relay 8](#_Toc463904983)

[Climate Control and Room Occupancy Prototype 8](#_Toc463904984)

[Discussion 10](#_Toc463904985)

[Conclusion 10](#_Toc463904986)

[Reflection 10](#_Toc463904987)

[References 10](#_Toc463904988)

[Appendix 11](#_Toc463904989)

Abstract

For this laboratory experiment, the goal was to prototype the climate control and room occupancy system chosen by our group. Each device was tested and implemented onto the prototype. Once each device was tested, a testing procedure was followed to ensure the prototype responded to a given set of inputs.

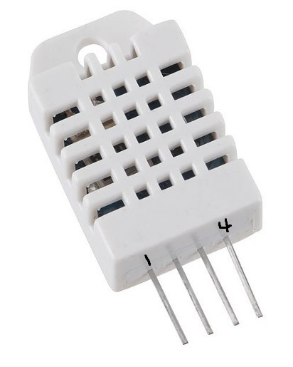
# Introduction

The objective of this lab was to develop a working prototype of a climate control and room occupancy system. The system must take in a user temperature input that is desired, and control the temperature accordingly. The system must also keep count of the number of people detected in the room and turn the lights on whenever there is motion detected. A flow chart of the system process can be found in Figure 1 of the Appendix. Testing procedures were developed for each component, some using previous lab experiment procedures. The following components were used in this experiment:

* Adafruit Metro Mini 328
* Etekcity MSR-R500 Digital Multimeter
* DHT22 Temperature and Humidity Sensor
* 1602A Character LCD
* PIR Sensor
* Pushbutton (2x)
* 12VDC Fan (2x)
* SRD-05VDC-SL-C Relay
* Breadboard and jumper cables

# Methods and Testing Procedures

## DHT22 Temperature and Humidity Sensor

The DHT22 sensor contains both a thermistor and humidity sensor to provide both temperature and humidity readings within a single data line. The sensor outputs a digital signal; therefore, it can be sent straight to any digital I/O pins. To test this sensor, the sample Arduino code provided by Adafruit Industries was used to output temperature and humidity readings onto the serial monitor. This was done with the following pin connections:

* Pin 1 to 5V on Mini
* Pin 2 to 7 on Mini
* 10kΩ Resistor from Pin 2 to Pin 1
* Pin 4 to GND on Mini

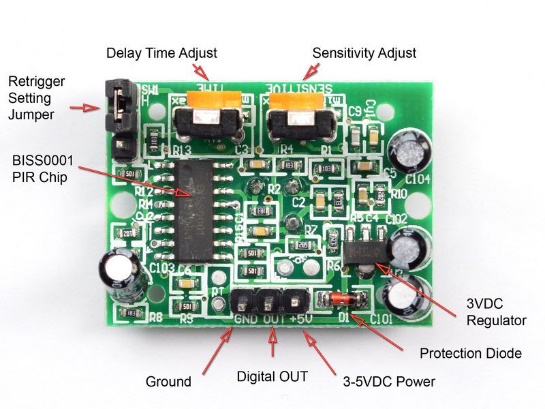
Pin 3 was ignored, as it is classified as a NULL pin on the datasheet. The example code was then uploaded to the Mega Mini. This code used the provided header file from Adafruit, which automatically calculates temperature into Celsius and humidity on a percentage scale. The code can be found in Figure 1A of the Appendix.

## PIR Sensor

The Passive Infrared Sensor tracks motion from up to seven meters. The lenses found around the sensor help amplify the range of the sensor to reach this distance. To test this sensor, a simple program was written to illuminate the on-board LED whenever motion was detected. This code can be found in Figure 2. This was done with the following pin connections

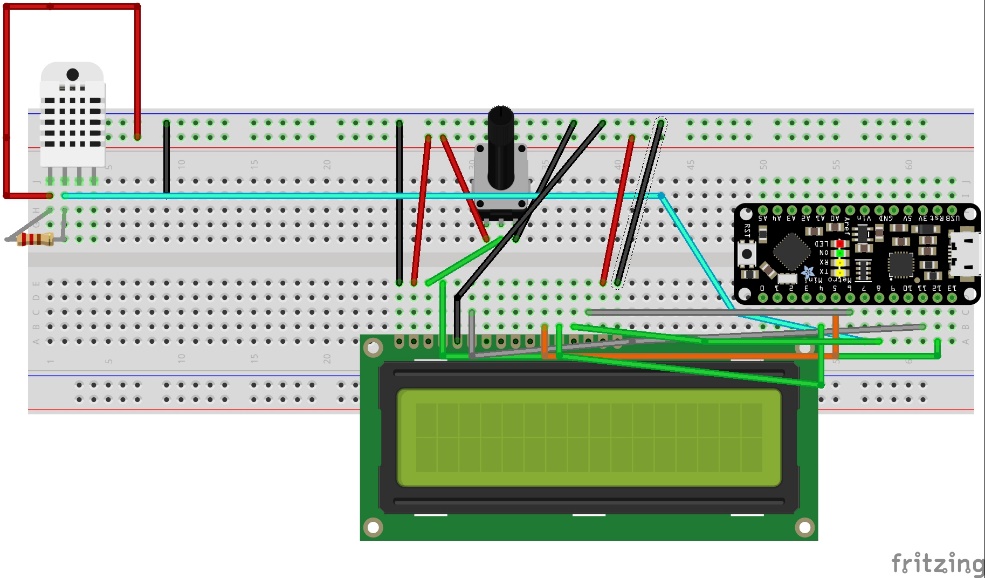
* +5V to 5V on Mini
* OUT to 10 on Mini
* GND to GND on Mini

Before uploading the code onto the Mini, the on-board potentiometers of the PIR sensor were adjusted to provide a minimum sensitivity and a five second output delay. This delay would keep the output high for that duration. In addition to this test, a simple counter was implemented to increment a set value whenever the PIR turned high. The final code for this procedure can be found in Figure 3.



## LCD

The 16x2 character LCD provides an alphanumeric display based on a specified output. To test this component, the DHT procedure was modified to print out on the LCD rather than the serial monitor. Using the Liquid Crystal header file, the LCD was initialized and was sent temperature and humidity readings. This code can be found in Figure 3A. The following pin connections were made to utilize the LCD.

* VCC to 5V on Mini
* VDD to GND
* VO to potentiometer wiper (10kΩ)
* RS to Pin 12
* RW to GND
* E to Pin 11
* D4 to Pin 5
* D5 to Pin 4
* D6 to Pin 8
* D7 to Pin 6
* A to 5V
* K to GND
* (See DHT procedure for pinout)

## Push Button

The final component to evaluate was a push button. When the button is pressed, the internal switch is turned on, allowing current to flow through the terminals. To test this simple component, a button counter program was written. The program would count how many times the button was pressed. This value was printed on the serial monitor. The following pin connections were made:

## Relay

The SRD-05VDC-SL-C relay allows a 5V signal to switch a high voltage signal on/off. To determine the functionality of this relay, a 9V battery was used to supply 9V to the relay. The Mini would send a 5V signal to turn on the relay and allow the 9V to flow. This signal would then drive the fans on, proving to be a better source of power than the Mini. The following pins were connected:

* COM to +9V
* NO to +V on fan
* GND on fan to common ground
* - on Relay to GND on Mini
* + to 5V
* S driven high/low with Mini

## Climate Control and Room Occupancy Prototype

Once all individual components were tested, they were applied to the prototype board. This design contained the same pinouts detailed in the previous procedures. The code from these procedures was organized and configured for the prototype build. The last two components that was included were 12VDC case fans. Since the relay was inoperable, the power pin was fed to an I/O pin on the mini. The analog pins were used since the other digital pins were cluttered with jumper cables. These fans will be fed through the digital I/O pins for the final design. A simple function was written to control each fan based on a comparison between the current and set temperatures. The fans were oriented in a push-pull configuration. When the set temperature was less than the current temperature, the push fan was activated. When the set temperature was greater, the pull fan was activated. The final code for the prototype can be found in Figure 3A.

# Results

## DHT22 Temperature and Humidity Sensor

When the test code was executed, the serial monitor provided the readings from the sensor. This output can be found in Figure 1B. To test the validity of these reading, a home temperature system was used as a comparison. These values were close within a few degrees Celsius. General conclusions were also made given that the sensor showed average room temperature readings. The sensor was confirmed to be functional and was imported to the prototype design.

PIR Sensor

When the code was executed and the sensor was given time to calibrate, motion was provided in its path. The circuit responded by illuminating the on-board LED for about five seconds. After this duration, the LED was driven low. After verifying the functionality of the PIR sensor, a simple counter was added that would increment a variable whenever the PIR detected motion. This test was executed, but the outcome was not desired. The counter would continuously increment on the serial monitor for the given duration delay. To adjust this outcome, the previous state and current state of the output was stored and compared to ensure that the counter variable would only increase once the output drove low. This was done in the outSent() function found in Figure 3. The code was executed and the counter properly incremented at the falling edge of the output signal. The sensor was confirmed to be functional and was imported to the prototype design.­­­

## LCD

When the code was executed, the LCD initially showed null values for the temperature and humidity readings readings. This is because the DHT22 requires two seconds to refresh the data output. Once two seconds passed, the temperature and humidity was displayed on the screen. These values were equivalent to the previous ones shown on the serial monitor. The LCD was confirmed to be functional and was imported to the prototype design.

## Push Button

When the code was first executed, the counter variable showed a similar reaction found in the PIR test. The variable was incrementing more times than the button was pressed. This bouncing issue was also resolved through comparing the previous and current states. When the button state was low (signifying the button was pressed), the button counter was changed and the previous state was saved. A fifty millisecond delay was applied to ensure that the device had sufficient time to debounce. Therefore, the counter would only increment once every time the button was triggered.

## Relay

When the input relay pin (S) was low, the normally open (NO) pin contained a 9V signal. When it was driven high, no change was observed. Using the multimeter, it was confirmed that the switch was not provided any current when the input was driven high. The voltage source was functional, however it seemed the relay was not triggering the switch. Based on this measured result, it is safe to assume that the given relay experienced a technical malfunction. This component was set aside and the fan was powered with a 5V source instead.

## Climate Control and Room Occupancy Prototype

The prototype was given a set of user inputs to test its response. Button response was immediate whenever a set temperature was configured. The case fans responded accordingly to the input data. There was no visible time lag between a change in temperature and the fan response. The PIR sensor also responded well in the system. There seemed to be a slight delay between motion and the output signal. This was expected, as many motion controlled lighting systems do indeed contain a small delay before turning the light on. A chart of the various testing procedures can be found below.

|  |  |
| --- | --- |
| **System Inputs** | **System Response** |
| Set temp exceeds current temp | A0 goes high and A1 goes low. Pull fan turns on |
| Temperatures are equal | A0 goes low, pull fan turns off. |
| Current temp exceeds set temp | A1 goes high and A0 goes low. Pull fan turns off, push fan turns on |
| Both input buttons are pressed | Set temp stays the same |
| Hand waved near PIR | Light turns on, off after 5s |
| Hand waved and input button is pressed | Light turns on and set temperature is changed simultaneously |

The highest priority system function is the user input. Therefore, the user should be able to immediately change the temperature using the push buttons. To compensate for this factor, the two second delay found in the temperature sensor reading was deleted. This delay had also affected the button response. Despite having no delay to refresh the temperature reading, the output of the DHT22 still showed valid readings throughout its run time. The tradeoff is that on boot, the system shows null values for a few seconds before displaying a valid reading.

# Discussion

# Conclusion

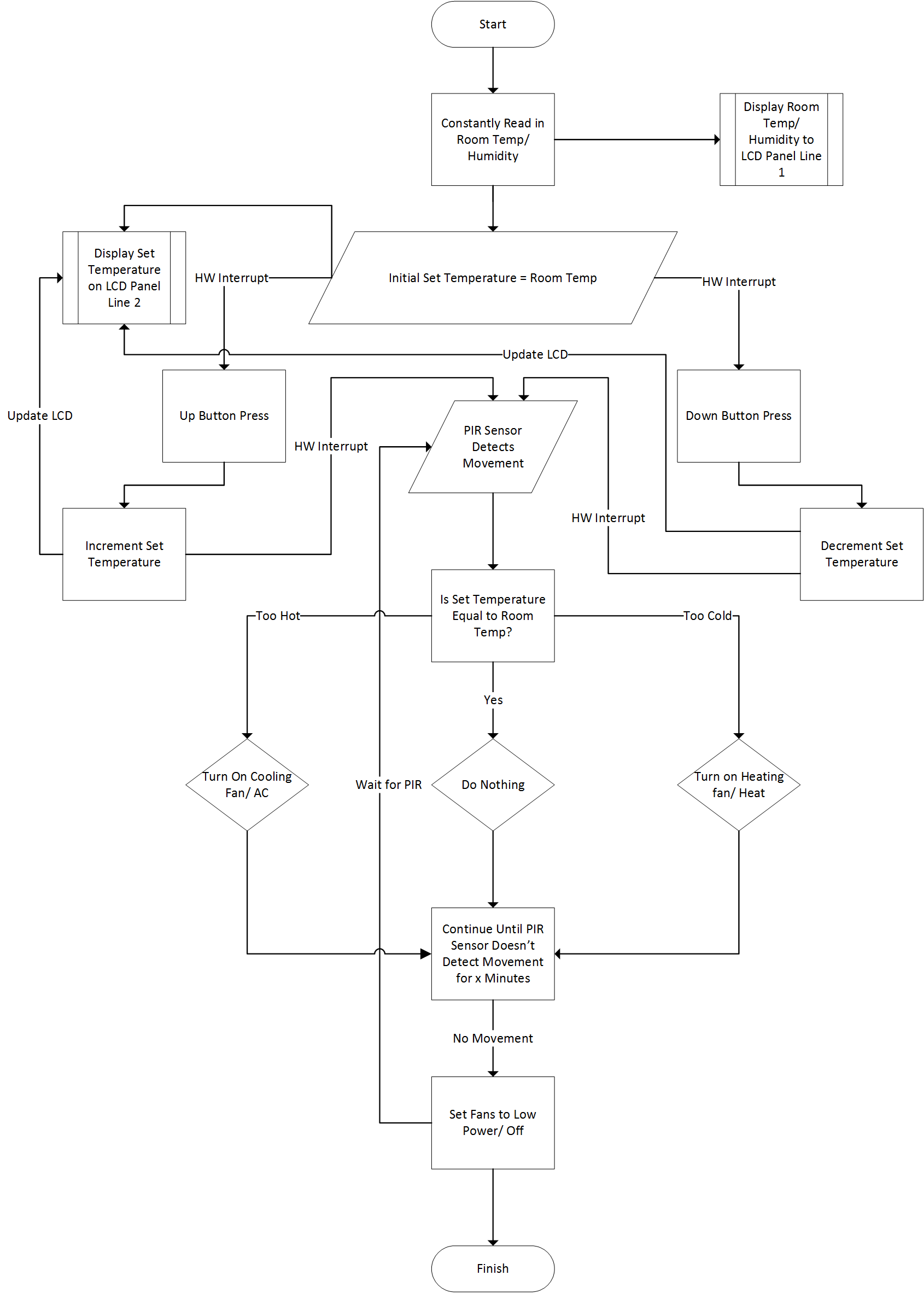
# Reflection

# References

Fortier, Paul. (2016). Laboratory #2. Handout.

# Appendix

*Figure 1: System Flow Chart*



*Figure 2A: DHT Test*

/\*

DHT sensor test

\*/

#include "DHT.h"

#define DHTTYPE DHT22

#define DHTPIN 7

DHT dht(DHTPIN, DHTTYPE);

void setup() {

Serial.begin(9600);

}

void loop() {

// Wait for valid temperature reading

delay(2000);

Serial.print("Temperature (C): ");

Serial.print(dht.readTemperature());

Serial.print("\t");

Serial.print("Humidity");

Serial.println(dht.readHumidity());

}

*Figure 2B: Serial Output (After 12 seconds)*

Temperature (C): 21.40 Humidity44.80

Temperature (C): 21.40 Humidity44.80

Temperature (C): 21.40 Humidity45.60

Temperature (C): 21.40 Humidity45.50

Temperature (C): 21.40 Humidity47.00

Temperature (C): 21.40 Humidity46.90

Temperature (C): 21.40 Humidity49.70

Temperature (C): 21.40 Humidity51.30

Temperature (C): 21.40 Humidity50.40

Temperature (C): 21.40 Humidity50.60

Temperature (C): 21.40 Humidity49.80

Temperature (C): 21.40 Humidity48.50

Temperature (C): 21.50 Humidity49.70

Temperature (C): 21.50 Humidity49.80

Temperature (C): 21.50 Humidity49.90

Temperature (C): 21.50 Humidity50.50

Temperature (C): 21.50 Humidity50.30

Temperature (C): 21.50 Humidity50.60

Temperature (C): 21.50 Humidity51.80

*Figure 3: PIR Test*

int led = 13;

int pir = 10;

int myVal = 0;

int lastState = LOW;

void setup() {

pinMode(led,OUTPUT);

pinMode(pir,INPUT);

Serial.begin(9600);

}

void loop() {

outSent();

if(digitalRead(pir) == HIGH)

digitalWrite(led,HIGH);

else

digitalWrite(led,LOW);

}

void outSent(){

// read the input pin:

int currentState = digitalRead(pir);

// compare the currentState to its previous state

if (currentState != lastState) {

// if the state has changed, increment the counter

if (currentState == LOW)

Serial.println(++myVval);

}

// save the current state as the last state,

lastState = currentState;

}

*Figure 4A: Prototype Code*

// include the library code:

#include <LiquidCrystal.h>

#include <DHT.h>

#define DHTTYPE DHT22

#define DHTPIN 7

DHT dht(DHTPIN, DHTTYPE);

/\*

\* LCD RS pin to digital pin 12

\* LCD Enable pin to digital pin 11

\* LCD D4 pin to digital pin 5

\* LCD D5 pin to digital pin 4

\* LCD D6 pin to digital pin 8

\* LCD D7 pin to digital pin 6

\* LCD R/W pin to GND

\* LCD VSS pin to ground

\* LCD VCC pin to 5V

\* 10K resistor:

\* ends to +5V and ground

\* wiper to LCD VO pin (pin 3)

\*/

// initialize the library with the numbers of the interface pins

LiquidCrystal lcd(12, 11, 5, 4, 8, 6);

const int led = 13;

const int pir = 10;

const int upButton = 3;

const int downButton = 2;

const int cool\_fan = A1;

const int heat\_fan = A0;

int current\_temp, set\_temp;

int lastUpState = HIGH;

int lastDownState = HIGH;

int lastPirState = LOW;

int roomCnt = 0;

void setup() {

// set up the LCD's number of columns and rows:

lcd.begin(16, 2);

lcd.print("TEMP SET HUM%");

Serial.begin(9600);

pinMode(pir,INPUT);

pinMode(upButton,INPUT\_PULLUP);

pinMode(downButton,INPUT\_PULLUP);

pinMode(cool\_fan,OUTPUT);

pinMode(heat\_fan,OUTPUT);

pinMode(led,OUTPUT);

}

void loop() {

current\_temp = dht.readTemperature();

temp\_RW();

fanControl();

lightControl();

}

void temp\_RW(){

//delay(2000);

upPress();

downPress();

lcd.setCursor(0,1);

lcd.print(current\_temp);

lcd.print("C");

lcd.print(" ");

if(set\_temp == 0)

set\_temp = current\_temp;

lcd.print(set\_temp);

lcd.print("C");

lcd.print(" ");

lcd.print(dht.readHumidity());

//delay(2000);

}

void upPress(){

// read the pushbutton input pin:

int buttonState = digitalRead(upButton);

// compare the buttonState to its previous state

if (buttonState != lastUpState) {

// if the state has changed, increment the counter

if (buttonState == LOW)

set\_temp++;

// Delay a little bit to avoid bouncing

delay(50);

}

// save the current state as the last state,

lastUpState = buttonState;

}

void downPress(){

// read the pushbutton input pin:

int buttonState = digitalRead(downButton);

// compare the buttonState to its previous state

if (buttonState != lastDownState) {

// if the state has changed, increment the counter

if (buttonState == LOW)

set\_temp--;

// Delay a little bit to avoid bouncing

delay(50);

}

// save the current state as the last state,

lastDownState = buttonState;

}

void fanControl()

{

// Push fan is activated, Pull is deactivated (Cools room)

if(current\_temp > set\_temp){

digitalWrite(heat\_fan,LOW);

digitalWrite(cool\_fan,HIGH);

}

// Pull fan is activated, Push is deactivated (Heats room)

else if(current\_temp < set\_temp){

digitalWrite(cool\_fan,LOW);

digitalWrite(heat\_fan,HIGH);

}

// Push and Pull fans deactivated (Temperature matches user input)

else{

digitalWrite(cool\_fan,LOW);

digitalWrite(heat\_fan,LOW);

}

}

void lightControl(){

pirCount();

if(digitalRead(pir) == HIGH)

digitalWrite(led,HIGH);

else

digitalWrite(led,LOW);

}

void pirCount(){

// read the out pin:

int currentState = digitalRead(pir);

// compare the currentState to its previous state

if (currentState != lastPirState) {

// if the state has changed, increment the counter

if (currentState == LOW)

Serial.println(++roomCnt);

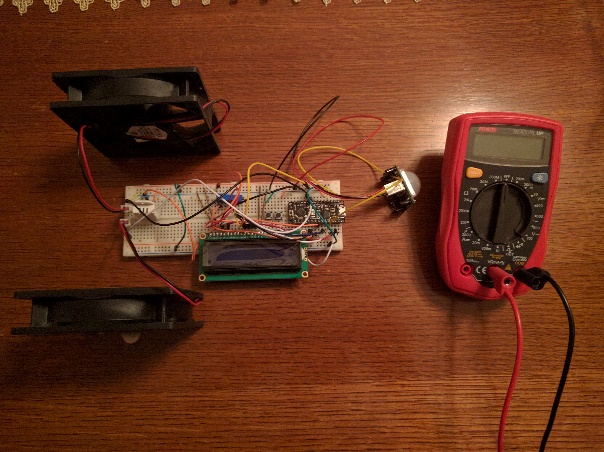
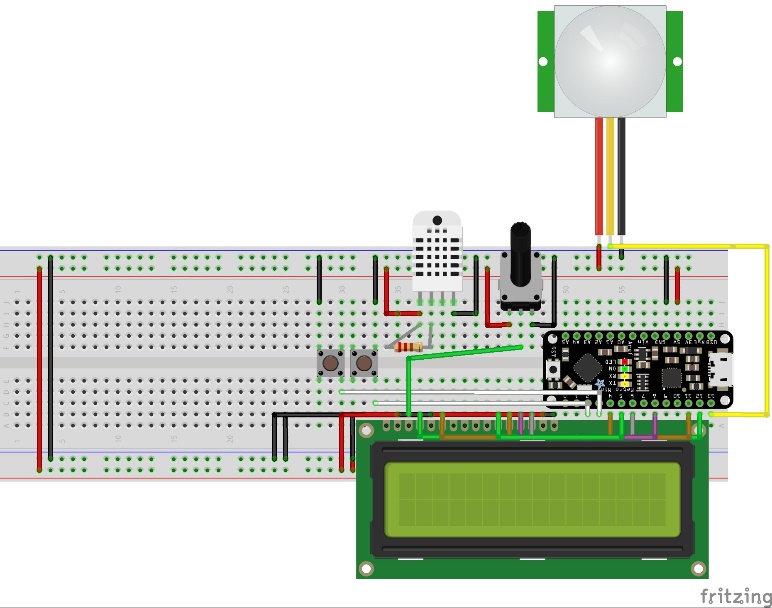
}

// save the current state as the last state,

lastPirState = currentState;

}

*Figure 4B & 4C: Prototype Schematic and Bench Setup*

**