# Description

The Temperature Sensor project is a MCU that measures the local temperature and logs data to a connected Host.

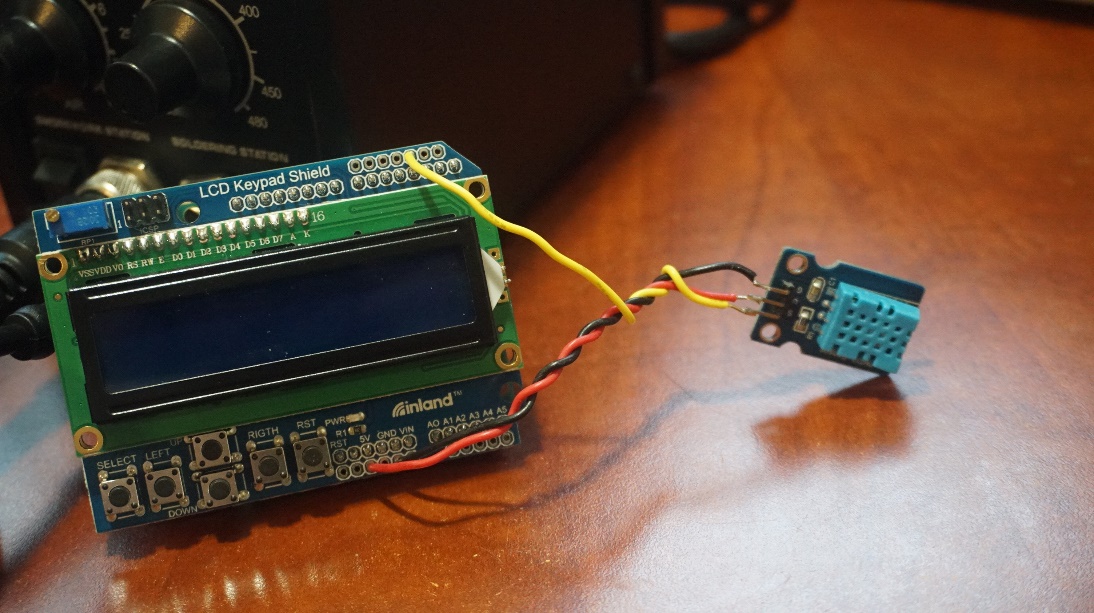


Figure – Temperature Sensor

# Requirements

The following requirements describes the functionality set forth by the Temperature Sensor project:

* The MCU shall read temperature from its local location.
* The MCU shall read temperature in degrees Fahrenheit.
* The MCU shall record temperature data in 10 second intervals.
* The MCU shall transmit the 10 second interval temperature data to the Host via Serial.
* The MCU shall implement a Round Robin scheduling algorithm with Interrupts.

# Design

The following sections describes the hardware, software, and configuration design decisions for the Temperature Sensor project.

## Hardware

The Arduino UNO was used as the MCU for this project. In addition, the following materials were used:

* DHT11 Temperature and Humidity Sensor
* 3x Wires (approximately 5” in length)

Optionally, an LCD may be connected to this project. I opted to use a 1602 LCD Keypad Shield.

The following diagram describes the project circuitry.

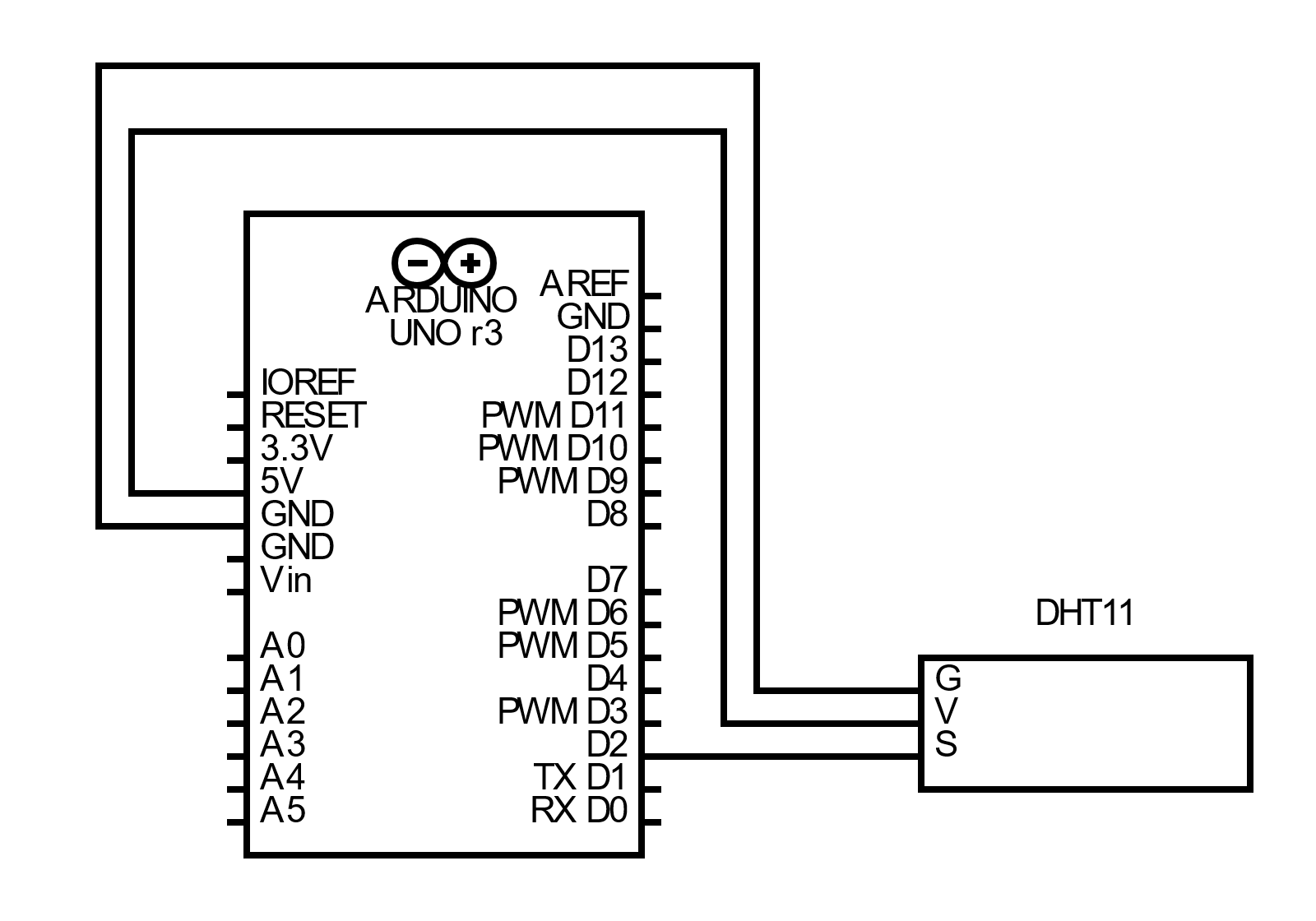


Figure – Schematic

## Software

The software language for the Arduino UNO is C++ using the built in Arduino libraries. In addition, the following libraries where used:

* DHT sensor library (for driving the DHT Temperature and Humidity Sensor: <https://www.arduino.cc/reference/en/libraries/dht-sensor-library/> )
* LiquidCrystal (Optional: for driving the LCD: <https://www.arduino.cc/reference/en/libraries/liquidcrystal/>)

There are three sections of the software: the Main Driver, Sensor Read and Interval ISR. The following sections describes each section.

### Main Driver

The Main Driver contains the setup and loop code for the UNO.

#### Setup

The setup code includes the initialization of the following variables:

|  |  |  |
| --- | --- | --- |
| Label | Type | Initialization Value |
| time | Unsigned long | 0 |
| trigger | Bool | False |
| sample\_buffer | TEMP::DATA\_BUFFER | Zeroed out |
| sensor\_buffer | TEMP::DATA\_BUFFER | Zeroed out |

In addition, the setup code performs the following:

1. The TIMER::setISRTimer() is called with argument 1 within a disable/reenable interrupts section.
2. The sensor pointer is allocated a TEMP::Sensor object with the DHTPIN, DHTTYPE, the address of the sensor\_buffer struct and the address of the sensor\_offset struct.
3. The Serial class is initialized with baud rate 9600 and the csv header lines are written out.

#### Loop

The main loop will call the sample() method from the sensor class at every frame. This will update the sensor\_buffer variable with the current temperature from the DHT Sensor. The loop() function will also check if the trigger flag is raised. If the trigger flag is True, the loop() function will print out the current time in seconds and the read temperature from the sample\_buffer in CSV format. After this, the trigger flag is set to False.

### Sensor Read

The Sensor Read software includes a Sensor wrapper class for the DHT sensor. Upon instantiation, the Sensor class holds the address for the sensor\_buffer and offset variables from the main thread. The Sensor::sample() class method performs the following operation:

1. Call the DHT readTemperature method to retrieve the current temperature.
2. Apply the displacement defined by the offset.
3. Update the sensor\_buffer with the calculated temperature. Note that this operation is the only part of this method that disables and reenables interrupts. Interrupts cannot be disabled with readTemperature() called.

### Interval ISR

The Interval ISR software contains the operations to setup a timer hardware interrupt and operations during an interrupt.

The configuration of setting up a timer interrupt is based off the example implementation from <https://www.instructables.com/Arduino-Timer-Interrupts/> . The OCR1A hardware register is set to count every 1 second interval that occurs from the hardware clock. Other register operations are performed to ensure that the timer interrupt is enabled.

The ISR() interrupt vector points to the OCR1A triggered interrupt via TIMER1\_COMPA\_vect event. The following operations occur:

1. The time variable is incremented by 1.
2. The trigger is set to True if the time variable is a multiple of INTERVAL.
3. The sample\_buffer temperature is set to the current value of the sensor\_buffer temperature.

## Configuration

The Temperature Sensor project offers configuration options for both hardware and software settings.

For hardware, an optional LCD display may be connected to the UNO to display a live update of the temperature. Pin settings may need to be updated in the temperature\_sensor\_p2.ino file.

For software, the temperature\_sensor\_p2.ino file contains the following variables that may be configured. Note that the rebuild and load operation must be performed for these settings to take effect.

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Default | Description |
| LCD\_ENABLE | 0 or 1 | 0 | Optional LCD enablement. 1=ON, 0=OFF |
| INTERVAL | unsigned long | 10 | The interval time in seconds to sample the temperature. |
| DHTPIN | uint8\_t | 2 | Pin number for the DHT signal line. |
| DHTTYPE | uint8\_t | DHT11 | Defines the type of DHT sensor. See DHT library documentation |
| SENSOR\_OFFSET | TEMP::DATA\_BUFFER | -1.66 | Calibration offset for the Temperature Sensor. This value is added to the raw temperature readings from the DHT. |

# Implementation

The source code of the Temperature Sensor project is as follows:

## temperature\_sensor\_p2.ino

/\*

  Temperature Sensor

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#define LCD\_ENABLE 0

#include <DHT.h>

#if LCD\_ENABLE

#include <LiquidCrystal.h>

#endif

#include "isr\_timer.hpp"

#include "temp.hpp"

// Sampling Variables

const unsigned long INTERVAL = 10;

volatile unsigned long time;

volatile bool trigger;

volatile TEMP::DATA\_BUFFER sample\_buffer;

// Temperature Sensor

const uint8\_t DHTPIN = 2;

const uint8\_t DHTTYPE = DHT11;

const TEMP::DATA\_BUFFER SENSOR\_OFFSET = {-1.66};

TEMP::DATA\_BUFFER sensor\_buffer;

TEMP::Sensor \*sensor = 0;

#if LCD\_ENABLE

// LCD Pins (if available)

const int pin\_RS = 8;

const int pin\_EN = 9;

const int pin\_d4 = 4;

const int pin\_d5 = 5;

const int pin\_d6 = 6;

const int pin\_d7 = 7;

const int pin\_BL = 10;

// LCD Object

LiquidCrystal \*lcd = 0;

#endif

void setup()

{

    // Initialize Interrupt Variables

    time = 0;

    trigger = false;

    sample\_buffer.temp\_f = 0.0;

    // Setup 1 Hz Timer Interrupt

    cli(); // Disable Interupts

    TIMER::setISRTimer(1);

    sei(); // Enable Interrupts

    // Initialize Temperature Sensor and Buffer

    memset(&sensor\_buffer, 0, sizeof(sensor\_buffer));

    sensor = new TEMP::Sensor(DHTPIN, DHTTYPE, &sensor\_buffer, &SENSOR\_OFFSET);

    // Setup Serial and Start CSV Logging

    Serial.begin(9600);

    Serial.println("time\_sec,temperature\_f");

#if LCD\_ENABLE

    lcd = new LiquidCrystal(pin\_RS, pin\_EN, pin\_d4, pin\_d5, pin\_d6, pin\_d7);

    lcd->begin(16,2); // 16 col, 2 rows

    lcd->setCursor(0,0);

    lcd->print("TIME");

    lcd->setCursor(8,0);

    lcd->print("TEMP (F)");

#endif

}

void loop()

{

    // Sample on every frame and update local buffer

    sensor->sample();

#if LCD\_ENABLE

    lcd->setCursor(0,1);

    lcd->print(time);

    lcd->setCursor(8,1);

    lcd->print(sample\_buffer.temp\_f);

#endif

    // If the interval trigger is set - log out to Serial

    // Note: this operation is SLOW - interrupts SHOULD BE enabled

    if (trigger)

    {

        Serial.print(time);

        Serial.print(",");

        Serial.println(sample\_buffer.temp\_f);

        trigger = false;

    }

}

// This triggers at 1 Hz (every second)

ISR(TIMER1\_COMPA\_vect)

{

    // Increment seconds counter

    time++;

    // Determine trigger state

    trigger = (time % INTERVAL) == 0;

    // Set sample value to the value(s) stored in the current buffer

    sample\_buffer.temp\_f = sensor\_buffer.temp\_f;

}

## temp.hpp

/\*

  Temperature Sensor

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#ifndef \_\_TEMP\_HPP\_\_

#define \_\_TEMP\_HPP\_\_

#include <Arduino.h>

#include <DHT.h>

namespace TEMP

{

    typedef struct data\_buffer

    {

        float temp\_f;

    } DATA\_BUFFER;

    class Sensor

    {

    public:

        Sensor(uint8\_t pin, uint8\_t type, DATA\_BUFFER \*buffer, const DATA\_BUFFER \*offset);

        bool sample();

    private:

        DHT \*dht;

        DATA\_BUFFER \*buffer;

        const DATA\_BUFFER \*offset;

    };

}

#endif

## temp.cpp

/\*

  Temperature Sensor

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#include <DHT.h>

#include "temp.hpp"

TEMP::Sensor::Sensor(uint8\_t pin, uint8\_t type, DATA\_BUFFER \*buffer, const DATA\_BUFFER \*offset)

{

    this->dht = new DHT(pin, type);

    this->dht->begin();

    this->buffer = buffer;

    this->offset = offset;

}

bool TEMP::Sensor::sample()

{

    // Read Fahrenheit Temperature

    // Sample Temperature and Apply Offset (Interrupts MUST be enabled)

    float temp\_f = dht->readTemperature(true) + offset->temp\_f;

    // Modify critical data

    cli();

    buffer->temp\_f = temp\_f;

    sei();

    // Return true on success

    return !(isnan(buffer->temp\_f));

}

## isr\_timer.hpp

/\*

  Temperature Sensor

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#ifndef \_\_TIMER\_HPP\_\_

#define \_\_TIMER\_HPP\_\_

#include <Arduino.h>

namespace TIMER

{

    void setISRTimer(const unsigned long hertz);

}

#endif

## isr\_timer.hpp

/\*

  Temperature Sensor

  Author: Nate Lao (nlao1@jh.edu)

  Designed for Arduino UNO

\*/

#include "isr\_timer.hpp"

void TIMER::setISRTimer(const unsigned long hertz)

{

    // Enables the TIMER1\_COMPA\_vect Interrupt Service Routine

    // The base value of the OCR1A timer compare register

    // for 1 Hz. To increase rate divide by the desired

    // frequency (input:hertz).

    const unsigned long TIMER\_COMPARE\_REGISTER\_1HZ = 15624;

    TCCR1A = 0; // set entire TCCR1A register to 0

    TCCR1B = 0; // same for TCCR1B

    TCNT1 = 0;  // initialize counter value to 0

    // set compare match register for x hz increments

    OCR1A = TIMER\_COMPARE\_REGISTER\_1HZ / hertz;

    // turn on CTC mode

    TCCR1B |= (1 << WGM12);

    // Set CS10 and CS12 bits for 1024 prescaler

    TCCR1B |= (1 << CS12) | (1 << CS10);

    // enable timer compare interrupt

    TIMSK1 |= (1 << OCIE1A);

}

# Usage

The following instructions describes the steps to use the Temperature Sensor project:

1. Implement the circuit as described in Section 3.1.
2. Connect the Arduino UNO to a Host computer with a Serial interface program installed.
3. Using an Arduino IDE, load the Temperature Sensor software to the Arduino UNO.
4. Using a Serial interface program, such as PuTTY.
   1. Ensure that logging to a file is enabled prior to starting the connection. It is recommended that the output file is saved to a .csv.
   2. Open a serial port to the UNO with baud rate of 9600.
   3. The following should appear:

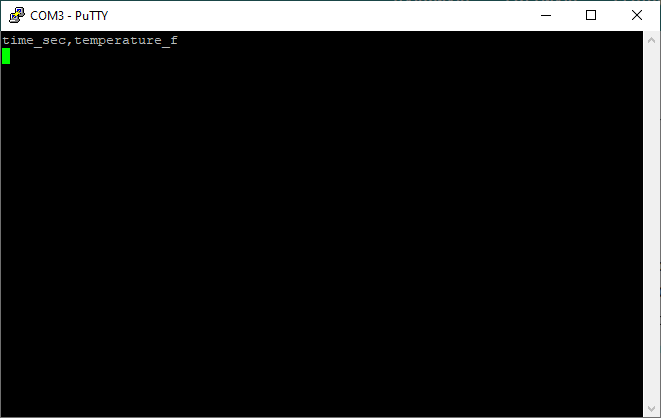


Figure – Serial Initial Output

* 1. For every 10 seconds, a log of the temperature will appear:

Text

Description automatically generated with medium confidence

Figure 4 – Serial Control Output

1. After the desired amount of time, the serial output file defined in Step 4.a will contain the recorded temperature measurements. This can be used for post processing analysis.
   1. Note: Depending on your host and serial application you use/settings, you may need to “clean up” csv file.
2. To exit the program, simply disconnect the UNO from any power supplies.

# Results

The following sections contains the results of various tests conducted on the Temperature Sensor.

## Control Test

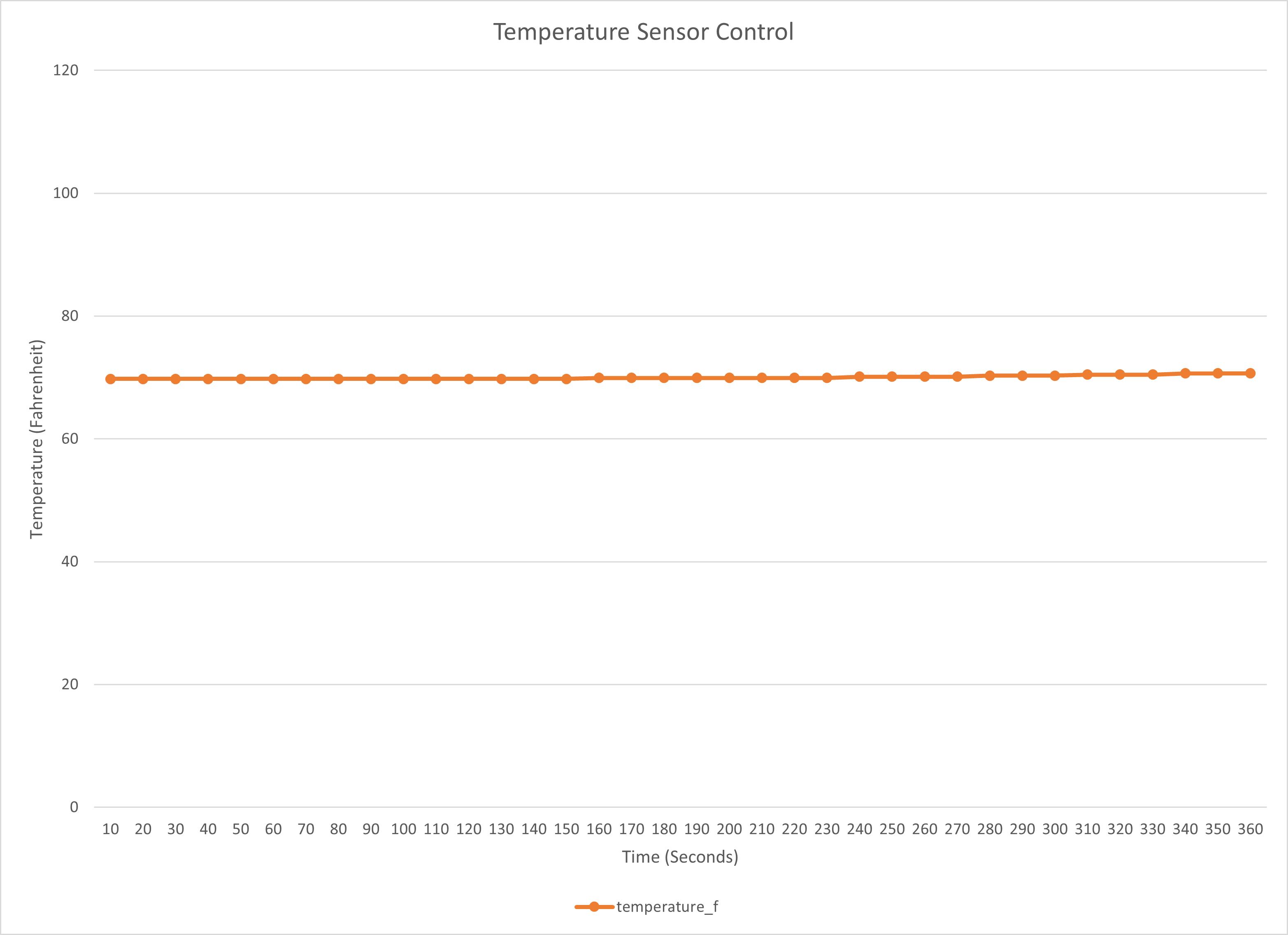
The following data and graph are the measurements of Temperature Sensor running at idle room temperatures.

Figure - Temperature vs. Time Graph (Control)

|  |  |
| --- | --- |
| Time (Sec) | Temp (F) |
| 10 | 69.76 |
| 20 | 69.76 |
| 30 | 69.76 |
| 40 | 69.76 |
| 50 | 69.76 |
| 60 | 69.76 |
| 70 | 69.76 |
| 80 | 69.76 |
| 90 | 69.76 |
| 100 | 69.76 |
| 110 | 69.76 |
| 120 | 69.76 |
| 130 | 69.76 |
| 140 | 69.76 |
| 150 | 69.76 |
| 160 | 69.94 |
| 170 | 69.94 |
| 180 | 69.94 |
| 190 | 69.94 |
| 200 | 69.94 |
| 210 | 69.94 |
| 220 | 69.94 |
| 230 | 69.94 |
| 240 | 70.12 |
| 250 | 70.12 |
| 260 | 70.12 |
| 270 | 70.12 |
| 280 | 70.3 |
| 290 | 70.3 |
| 300 | 70.3 |
| 310 | 70.48 |
| 320 | 70.48 |
| 330 | 70.48 |
| 340 | 70.66 |
| 350 | 70.66 |
| 360 | 70.66 |

Table – Temperature vs. Time Data (Control)

## Hot Air Test

The following data and graph are the measurements of the Temperature Sensor subjected to a low heat source (hot air blower) for approximately 5 minutes and letting return to room temperature for approximately 5 minutes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time (Sec) | Temp (F) | Time (Sec) | Temp (F) | Time (Sec) | Temp (F) |
| 10 | 71.2 | **210** | 152.38 | **410** | 102.88 |
| 20 | 71.2 | **220** | 150.94 | **420** | 100.18 |
| 30 | 71.02 | **230** | 148.06 | **430** | 97.84 |
| 40 | 71.92 | **240** | 146.26 | **440** | 95.5 |
| 50 | 97.48 | **250** | 143.02 | **450** | 93.52 |
| 60 | 121.6 | **260** | 138.7 | **460** | 91.9 |
| 70 | 133.84 | **270** | 135.64 | **470** | 90.1 |
| 80 | 137.8 | **280** | 136 | **480** | 88.3 |
| 90 | 139.6 | **290** | 137.8 | **490** | 86.86 |
| 100 | 137.44 | **300** | 140.86 | **500** | 85.6 |
| 110 | 132.76 | **310** | 139.96 | **510** | 83.98 |
| 120 | 127.72 | **320** | 136 | **520** | 82.72 |
| 130 | 127 | **330** | 131.68 | **530** | 81.64 |
| 140 | 130.42 | **340** | 127.54 | **540** | 80.56 |
| 150 | 137.08 | **350** | 123.22 | **550** | 79.84 |
| 160 | 140.86 | **360** | 119.26 | **560** | 78.94 |
| 170 | 142.48 | **370** | 115.48 | **570** | 78.22 |
| 180 | 144.64 | **380** | 112.06 | **580** | 77.5 |
| 190 | 148.06 | **390** | 108.64 | **590** | 76.96 |
| 200 | 151.3 | **400** | 105.58 | **600** | 76.42 |

Table – Temperature vs. Time Data (Hot Air)

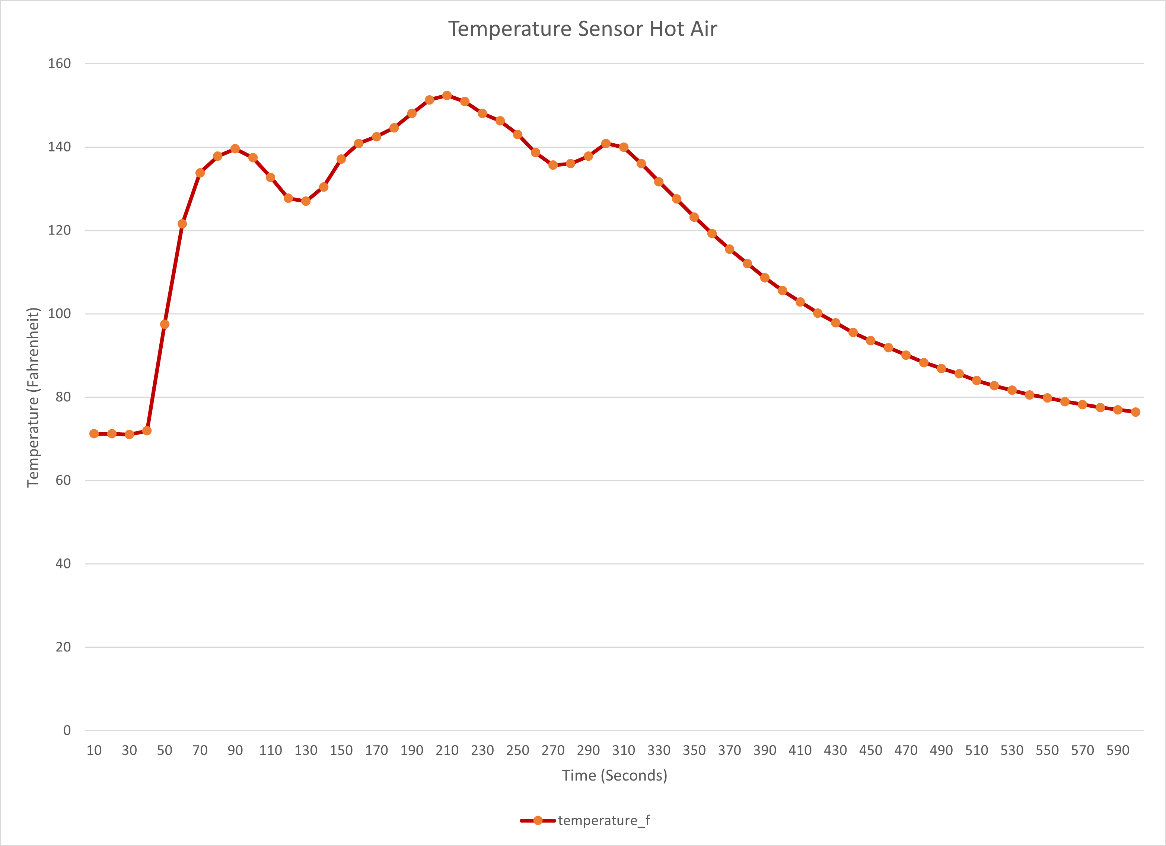


Figure – Temperature vs. Time Graph (Hot Air)

## Refrigerator Test

The following data and graph are the measurements of the Temperature Sensor subjected to refrigerator temperatures for approximately 5 minutes and letting return to room temperature for approximately 5 minutes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time (Sec) | Temp (F) | Time (Sec) | Temp (F) | Time (Sec) | Temp (F) |
| 10 | 70.48 | 210 | 52.12 | 410 | 55.72 |
| 20 | 70.66 | 220 | 51.4 | 420 | 56.62 |
| 30 | 70.84 | 230 | 50.68 | 430 | 57.7 |
| 40 | 70.84 | 240 | 49.96 | 440 | 58.6 |
| 50 | 70.12 | 250 | 49.42 | 450 | 59.5 |
| 60 | 69.04 | 260 | 48.88 | 460 | 60.4 |
| 70 | 67.78 | 270 | 48.34 | 470 | 60.94 |
| 80 | 66.34 | 280 | 47.98 | 480 | 61.66 |
| 90 | 64.9 | 290 | 47.62 | 490 | 62.2 |
| 100 | 63.64 | 300 | 47.26 | 500 | 62.74 |
| 110 | 62.2 | 310 | 46.9 | 510 | 63.1 |
| 120 | 60.94 | 320 | 46.54 | 520 | 63.64 |
| 130 | 59.86 | 330 | 46.36 | 530 | 64 |
| 140 | 58.6 | 340 | 46.36 | 540 | 64.36 |
| 150 | 57.52 | 350 | 47.44 | 550 | 64.72 |
| 160 | 56.44 | 360 | 48.88 | 560 | 65.08 |
| 170 | 55.54 | 370 | 50.5 | 570 | 65.26 |
| 180 | 54.64 | 380 | 51.94 | 580 | 65.62 |
| 190 | 53.74 | 390 | 53.38 | 590 | 65.8 |
| 200 | 52.84 | 400 | 54.46 | 600 | 66.16 |

Table – Temperature vs. Time (Refrigerator)

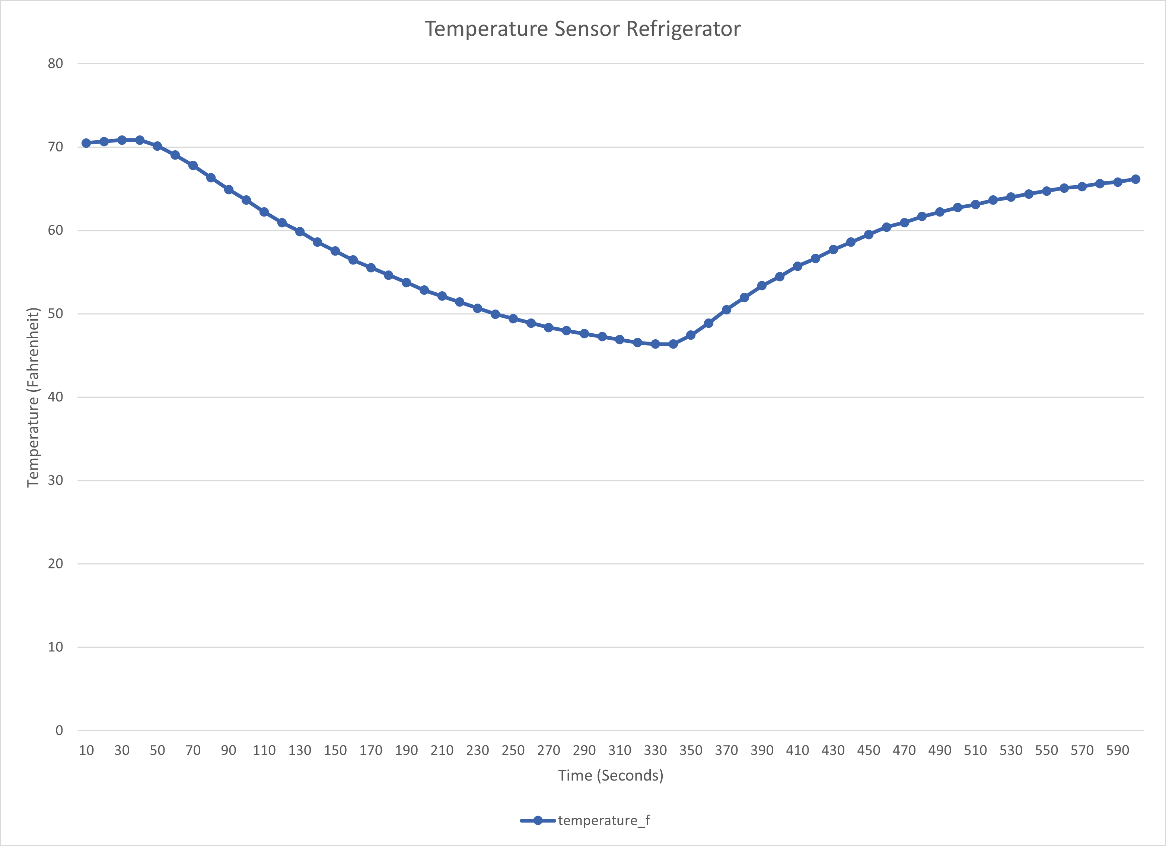


Figure – Temperature vs. Time Graph (Refrigerator)

# DEMO

A demonstration of the Temperature Sensor project can be found here:

<https://youtu.be/mmaa9pm_DD4>