Temperature Control

Team 1

Alex Barteau, Roshan Barua, Ruturaj Pujari

2016

Contents

[Abstract 2](#_Toc451368152)

[System Description/Introduction 2](#_Toc451368153)

[Hardware/Software Architecture 2](#_Toc451368154)

[WIFI - ESP8266 2](#_Toc451368155)

[AT Command Set 2](#_Toc451368156)

[TM ESP8266 Library 3](#_Toc451368157)

[Temperature Sensor- DS18B20 3](#_Toc451368158)

[TMOneWire Library 4](#_Toc451368159)

[System Design 4](#_Toc451368160)

[Wi-Fi Connection 5](#_Toc451368161)

[Physical 5](#_Toc451368162)

[Application Layer 5](#_Toc451368163)

[Call Back Design 5](#_Toc451368164)

[Temperature 5](#_Toc451368165)

[PID Control 6](#_Toc451368166)

[Conclusion 8](#_Toc451368167)

# Abstract

Accurate temperature control of homes and apartments classically have been signal point systems. A central temperature monitoring point is usually one that measures the temperature of a single location, and either opens or closes a contact to indicate if heat should be turned off or on. These systems are simple and reliable, but have room for dramatic improvement. The goal of the project is to extend a simple thermostat, to a multi-point solution. In case of a New York apartment, this would allow for accurate temperature control of a multi-room home.

# System Description/Introduction

The system consists of a STM32F4 is connected to a standard thermostat point using a relay to close or open the circuit, and secondary STM32F4 providing remote temperature sensors, Figure 1: System Design. This primary point would serve as an access point for additional temperature nodes, as well as host a simple http server displaying the current measured temperatures, and set point. The set point can be directly updated through the HTTP server. Additional STM32F4 will serve as remote temperature measurement points. The temperature will be relayed back to the primary (to calculate an average temperature) and used to control the heating of the home. If the connection to the secondary point is lost, dropped, or never present, the primary system will use its local temperature measurement for control instead.



Figure 1: System Design

Include a high level system diagram and Description

# Hardware/Software Architecture

The software is broken into 3 main segments corresponding to the unique hardware in Figure 1: System Design, input (temperature), control (PID), communication (WiFi) Figure 2: Software Architecture. Though the “station” microcontrollers do not the PID controller, it was included in their deployments. This allows a single microcontroller to be swapped out on the fly as the primary by simply changing addressing pins on the device and resetting it. This reusable and common code structure allowed for the team to reuse large segments of code easily within the “access point” and “station” microcontrollers. Lastly, the WiFi communication code is similarly shared between all microcontrollers. Simply the request, response, and polling behaviors are modified based on each devices unique pinned address.

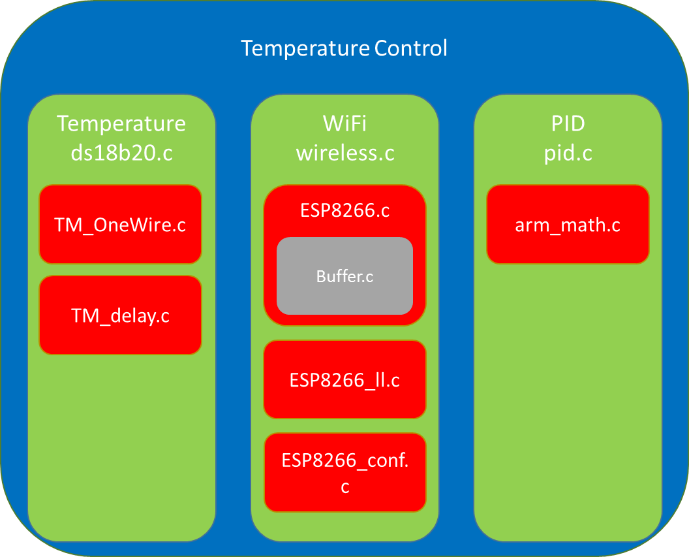


Figure 2: Software Architecture

## WIFI - ESP8266

The team chose to use an ESP8266 as the Wi-Fi module as it is a cheap UART Wi-Fi module, that already implements all of TCP, UDP, and SSL. This allows the device to essentially act as a stream based device rather than a packet solution. Additionally, the ESP8266 uses the AT command set which allows for easy computer based experimentation to ensure commands work as intended.

### AT Command Set

The AT command set is a subset of the Hayes command set. These are string based commands originally designed to control dialup modem. The team only needed a few AT commands primarily: Access Point Hosting, Server Hosting, Open Connection, Close Connection, Send Data.

All commands except for receiving data from a connection are essentially blocking. The command and the arguments are sent via UART, and then the ESP8266 response with either Error, the command repeated, or Ok.

As long as an AT command is not running, the ESP8266 can place “IPD:ConnectionInfo:Data” onto the UART. Inside the ConnectionInfo is the number of bytes, connection number, IP, and port in the received message. Typically, this will be an entire TCP stream or UDP packet unless the stream exceeds the ESP8266 buffer size. There is no end of message indicator. Similarly, a “Connection Received” message can be sent via UART whenever the Wi-Fi device is idle.

The Basic AT command set list is defined in the Esp8266.c wherein the regular commands used are:

#define ESP8266\_COMMAND\_CWMODE 5

#define ESP8266\_COMMAND\_CIPSERVER 6

#define ESP8266\_COMMAND\_CIPDINFO 7

#define ESP8266\_COMMAND\_SEND 8

#define ESP8266\_COMMAND\_CLOSE 9

#define ESP8266\_COMMAND\_CIPSTART 10

#define ESP8266\_COMMAND\_CIPMUX 11

### TM ESP8266 Library

The TM ESP8266 Libraries provided a call back based ESP8266 driver. The library required us to implement low level UART initiation, device reset, UART send data, and UART Receive data interrupt handler. These low level functions allow the library to initialize and communicate with the ESP8266.

The library provided weak functions that had to be overridden to handle the callbacks in the case of ESP8266 informing the STM32F4 that it has received a new connection, or it has received a new complete message.

Data requests including HTTP, and temperature as well as the corresponding parsing is entirely done by the “data received” callback function.

## Temperature Sensor- DS18B20

The DS18B20 is a digital temperature sensor that uses the “OneWire” protocol to issue commands and retrieve data. The sensor provides from 9 to 12 bits of resolution which takes up to 750ms to process. After a temperature conversion command has been issued, the communication wire must be left silent for the conversion to complete. The DS18B20 must be powered with 3.3V to 5V.

### TMOneWire Library

The TMOneWire library provided the basic communication mechanism with which team issued commands to the DS18B20. The entire timing and connection establishing levels were provided within the library. The team built functions on this base to issue the “temperature read” command to the DS18B20 as well as to retrieve and convert the measured temperature to Celsius. The command temperature processing was built into a timed task to prevent any timing violations during a temperature conversion cycle.

# System Design

We have connected 4 STM32 Boards with one acting as an access point (Home Base Network) and the other three boards acting as the clients to the board as shown in Figure 1: System Design.

## Wi-Fi Connection

### Physical

The ESP8266 WiFi modules provide connectivity via UART. The modules handle the entire TCP/UDP and even SSL layer processing. Therefore communication with remote devices can be thought of like stream based communication.

### Application Layer

Within the callback functions used to implement the wifi connections, simple data parser were built. These parsers handled simple HTTP requests and replies as well as the interboard temperature request and reply communication.

### Call Back Design

The ESP8266 AT command set library the team used provided a basic set of callback functions that needed to be implemented. These functions provided the basis of connection establishing, sending of data, receiving data, and communication errors. This callback based designed allowed the team to develop without use an RTOS, but complicated keeping data streams separate. After struggling to isolate streams, the team decided to only allow one active data stream at a time. Therefore, one the primary “access point” was pulling the devices, it would connect to them and processing their data sequentially. Additionally, the HTTP server was shut down for this duration to prevent HTTP requests from interfering with communication.

Building on top of the ESP8266 library proved to be quite challenging. The library was buggy, and had be fixed. For example, communication streams needed to have a new line character in them to be processed immediately. Additionally, Messages between length of 28 and 128 bytes were incorrectly parsed if they had a new line character in the middle of the message. Finally the GDB print statements appeared to interfere with the timing of the UART connection with the ESP8266 modules. This meant that bytes were either dropped during sending and receiving. This made developing and testing very challenging.

## Temperature

The DS18B20 is a digital temperature sensor that must be commanded to perform a new temperature measurement. Then the new measurement can be read following a device reset. The process for performing a new measurement and retrieving the value is outlined below.

1. Reset the Line
2. Send the Connection to a Specific device
3. Perform a byte read twice
4. Recover the float value
5. Reset
6. Send a Request to get new temperature
7. Wait 750ms (time period of 12bit measurement)

## PID Control

The input into the PID control loop is the average of all of connected temperature sensors. The average must be calculated by counting the number of responses received to prevent the sensed value being incorrectly calculated if one of the remote nodes disconnects from the network without replying. To handles this disconnect scenario, all of the remote nodes are polled twice (once every 15 seconds) before attempting to calculate an average. This gives the system a higher chance of receiving data from all nodes. Every 30 seconds, the PID loop is recalculated to find what the actuation level should be. The PID output is clamped to 0% for all outputs below 10% and the maximum output is 100%. This is to prevent the physical relay from chattering on and off very quickly and wearing out with low actuation levels.

The output signal from the PID loop is a closed or open relay contact. Therefore the output level is converted to a PWM signal with period 1 minute. The low PWM period is account for the slow temperature changes in a room or house.

# Conclusion

The entire wireless temperature control system could have been implemented directly on the ESP8266. The embedded wifi module has multiple GPIO that could have been used for both the PWM output as well as measuring the DS18B20 temperature sensor. Additionally, developing solely on the ESP8266 would have made it easier to keep communication stream separate, as well as reply to data requests in a more timely fashion.