Project Writeup (Beaglebone Cobalt)

Gabriel Faulhaber
Minhoe Kim
Ken Ni

1. System explanation
    - What does the system do?

        The smart temperature regulator reads the temperature and turns on a fan if the user prefers it to be cooler. LED indicators display status of the fan. Additionally we generate a website to show current and historical temperature data.

    - How does it do it?

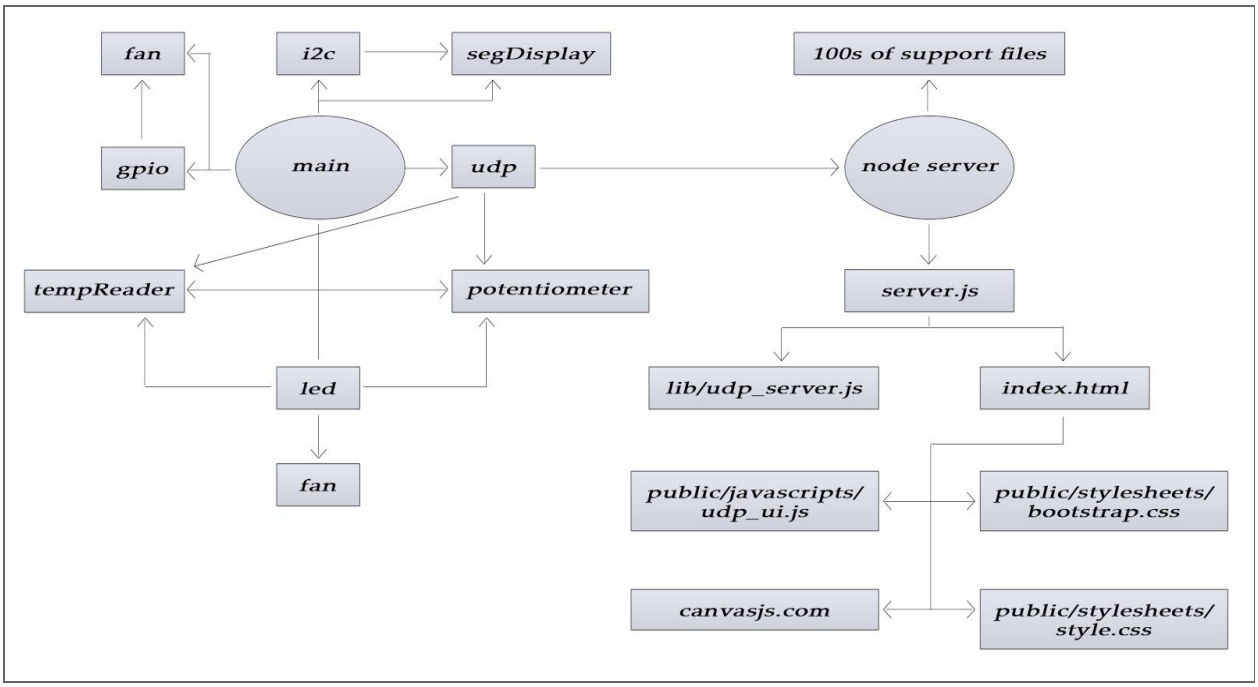

Figure 1. System structure

1. The main file accesses the following initialization functions in the following header files:
    - fan.c
    - gpio.c
    - i2c.c
    - segDisplay.c
    - tempReader.c
    - potentiometer.c
    - udp.c

2. The main file then runs thread functions to initialize the udp, display, led and fan.
3. The i2c file calls the segDisplay file to do the low level cape display functions
4. The program simultaneously evaluates the current temperature vs the preferred temperature determined by potentiometer to set the appropriate fan, led, and display value.
5. The udp takes one of 3 commands from the node server over port 12345:
    a. Preferred temperature
    b. Actual temperature
    c. Fan status

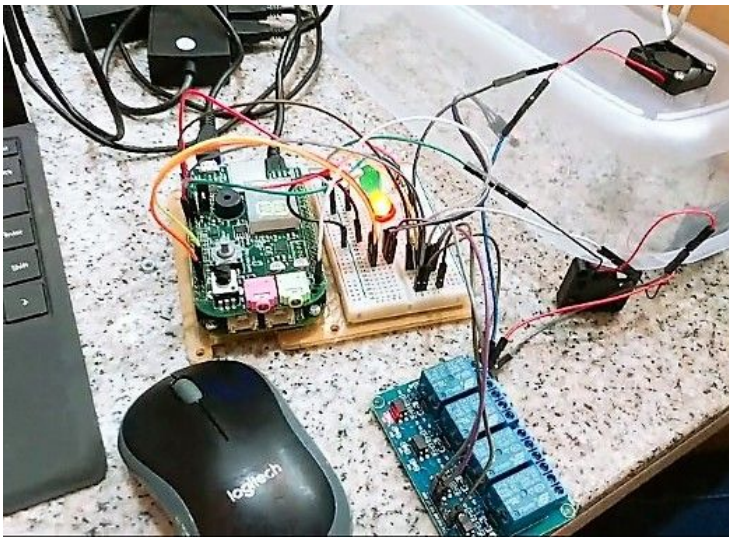

Figure 2. Wiring all components
(Temperature sensor and some fans are inserted in the box)

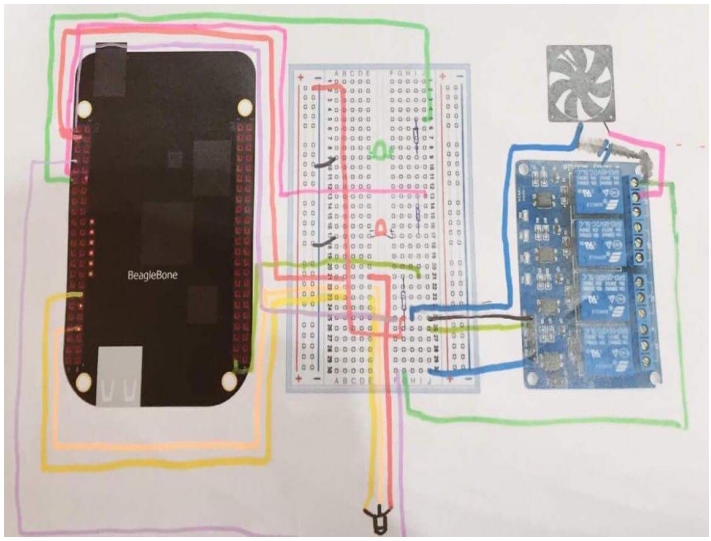

Figure 3. Wiring diagram of Figure

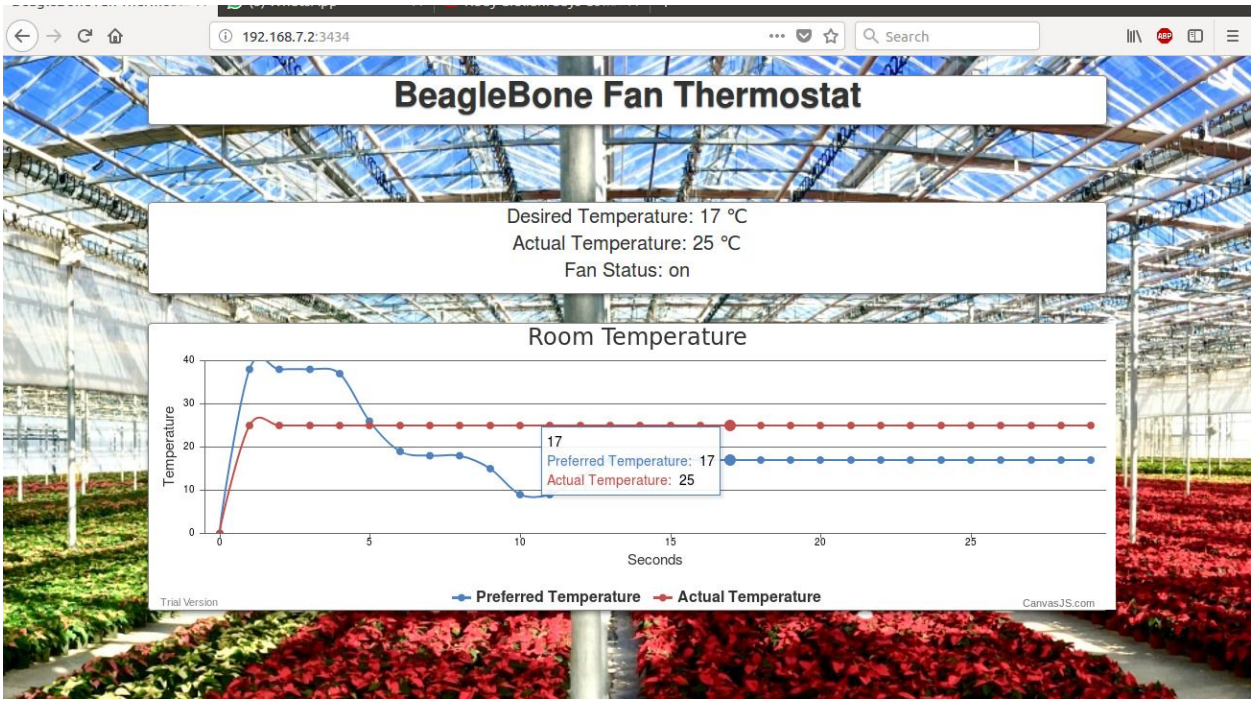

Figure 4. Screenshot of Website

How the website works: the website is running on the node.js runtime environment and uses npm as the package manager.

1. The server.js file connects all required packages, udp and website files.
2. The udp_server.js file is designed to connect the node server to the udp.c file of the main C server over port 12345.
3. The udp_ui.js sends commands via the udp_server.js file requesting actual temperature, preferred temperature, and fan status. In turn, this file uses DOM manipulation to update html IDs specific to each variable.
4. The index.html connects to the udp_ui.js links to the udp_ui.js file, the bootstrap.css and style.css. It also connects to the canvasjs.com js file so that we can generate a graph. We had to customize this graph code so that it would allow us to enter dynamic data and be displayed in a way that was not offered as a specific template option.

2. Feature table

	Description	Completeness	Code	Notes
Temperature sensor	Reads the room temperature	5	C	Caution: loose wiring causes temperature sensor to read wrong temperature
LED	Indicates fan status (Green when on and Red when off)	5	C	
Fans (Relay)	Turn on when temperature is higher than user input; turn off otherwise	5	C	
Potentiometer	Inputs User preferred temperature	5	C	
14 Segment display	Displays the current room temperature read by temperature sensor	4	C	Number displayed changes too quickly to read
Website	Connects C to Node.js to show status & historical statistics	4	JavaScript(70 %), html (15%) css(15%), Node.JS	The graph displayed was from canvasjs.com and then customized to accept dynamic content

3. Extra Hardware & Software Used
    a. 5V Fan x3
    b. Temperature sensor (TMP36)
    c. 5V Relay
    d. Red and Green LED

4. Explanation of Challenges
    a. Figure out how to control the fan with GPIO took up a lot of time.
        i. Fan worked connecting to ground(P9.1) and VDD_3V3(P9.3).
        ii. Replaced VDD_3V3 wire to any of the GPIO (used GPIO 70 (P8.45) for our project) for controlling On and Off since GPIO produces 3.3V.
        iii. Fan did not run and could not control through the GPIO. Seemed like the fan is not getting enough power to be turned on.
        iv. Started googling about how to boost up 3V to 5V and found we can switch the SYS-5V pin by using “Relay”.
        v. Figuring out wiring relay, fan, and the beaglebone took so much time.

    b. Integrating a graph into the website
        i. Requirements: the graph needed to have multiple lines, and be able to accept dynamic data without having to reload the page every few seconds.
        ii. Issue: most of the graphs I could find were easy to integrate with dynamic data OR easy to have 2 series tracked on it, but not both.
        iii. Resolution: we were able to use the CanvasJS.com graph, which had 2 options, one for dynamic data and one for multiple series. We were able to edit the graphs so that they could allow both of these simultaneously. Even still, the dynamic data template was designed to generate dynamic data from an algorithm, not from an outside source. We were able to program the 2 series of data as arrays, which could have a number of datapoints in the range of [0,60] each being represented by a second since page load time. This was achieved using a queue system, which called for the most recent value of the preferred/actual temperature variables.

5. Acknowledgements

    - For the webserver, we took some of the base files from the 11-UdpRelay.zip file offered in the June 11th class notes, then did extensive fixes and customizations.
    - For the website graph, we used a javascript external file from the canvasjs.com website, and then customized it to use dynamic content and to display more appropriately for our content.

- To add more fans to make regulating the room temperature more efficient connect the power lead to common terminal of the relay and connect the GND wire to any DGND pin.
- The photo for the website was also used from an online source.