CS-350 Module Seven Project

02/23/25

Brandon Gerbasi

**Overview of the Process**

The implementation of the smart thermostat involved several key steps, including configuring the temperature and humidity sensor, setting up GPIO peripherals, integrating an LCD display, and ensuring seamless communication via UART. The development process required careful debugging, especially with LED behavior, button interactions, and ensuring accurate data transmission over the serial port.

Initially, the thermostat system was set up with a **default set point of 72°F**. The **AHT20 temperature sensor** was integrated via the **I2C interface**, enabling real-time temperature readings. To visually indicate the thermostat’s state, **two LEDs (red and blue)** were programmed to either **fade in and out** or remain **solid** depending on the heating or cooling conditions. The **LCD display** was implemented to alternate between displaying the **current time, temperature, system state, and setpoint**, ensuring clear user feedback. Finally, **UART communication** was established to send thermostat status updates to the server every **30 seconds**, formatted as a **comma-delimited string**.

**Challenges Faced & Solutions Implemented**

One of the most significant challenges was ensuring that the **LED indicators** responded correctly to state transitions. Initially, the LEDs did not turn on, even though the system state was changing. After debugging, it was discovered that the **state machine needed proper LED control within the on\_enter\_\* and on\_exit\_\* functions** to correctly manage transitions between heating, cooling, and off states. The solution involved ensuring that **pulse() was used for fading states** and **on() for solid states** when the setpoint was reached.

Another major challenge was that the **LCD display was not updating** despite proper wiring. After troubleshooting, it was found that the **updateScreen() function was not being called properly**. The solution was to ensure that the **LCD screen refresh alternated correctly between displaying the current temperature and the system state** at regular intervals. This update ensured a clear and informative display.

Lastly, **button functionality** required debugging, as pressing the buttons did not initially change the thermostat state. This issue was traced back to the **GPIO button configurations**, where pull\_up=False needed to be explicitly set for proper detection. Assigning the correct event handlers resolved this issue, allowing smooth transitions between thermostat states.

**What I Learned & Future Improvements**

This project provided **valuable hands-on experience** in embedded systems, GPIO programming, and state machine management. Debugging hardware interactions required **systematic testing and a deep understanding of Python’s GPIO and threading capabilities**.

If given the opportunity to improve this project further, I would explore adding:

* **Wi-Fi connectivity** to send real-time temperature data to a web dashboard.
* **Machine learning-based adaptive temperature control**, where the thermostat could learn user preferences over time.
* **Battery backup integration** to maintain functionality during power outages.

**Conclusion**

Overall, this project was an insightful experience in developing a **fully functional, hardware-integrated thermostat system**. By successfully **configuring the peripherals, debugging state management issues, and optimizing user interactions**, the thermostat met all functional requirements. Despite initial setbacks, systematic troubleshooting and code improvements led to a **fully operational prototype**. This project has strengthened my understanding of **embedded systems, Python programming, and IoT-based device interactions**, preparing me for more advanced hardware-software integrations in the future.