Corvinna Curtis

Professor Morales

CS350

19 October 2025

7-1 Final Project: Thermostat Lab

For this project, I created a smart thermostat prototype using a Raspberry Pi 4B, an AHT20 temperature sensor, three buttons, two LEDs, and a 16x2 LCD display. The goal was to demonstrate the low-level functionality of a thermostat before moving into a full Wi-Fi–connected version. My prototype reads the current temperature through the AHT20 sensor using I2C communication. The LEDs are controlled through GPIO pins to indicate whether the system is heating or cooling, and the three buttons let me cycle through modes and adjust the temperature set point up or down.

The LCD shows the date and time on the top row, while the second-row alternates between the current temperature and the current mode with the set temperature. The system also sends status updates through UART every 30 seconds, simulating how data would be transmitted to SysTec’s server software in a real production setup. The red LED fades in and out when heating is on, and the blue LED fades when cooling is active. When the room temperature reaches the set point, the LEDs switch from fading to a solid light to show that the system has stabilized.

This prototype supports all the peripherals required by SysTec’s project requirements. The I2C interface handles temperature readings from the AHT20 sensor, the GPIO pins manage the LEDs and buttons, and the LCD communicates through digital pins to display the thermostat’s data. The buttons are configured using GPIO interrupts, so the thermostat responds right away when a button is pressed. The UART serial output allows the device to send updates to the simulated server, which represents the cloud connection that will be implemented later. All of these parts work together through a Python-based state machine that controls whether the system is in heating, cooling, or off mode.

In the next phase, this thermostat will connect to the cloud using Wi-Fi instead of serial communication. The same temperature data that is currently being sent through UART will be formatted into a network message, likely JSON or a similar lightweight format, and transmitted to SysTec’s analytics platform. This would let the thermostat send regular temperature reports, mode changes, and even receive remote configuration commands from the cloud. The Wi-Fi connection will also allow over-the-air updates and data logging, which would be important for long-term product reliability and support.

The Raspberry Pi is great for prototyping because it already supports Python, I2C, GPIO, and UART, all of which were used in my building. It also includes built-in Wi-Fi, which makes it easy to move to the cloud stage. The downside is that it runs a full operating system, which uses more power and resources than what’s needed for a small thermostat. It’s perfect for development and early testing but may be more expensive and complex for mass production.

Microchip microcontrollers, such as the PIC32 or SAMD series, are more lightweight and designed for embedded systems. They have enough Flash and RAM to support the thermostat’s code and offer low power consumption, which is ideal for something that will run 24/7 in a home. They also have reliable I2C, GPIO, and UART support, and some include Wi-Fi modules. The trade-off is that programming and debugging these boards take more setup time and are usually done in C or C++, which makes development slower than on the Pi.

Freescale, now part of NXP, produces advanced microcontrollers like the Kinetis and i.MX RT series. These chips combine the speed and memory of a Raspberry Pi with the efficiency of a Microchip MCU. They also support all the required peripherals and can easily integrate Wi-Fi. The i.MX line would work well for scaling production, as it can handle the thermostat code plus secure connectivity and potential user interface features later on.

For this prototype, the Raspberry Pi 4B was the right choice because it allowed fast development and testing with full access to Python libraries and simple hardware setup. However, for the production version, I would recommend moving to an NXP (Freescale) i.MX RT microcontroller or a Microchip Wi-Fi–enabled MCU. Both options would reduce power usage and cost while keeping support for the required peripherals. These architectures also have enough Flash and RAM to handle the thermostat logic, Wi-Fi communication, and any future cloud-based features.

Overall, the smart thermostat prototype successfully demonstrated all of the required features using the Raspberry Pi. It reads real temperature data, responds to user input, and outputs information to a simulated cloud connection. The next step will focus on building a production-ready version that connects directly to SysTec’s cloud over Wi-Fi and runs efficiently on a lower-power microcontroller architecture.