**Smart Thermostat Prototype**

My thermostat prototype executes low-level smart thermostat on a Raspberry Pi 4B. The system reads room temperature from an AHT20 sensor over I²C, drives two status LEDs for heating and cooling, accepts input from three buttons, shows status on a 16×2 LCD, and publishes a periodic status string over the UART. A small state machine coordinates the modes (OFF → HEAT → COOL) so the UI stays responsive while the display and telemetry update in the background. The work here proves the control logic and I/O, which positions the project for a later Wi-Fi/cloud phase.

**Hardware and Software Summary**

Peripherals. The AHT20 temperature/humidity sensor is connected via the RPi’s I²C bus (address 0x38). The display is a standard HD44780-compatible 16×2 panel in 4-bit mode. One “mode” button cycles thermostat states; the other two increment/decrement the setpoint. Red and blue LEDs indicate heating and cooling. GPIOZero’s PWMLED and Button classes simplify LED fading and button interrupts, respectively (GPIO Zero, n.d.).

Core libraries. The Adafruit AHTx0 driver exposes temperature in °C, which I convert to °F for the UI and logic (Adafruit, 2024). python-statemachine provides a clear OFF/HEAT/COOL graph with entry actions (Python State Machine, n.d.). The Raspberry Pi’s primary UART is used for a CSV status line every 30 seconds (Raspberry Pi Foundation, 2024).

**State Machine and Behavior**

The thermostat has three states: OFF, HEAT, and COOL. The green button raises a mode\_next event to rotate through the states. In HEAT, if the current temperature is below the setpoint, the red LED fades (calling for heat); at or above the setpoint, it is solid. In COOL, if the temperature is above the setpoint, the blue LED fades; at or below the setpoint, it is solid. In OFF both LEDs are off. A deadband prevents flicker near the setpoint. The state machine keeps the logic easy to read and test, while the display/UART loops run on their own threads so button presses are snappy (Python State Machine, n.d.).

**I/O Implementation Notes**

I²C (AHT20). The sensor is initialized on the default I²C bus and read once per second. The library’s default address (0x38) matched what I²C detect reported during bring-up (Adafruit, 2024).

LCD. The LCD uses the same RS/E/D4–D7 mapping from earlier milestones. A small helper pads/truncates to 16 characters and only redraws when content changes to avoid flicker.

UART. The program tries /dev/serial0 and falls back as needed; on this Pi the working port was /dev/ttyAMA0. The Linux serial console must be disabled for user serial I/O; raspi-config handles that and points serial0 at the active UART (Raspberry Pi Foundation, 2024).

**What the User Sees**

The top LCD line shows date/time. The second line alternates every few seconds between current temperature (and humidity) and the state with the setpoint. The CSV sent over the UART every 30 seconds is:

<state>,<current\_F>,<setpoint\_F>

For example: cool,78.1,72.

**Test Results and Lessons Learned**

The small-steps approach from the milestones paid off. I verified each piece alone (AHT20 readout, simple LED fade, single-button tests) before combining them. The biggest wiring pitfall was the LCD contrast and RW/RS pins; once that settled, the display was reliable. On the software side, the state machine kept the LED rules readable, and putting the display/UART in separate threads preserved button responsiveness even while the LCD refreshed (GPIO Zero, n.d.; Python State Machine, n.d.).

**Architecture Options for a Wi-Fi Production Design**

The prototype runs well on the Raspberry Pi 4B. For a production thermostat with integrated Wi-Fi and long-term BOM and power targets, three paths were evaluated:

* Raspberry Pi 4B (SBC). Abundant RAM/Flash, mature Linux stack, built-in dual-band Wi-Fi, and fast development; tradeoffs are higher cost and power for a wall device (Raspberry Pi Foundation, 2021).
* Microchip PIC32MZ-W1 / WFI32 module. Single-chip MCU with integrated Wi-Fi and security, plus the peripherals needed here (I²C, UART, PWM/GPIO). Flash/RAM are sufficient for RTOS + TLS and cloud SDKs, with certified RF modules to shorten approvals (Microchip Technology, 2022).
* NXP i.MX RT crossover MCUs (with certified Wi-Fi module). High-performance Cortex-M MCUs with large on-chip SRAM and external QSPI Flash; pair with a certified Wi-Fi module via SDIO/SPI for more headroom while staying below SBC cost/power (NXP Semiconductors, n.d.).

This project reinforced how much reliability comes from basic discipline: wiring carefully, labeling GPIOs, and testing each layer in isolation. The button→state→LED chain is a simple story to explain and debug, and the state machine kept it that way. Small touches, such as a display helper to prevent flicker or a UART probe that falls back cleanly, made the demo feel polished.

**References**

Adafruit. (2024). *Adafruit CircuitPython AHTx0 (AHT10/AHT20) library documentation*. https://docs.circuitpython.org/projects/ahtx0/en/latest/

GPIO Zero. (n.d.). *GPIO Zero documentation: Buttons, PWMLED*. https://gpiozero.readthedocs.io/

Microchip Technology. (2022). *PIC32MZ-W1 (WFI32E01) family overview* [Product brief]. https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus/pic32mz-w1-family

NXP Semiconductors. (n.d.). *i.MX RT crossover MCUs* [Family overview]. https://www.nxp.com/imx-rt

Python State Machine. (n.d.). *python-statemachine documentation*. https://python-statemachine.readthedocs.io/

Raspberry Pi Foundation. (2021). *Raspberry Pi 4 Model B product brief/datasheet*. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

Raspberry Pi Foundation. (2024). *Using the UART on Raspberry Pi* (enable hardware serial, disable login shell). https://www.raspberrypi.com/documentation/computers/configuration.html#using-the-uart