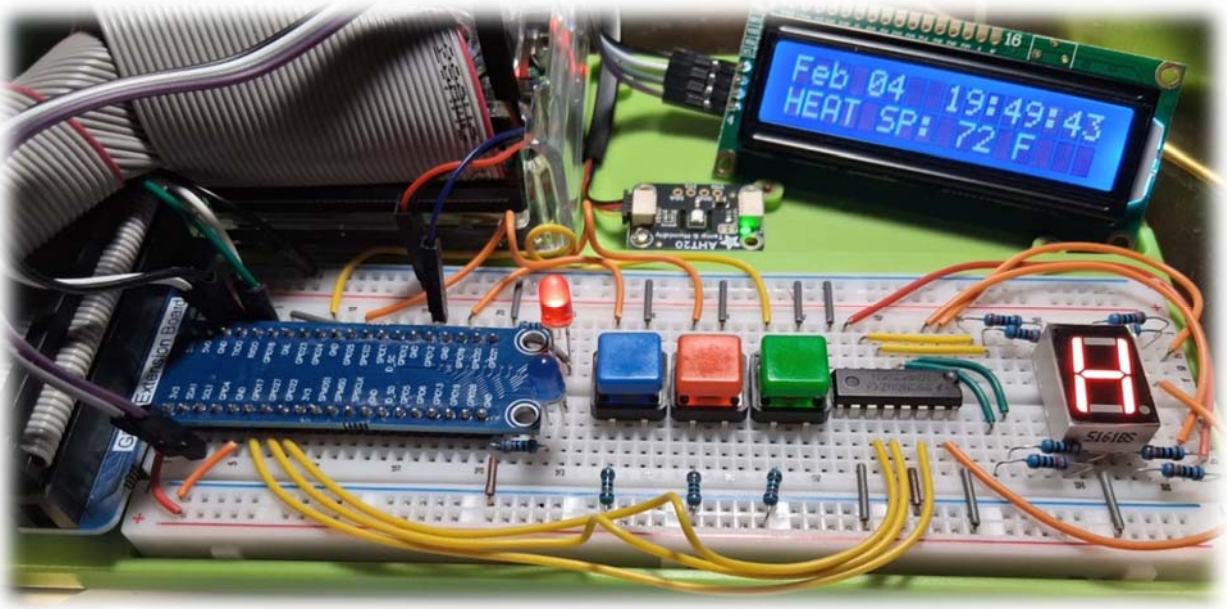
Southern New Hampshire University

SysTec Smart Thermostat Prototype Report

CS350 - Final Project
GCZ79
02/19/2026



The goal of this project was to develop a prototype smart thermostat for SysTec to demonstrate low-level thermostat functionalities and a plan to add cloud connectivity in the future.

The prototype was implemented using a Raspberry PI 4, Python, and common peripherals including an AHT20 temperature sensor, a 16x2 I2C LCD, LEDs, buttons, and UART serial communications. In addition, with the approval of my manager (instructor), I connected a 7-segment display to improve user feedback and to evaluate a more compact and slightly cheaper alternative to an LCD-based production design.

STATE MACHINE DESIGN

The behavior of the thermostat is controlled by a TemperatureMachine state machine with three states: off, heat, and cool. State transitions are triggered on green button release after a short press, while a long press toggles the temperature scale. Detecting the action on release prevents unintended state changes when the user is attempting a long press.

Entering the heat state illuminates the red LED, which fades in and out if the room temperature is below the set point (default setPoint = 72° F) or remains solid when the set point is reached. Similarly, entering the cool state illuminates the blue LED with fade or solid logic based on temperature. The off state turns both LEDs off. The state machine also updates the 7-segment display to show O, H, or C depending on the current state.

Complete state machine documentation is provided in the submitted draw.io PDF file.

PERIPHERALS

The prototype thermostat interfaces with multiple peripherals to implement the required functionality. Temperature and humidity are measured using the AHT20 sensor via the I2C bus. Two GPIO-controlled LEDs provide visual feedback for heating and cooling states, including PWM fading behavior when the temperature deviates from the set point. Three hardware buttons

generate interrupt-driven input to cycle system states and adjust the temperature set point, with long-press detection used to toggle the temperature scale. A 16×2 LCD connected through I2C displays the current date/time and alternates between temperature readings and system status. A seven-segment display provides immediate user feedback during status and value changes, and displays an animation during data transmission. Finally, a UART serial interface transmits every 30 seconds a comma-delimited system status string state, temp, setPoint for integration with external services. Detailed implementation is provided in the submitted Python source file.

ARCHITECTURES AND CLOUD CONNECTIVITY

While the prototype uses UART for serial output, the production version will connect to SysTec's cloud via Wi-Fi. Three architectures were analyzed against the business requirements: Raspberry PI, Microchip, and Freescale.

Peripheral Support: All three architectures support the required I2C (temperature sensor), GPIO with interrupts (buttons), PWM (LED fading), UART (serial), and LCD interfaces. The Raspberry Pi 4 provides extensive library support (GPIOZero, RPLCD) for rapid prototyping with Python (Raspberry Pi Foundation, 2024). The Microchip PIC18F27Q10 and Freescale Kinetis KL02 offer hardware I2C, GPIO, PWM, and UART modules, and are typically programmed in C/C++ (Microchip Technology Inc., 2020; NXP Semiconductors, 2017).

Wi-Fi Connectivity: The Raspberry Pi 4 includes integrated IEEE 802.11ac Wi-Fi, enabling immediate cloud connectivity. Both Microchip and Freescale architectures require external Wi-Fi modules (e.g., ESP8266), adding hardware complexity and cost.

Memory and Code Support: The Raspberry Pi's up to 8GB RAM and spacious Flash memories easily accommodate the Python-based prototype (~25KB source code plus libraries).

The PIC18F27Q10 offers 128KB Flash and 3.6KB RAM, while the Kinetis KL02 provides 32KB Flash and 2KB RAM, both sufficient for C/C++ implementations, but requiring code optimization and potential feature limitations.

Recommendations: I suggest completing the development with Raspberry Pi to take advantage of the existing Python code. For cost-optimized production, the team should evaluate migrating to Microchip or Freescale architectures, accounting for code rewrite effort, external Wi-Fi modules, and limited memory for future expansion.

References

Microchip Technology Inc. (2020). *PIC18F27/47Q10: 28/40-pin, Low-Power, High-Performance Microcontrollers (DS40002043C).* https://ww1.microchip.com/downloads/en/DeviceDoc/PIC18F27_47Q10-data-sheet-40002043C.pdf

NXP Semiconductors. (2017). *Kinetis KL02 32 KB Flash: 48 MHz Cortex-M0+ based microcontroller (KL02P20M48SF0, Rev. 5).* https://www.nxp.com/docs/en/data-sheet/KL02P20M48SF0.pdf

Raspberry Pi Foundation. (2024). *Raspberry Pi 4 Model B specifications*. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/