Thermostat: Final Project

Mohamed Aziz Zaghdoudi

# Introduction

SysTec seeks to expand into the rapidly growing smart thermostat market. To support this initiative, a prototype thermostat was developed using the Raspberry Pi 4B, enabling basic thermostat functionality such as temperature sensing, LED status display, LCD output, and simulated server communication via UART. The goal of this phase is to prove out the core embedded functionality before transitioning to a Wi-Fi-enabled production model.

# Prototype Functionality Summary

- Temperature Sensing: AHT2 sensor reads room temperature via the I2C bus.  
- State Management: A state machine controls OFF, HEAT, and COOL states, with transitions triggered by a button.  
- LED Indicators: Red LED fades in HEAT mode (solid when target reached), Blue LED behaves similarly in COOL mode.  
- Temperature Control: Buttons allow for setpoint increase/decrease.  
- User Feedback: LCD displays date/time, temperature, and setpoint/state alternately.  
- Data Simulation: Status data is sent over UART every 30 seconds to simulate cloud communication.

# State Machine Operation and Specification

The thermostat operates as a finite state machine with three states: OFF, HEAT, and COOL. The machine cycles states through a single button press. The transition logic is deterministic and linear, progressing OFF → HEAT → COOL → OFF. On entering HEAT or COOL, the thermostat compares the setpoint to the current temperature to determine LED behavior: a pulsing LED indicates active heating/cooling (temperature not yet reached), while a solid LED indicates the target has been reached. The state machine is implemented using the statemachine Python package and runs concurrently with display and serial communication threads.

# Coding Best Practices Applied

The code follows structured modular design and separation of concerns:  
- Proper commenting and documentation of methods and classes  
- Use of threading to ensure non-blocking display updates  
- Hardware abstraction through dedicated classes (e.g., ManagedDisplay)  
- Use of descriptive variable names and constants  
- Centralized configuration for GPIO and UART settings  
- Error handling and state cleanup upon program termination

# Peripheral Support and Integration

The thermostat supports the following peripherals:  
- Temperature Sensor (AHT20): Connected via I2C; initialized and read using Adafruit’s adafruit\_ahtx0 library.  
- LED Indicators: PWM-enabled GPIO (GPIO 18 and 23) for fading effects, controlled via gpiozero.PWMLED.  
- Buttons: GPIO inputs on GPIO 24, 25, and 12 using gpiozero.Button, with interrupt-driven callbacks.  
- LCD Display: A 16x2 monochrome character LCD connected via multiple GPIO pins, managed via the adafruit\_character\_lcd library.  
- UART Communication: Serial communication over /dev/ttyS0 at 115200 baud simulates sending temperature and state data to a server.

# Hardware Architecture Evaluation

Three architectures were evaluated for long-term deployment:

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Raspberry Pi 4B | Microchip (e.g., SAMW25) | Freescale/NXP (e.g., i.MX RT1050) |
| Peripheral Support | Full I2C, UART, GPIO, PWM | Supports I2C, UART, GPIO, PWM | Supports I2C, UART, GPIO, PWM |
| Wi-Fi Connectivity | Requires external module | Built-in Wi-Fi | External Wi-Fi module required |
| Flash & RAM | 4GB RAM, expandable storage | ~256KB Flash, 64KB RAM | Up to 4MB Flash, 512KB RAM |
| Ease of Development | Linux OS, Python, rich ecosystem | Embedded IDE (Atmel Studio, MPLAB) | Embedded IDE (MCUXpresso, FreeRTOS) |
| Cloud Scalability | Supports full Linux cloud stack | Suitable for basic IoT clients | Advanced edge computing capabilities |

# Cloud Wi-Fi Connectivity Options

- Raspberry Pi: Connects to the internet via USB or built-in Wi-Fi (on models like Pi 4B+). It can run full-stack cloud clients (e.g., MQTT with Eclipse Paho or HTTP with Requests in Python) to push data to services like AWS IoT, Azure IoT Hub, or custom REST APIs.  
- Microchip SAMW25: Features built-in Wi-Fi and supports embedded TCP/IP and TLS stacks. Ideal for low-power, cost-sensitive devices. Compatible with Harmony framework and cloud connectors (e.g., AWS IoT core libraries).  
- Freescale/NXP i.MX RT: Requires an external Wi-Fi module but supports high-speed interfaces and FreeRTOS + LwIP for cloud integration. Suitable for edge computing and devices that also perform local analytics or processing.

# Architecture Capability Summary

- Raspberry Pi 4B: Ideal for prototyping and advanced functionality. With its full OS, it can handle complex networking, data buffering, and even host its own dashboard or cloud connector.  
- Microchip SAMW25: Lightweight and efficient for production environments. Less suitable for heavy local computation but highly optimized for IoT telemetry.  
- Freescale/NXP i.MX RT1050: Balances processing power and embedded efficiency. Designed for robust industrial use cases with real-time processing needs and modular connectivity options.

# Conclusion

The smart thermostat prototype was successfully developed and tested using a Raspberry Pi 4B, which offered the necessary GPIO, I2C, UART, and processing capabilities for rapid development. The device read temperature data, controlled LED status indicators, managed state transitions through button presses, and sent telemetry over UART. Looking ahead, while the Raspberry Pi served as an excellent development platform, production deployment will benefit from exploring architectures like Microchip SAMW25 or Freescale/NXP i.MX RT1050 for better integration, lower power consumption, and built-in Wi-Fi capabilities. Each option offers a path forward depending on the performance, cost, and connectivity requirements of SysTec's commercial goals.