**7-2 Final Project: Thermostat Lab**

Allan O'Driscoll

Southern New Hampshire University

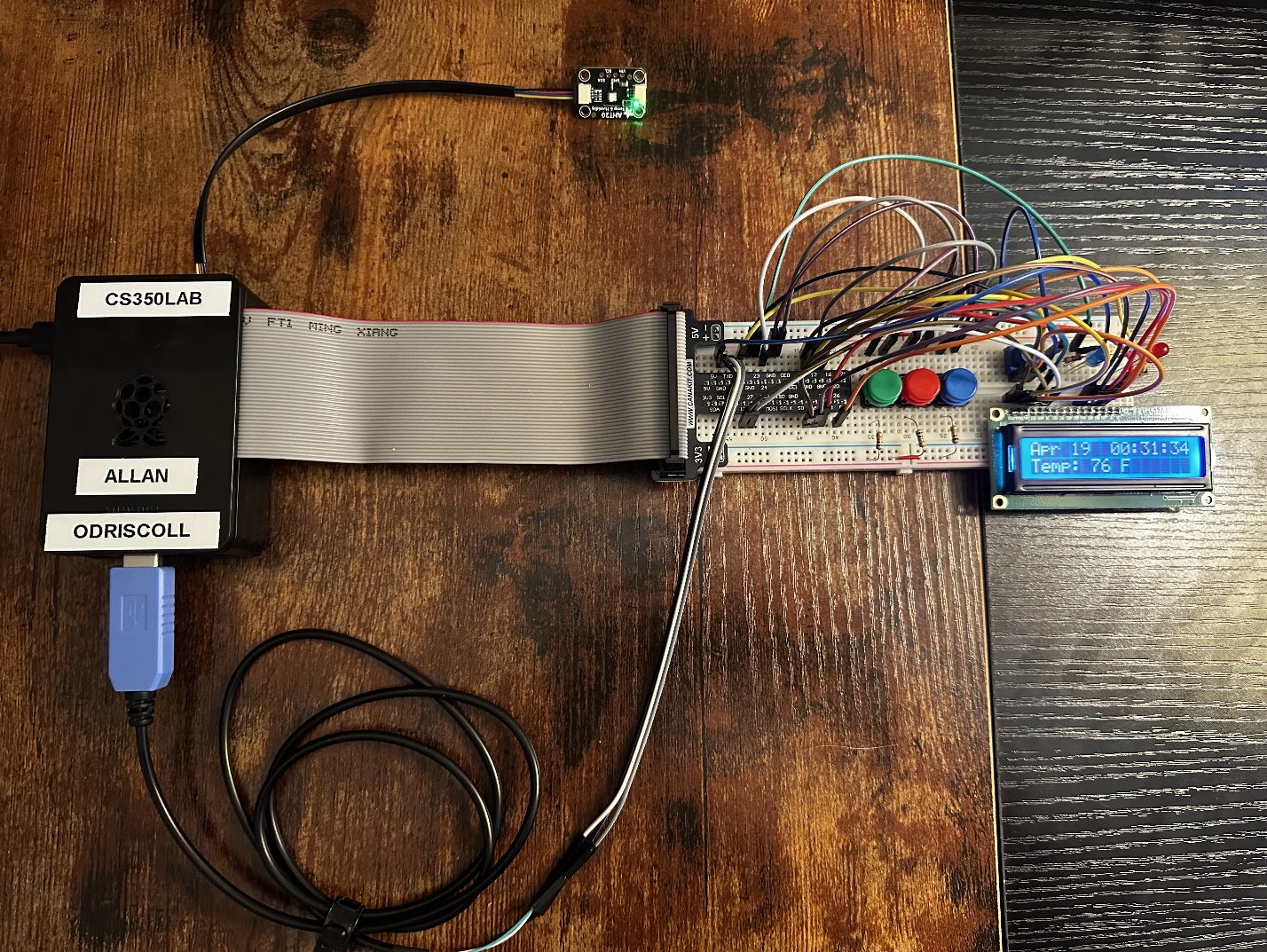
CS 350: Emerging Systems Architectures & Technologies

Professor Roland Morales

April 19, 2025

**Report**

The SysTec engineering department has been tasked with developing a smart thermostat that will allow the company to enter the global market for home-based environmental controls. This document will present a prototype that was built as a proof of concept for this project, along with a discussion of the device's requirements. Next, it will discuss several possible hardware architectures that fit the needs of a smart thermostat and the required peripherals. An image of the prototype is shown below for reference.



A state machine diagram has been included as a separate attachment that documents the technical behaviors and functionality of the system. The basic requirements for the smart thermostat include the following:

* The device must have a mechanism to set the mode to either OFF, HEAT, or COOL. This is accomplished with a button (green) that cycles through the three possible modes. This button is connected to the system through GPIO pin 24. The software implements an interrupt-driven state machine that handles the transition between the various modes.
* The device must have a mechanism to define a temperature set point. This is accomplished with two buttons connected to GPIO pins 12 and 25. The first button (red) increases the set point by one degree. The second button (blue) decreases the set point by one degree. When the unit starts, the set point should be initialized to 72 degrees Fahrenheit, but it may be adjusted to the user’s desired temperature using the buttons. Button presses generate interrupt signals that allow the system to react.
* The device must be capable of sensing the ambient temperature in the room. This is accomplished through the use of an AHT20 temperature sensor that communicates using the I2C protocol.
* The device must have a visual indication of the current mode of operation. This is accomplished through the use of two light-emitting diodes. When the system is OFF, both lights will also be off. When the system is in HEAT mode, a red light will be on and will pulse when the system is actively heating. When the system is in COOL mode, a blue light will be on and will pulse when the system is actively cooling.
* The device must have an LCD display that displays the current date and time on one line and alternates between the current temperature reading and the mode plus temperature set point on the second line. The thermostat will use a 1602A LCD for this purpose, along with several additional GPIO pins for communication.
* The device must communicate the mode, temperature reading, and set point with SysTec’s servers. In the prototype, this activity is simulated using several serial ports (i.e., the UART).

**Hardware Architecture**

Three hardware architectures have been selected for review. These include the Raspberry Pi (used in the prototype), the Microchip PIC32MZ2051W104132 controller, and the NXP (formerly Freescale) 88MW320 controller. The thermostat makes use of several interfaces to communicate with the peripherals. First, the General Purpose Input/Output (GPIO) interface is used to integrate with the buttons, LEDs, and LCD display. Second, the Inter-Integrated Circuit (I2C) interface is used to integrate with the AHT20 sensor to read the ambient temperature and humidity. Last, the Universal Asynchronous Receiver/Transmitter (UART) interface is used to simulate network activity. The completed thermostat must also support WIFI in order to transmit information to the cloud systems. All three of the selected architectures are capable of supporting the required peripherals. A comparison of each is found in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement** | **Raspberry Pi** (Raspberry Pi, 2024) | **Microchip** (Microchip, 2025) | **NXP/Freescale** (NXP, 2024) |
| Architecture Type | Single Board Computer | Microcontroller | Microcontroller/System on a Chip. |
| CPU | Quad core 64-bit ARM-Cortex A72 | MIPS32 M-Class | ARM Cortex-M4F |
| Memory | 4 GB | 256K SRAM | 512K SRAM |
| GPIO | 28 GPIO Pins | 62 GPIO Pins | 35 GPIO Pins |
| I2C | 6x I2C (via GPIO pin multiplexing) | 2x I2C | 2x I2C |
| UART | 6x UART (via USB ports and GPIO pin multiplexing) | 3x UART | 3x UART |
| WIFI | 802.11 b/g/n/ac Wireless LAN | 802.11 b/g/n Wireless LAN | 802.11 b/g/n Wireless LAN |

The Raspberry Pi is a single-board computer that runs the Linux Operating System. It is an excellent choice for the prototype because it is multipurpose, flexible, and designed for ease of experimentation. It has more than enough memory (4 GB) and processing power for the task. The thermostat system uses a total of 13 GPIO pins (including TXD and RXD). The GPIO pins are used to drive the buttons, LEDs, and LCD display. The Raspberry Pi makes 28 GPIO pins available, which is sufficient to support the required functionality (Raspberry Pi, 2024). It also has full support for I2C, which is made accessible via a Qwiic Shim that is attached to a few of the GPIO pins. This interface is used to drive the AHT20 temperature sensor. Last, the Raspberry Pi supports up to six UARTs that are made accessible via USB ports and several GPIO pins (Raspberry Pi, 2024). The UARTs are used in the prototype to simulate network traffic. The prototype utilized a Raspberry Pi 4B. However, the cheaper Raspberry Pi Zero 2w would be sufficient for the device.

The Microchip and NXP/Freescale selections are similar because they are both microcontrollers. Like the Raspberry Pi, they have hardware and software-based interfaces to support the required peripherals. However, these are lower-level devices that require more technical expertise from the development team to build the hardware and software components. Even so, they may be a good choice due to their smaller size, power requirements, and cost. The PIC32MZ2051W104132 has 256K of SRAM, while the 88MW320 has 512K (Microchip, 2025; NXP, 2024). The system’s memory is used to store code and data. The Microchip microcontroller exposes 62 GPIO pins, and the NXP microcontroller exposes 35 (Microchip, 2025; NXP, 2024). Both microcontrollers have sufficient GPIO pins to support the buttons, LED lights, and LCD display. Each microcontroller supports two I2C blocks capable of driving the AHT20 temperature sensor (Microchip, 2025; NXP, 2024). This is sufficient since we only have one I2C device. The microcontrollers also include three UART interfaces for integration with the serial ports (Microchip, 2025; NXP, 2024). The UART was used in the prototype to simulate network activity. It may or may not be used in the final product.

**Cloud Support**

All three devices were selected because they support Internet communications via Wi-Fi. The Raspberry Pi runs a full Linux operating system and can be configured to connect to local Internet access points. In the prototype, the Wi-Fi connection information was preconfigured when the initial image was written on the SD card. However, the Wi-Fi configuration can also be controlled via a set of configuration files and by using the raspi-config tool. The Microchip and NXP/Freescale microcontrollers are also fully capable of connecting to Wi-Fi networks. They support 802.11 b, g, and n protocols. Connectivity is established using the Wi-Fi configuration (SSID, password, and mode) and a software-based driver.

The challenge with all three choices is bootstrapping the Wi-Fi connection since a consumer device cannot generally be pre-configured, and the device itself has a very limited user interface. I have seen several mechanisms that bootstrap the Internet connection on an IoT device. The first involves communicating with the device using an alternate interface. For example, an application running on a mobile device can connect to the thermostat using Bluetooth to configure the Wi-Fi settings. Another approach is to have the device itself establish a temporary Wi-Fi network for configuration. A mobile device can join this Wi-Fi network to establish the configuration. Once the thermostat has the necessary configuration, it can operate independently. This process can be simplified by using open-source packages such as freeRTOS and libraries offered by cloud providers like AWS (FreeRTOS, n.d.; Amazon, n.d.). Once the Wi-Fi connection has been established, the device can communicate with servers on the Internet using the provided networking libraries. For example, Python code running on a Raspberry Pi may use the requests module to communicate with the server using a REST over HTTPS.

**Architectures’ Capabilities that Support the Code**

Several additional features of the architecture help to support the code. These include timers and interrupts. Interrupts are used to detect events like button presses. When using interrupts, the program does not need to poll the state of the button. Instead, a method is invoked automatically when the button is pressed. All three platforms support interrupts on GPIO pins and can be configured to detect when the signal changes from low to high or vice versa (Robotics Backend, 2020; Microchip, 2025; NXP, 2024). Timers also use interrupts to trigger time-based events. The Microchip controller supports a number of 16-bin and 32-bit timers/counters (Microchip, 2025). Likewise, the NXP controller supports four general-purpose times that can be programmed in various modes (NXP, 2024). Timers could be used on the device to update the display or trigger an update on the cloud server.

The code for the prototype was written in Python, which accelerated the development process. There are numerous Python libraries that support the required functionality.  The gpiozero library provided support for the three buttons and the two LEDs. The board, adafruit\_character\_lcd, and digitalio libraries provided support for the 1602A LCD display. The board and adafruit\_ahtx0 libraries provided support for the AHT20 temperature sensor, and the serial library provided support for communication over the USB and UART-based serial ports. Although the code was written in Python, it could have easily been written in another language, such as C, C++, or Java.

Code for microcontrollers like the PIC32MZ2051W104132 and 88MW320 is often written in C, or C++. For example, the PIC32 series is supported by a set of tools, including the MPLAB IDE, MPLAB C32 C Compiler, and a set of peripheral libraries (Microchip, 2008). Libraries are provided for all the necessary interfaces, including GPIO, I2C, and UART. You simply need to include the appropriate header files (e.g., ports.h, i2c.h, uart.h) and link with the appropriate libraries. Another possibility is to utilize an open-source framework such as FreeRTOS, which provides common functionality for many microcontrollers (FreeRTOS, n.d).

**References**

Amazon. (n.d.). Wi-Fi library, FreeRTOS. Amazon.com. Retrieved April 19, 2025, from https://docs.aws.amazon.com/freertos/latest/userguide/freertos-wifi.html

FreeRTOS. (n.d.). Freertos.org. Retrieved April 19, 2025, from https://www.freertos.org/

Microchip. (2008). Getting Started with PIC32 User’s Guide. Microchip.com. Retrieved April 19, 2025, from https://ww1.microchip.com/downloads/en/DeviceDoc/61146B.pdf

Microchip (2025). PIC32MZ W1 MCU and WFI32 Module with Wi-Fi and Hardware-Based Security Accelerator Data Sheet. Microchip.com. Retrieved April 19, 2025, from https://ww1.microchip.com/downloads/aemDocuments/documents/WSG/ProductDocuments/DataSheets/PIC32MZ-W1-and-WFI32E01-Family-Data-Sheet-DS70005425.pdf

NXP. (2024). Wireless Microcontroller - IEEE 802.11n/g/b. Nxp.com. Retrieved April 19, 2025, from https://www.nxp.com/docs/en/data-sheet/88MW320-88W322.pdf

Raspberry Pi. (2024). DATASHEET, Raspberry Pi 4 Model B. Raspberrypi.com. Retrieved April 19, 2025, from https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

Robotic Backend. (2020, February 3). Raspberry Pi GPIO interrupts tutorial. The Robotics Back-End; https://roboticsbackend.com/raspberry-pi-gpio-interrupts-tutorial/