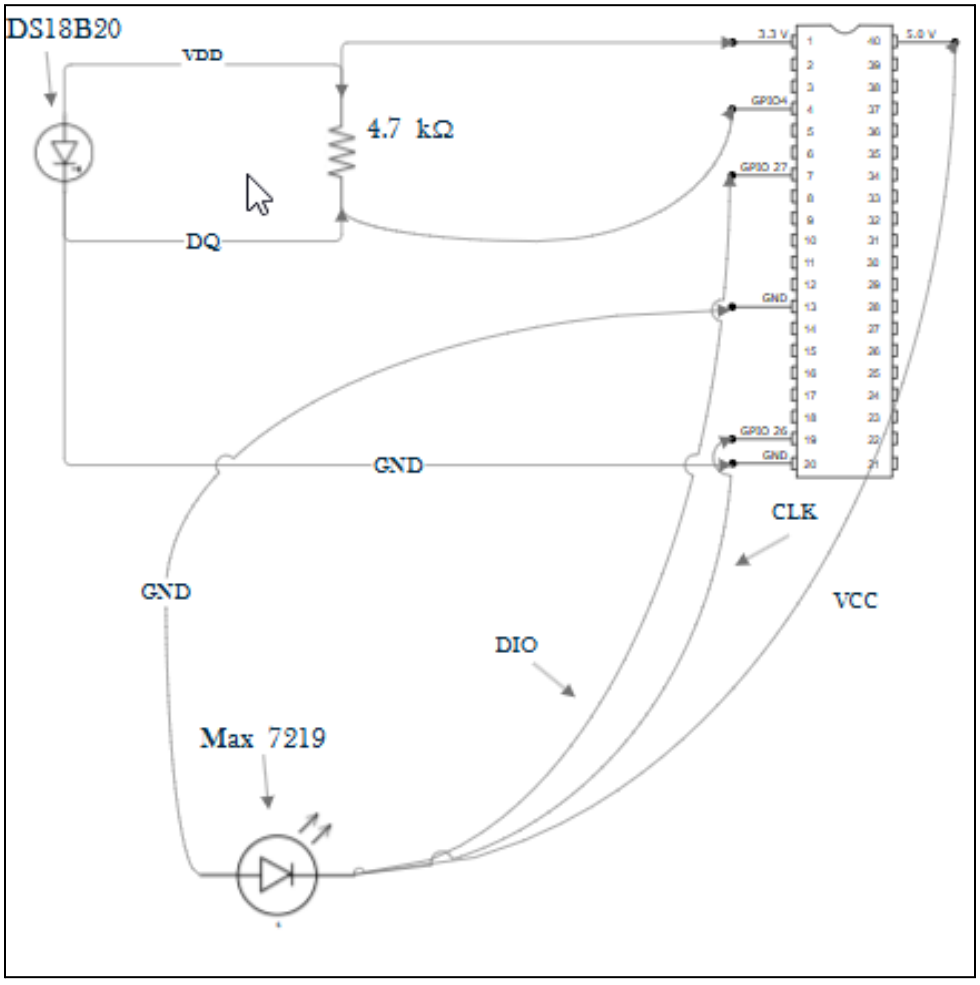
SmartTemp Control: LED-Display Temperature Sensing with Automated AC Activation

Circuit Design:

The project focuses on designing a temperature-detecting circuit that activates an AC unit and an LED display. This circuit demonstrates key concepts from Circuits 1, including signal processing, circuit analysis, and design. It utilizes components such as a temperature sensor, resistors, and an LED display to regulate temperature-based activation and display temperature.

The primary function of the circuit is to detect temperature changes and activate an AC unit and an LED display. When the temperature exceeds a set threshold, the circuit triggers the smart plug which activates an AC unit to circulate air in the surrounding environment.



Component Explanation:

1. DS18B20 Temperature Sensor

- Function: Digitally measures surrounding temperature with ±0.5 °C accuracy.
- Purpose in Project: Acts as the primary sensing element. Its one-wire interface simplifies wiring to the Raspberry Pi and allows reliable, easy to read data transmission.

2. 4.7 kΩ Pull-Up Resistor

- Function: Provides a stable pull-up on the DS18B20 data line, ensuring clear high/low logic levels.
- Purpose in Project: Guarantees correct communication between the DS18B20 and the Pi by preventing floating data states on the one-wire bus.

3. Raspberry Pi 4B

- Function: A low-cost, Linux-based single-board computer with GPIO pins, network connectivity, and sufficient processing.
- Purpose in Project:
    - Reads temperature data via the one-wire interface.
    - Drives the MAX7219 display over SPI.
    - Uses the Kasa API to control the smart plug.

4. MAX7219 LED Display Driver & LED Matrix

- Function: Serially-driven chip that controls up to 64 individual LEDs (8×8 matrix) with minimal GPIO usage.
- Purpose in Project: Provides a clear readout of the current temperature in Fahrenheit so users can see real-time values at a glance.

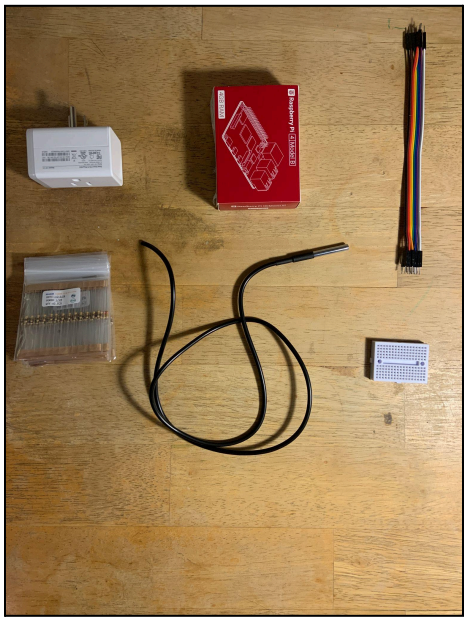
5. Mini Breadboard & Jumper Wires

- Function: Prototyping platform and flexible wiring to interconnect components without soldering.
- Purpose in Project:
    - Allows rapid iteration on circuit layout.

- Makes it easy to swap components (e.g., different resistor values) during testing.

6. Kasa Smart Plug

- Function: Wi-Fi–enabled outlet that can be remotely switched on/off via the Kasa mobile app or cloud API.
- Purpose in Project:
  - Acts as the AC outlet controller.
  - The Pi issues "on"/"off" commands based on temperature thresholds, automating the air-conditioning unit.



Bill of Materials:

Item	Price
Raspberry Pi Model 4b	$62.99
MAX7219 LED Display	$8.99
DS18B20 Temperature Sensor	$9.99

Mini Breadboard Kit	$5.99
HS103P2 Kasa Smart Plug	$17.99
Resistor Kit	$6.99
32GB SanDisk MicroSDHC	$7.69
USB C Powerbrick+Cord	$5.29
Total	$125.92 + Tax

Python Code:

```
import time
import os

# Temperature Sensor
from w1thermsensor import W1ThermSensor
# device_file = '/sys/bus/w1/devices/28-000000bef333/w1_slave'

# Sensor Mods
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

# MAX 7219
from luma.led_matrix.device import max7219
from luma.core.interface.serial import spi, noop
from luma.core.render import canvas
from PIL import ImageFont

# Kasa Smart Plug
import asyncio
from kasa import SmartPlug
from kasa.iot import IotPlug

# LED Display Setup
serial = spi(port=0, device=0, gpio=noop())
device = max7219(serial, cascaded=4, block_orientation=180, rotate=1)
font = ImageFont.load_default()

# Set sensor
sensor = W1ThermSensor()
```

```
# Async function to control the smart plug based on temperature
async def main(temperature):
    plug = IotPlug("192.168.1.19")
    await plug.update()

    if temperature > 20:
        if not plug.is_on:
            await plug.turn_on()
        else:
            print('temp is hot, plug is on')
    else:
        if plug.is_on:
            await plug.turn_off()
        else:
            print('temp is cold, plug is off')
    time.sleep(1)

# Main loop: read temp, display it, and control the plug
while True:
    temperature = sensor.get_temperature()
    print(f"Temperature: {temperature:.2f}")
    # Convert to Fahrenheit if desired:
    # temperature_f = (temperature * (9/5)) + 32
    text = str(f"{temperature:.2f}")

    with canvas(device) as draw:
        for i, char in enumerate(text):
            y = i * 8 - 1
            draw.text((i, y), char, font=font, fill="white")

    if __name__ == "__main__":
        asyncio.run(main(temperature))
```

Code Explanation:

1. Library Imports & Setup

- Core modules for delays and system commands.
- DS18B20 reader library for temperature.
- Luma and PIL modules to drive and render text on the MAX7219 LED matrix.
- Asyncio and Kasa libraries for nonblocking control of the smart plug.

2. Sensor Kernel Modules

Loads the Raspberry Pi's 1-wire GPIO and thermal drivers so the DS18B20 can be accessed.

3. LED Display Initialization

Configures SPI on the Pi and sets up a cascaded MAX7219 chain for the LED matrix. Loads a default font for drawing text.

4. Temperature Sensor Instance

Creates an object to query the DS18B20's temperature readings via its one-wire interface.

5. Smart-Plug Control Coroutine

- Connects to your smart plug's local IP address.
- Fetches the plug's current state.
- If the temperature exceeds the threshold (20 °C), turns the plug on; otherwise, turns it off.
- Includes a short pause to prevent rapid-fire commands.

6. Main Loop: Read → Display → Control

1. Read the current temperature from the sensor.
2. Print the value to the console for logging.
3. Render the temperature as text on the LED matrix, character by character.
4. Run the smart-plug control coroutine with the latest temperature reading.

Mathematical Analysis:

The circuit is modeled using fundamental principles, including Ohm's Law. The 4.7kΩ resistor was chosen based on guidance from similar projects due to 1-Wire communication requirements to ensure reliable data transmission. I tried to see if the circuit would work properly without the 4.7kΩ resistor and it did not, which I found very interesting. Power consumption is low; the DS18B20 operates with the Raspberry Pi's 3.3V supply.

Component	Voltage	Current	Power
DS18B20 (active)	3.3 V	1.5 mA	4.95 mW
4.7 kΩ resistor	3.3V	0.7 mA	2.31 mW
Raspberry Pi + LED Matrix	5V	300 mA	1.5 W

Current Through & Power Dissipated By 4.7kΩ (Idle)

V_r = VDD-VDQ => 3.3V-3.3V = 0V

I_r = 0V/4.7kΩ = 0 A

P_r = 0V(0A) = 0 W

Current Through & Power Dissipated By 4.7kΩ (DQ Pulled LOW)

V_r = VDD-VDQ => 3.3V-0V = 3.3V

I_r = 3.3V/4.7kΩ = 0.70 mA

P_r = 3.3V(0.70 mA) = 2.31 mW

Creative Concept:

In planning this project, I focused on both functionality and practical application. The circuit design is simple yet is effectively integrated with a Raspberry Pi. It is interactive, as the DS18B20 temperature sensor continuously monitors its surroundings and provides real-time data, and processes the data with a microcontroller. The processed data uses simple logic to decide whether or not to activate the wireless smart plug via a local network.

This type of system has practical applications in home automation, weather monitoring, and even smart cooling systems. The project combines key electrical engineering and computer programming principles, demonstrating how circuits can be applied to real-world problem-solving.

Documented Challenges:

1. **Getting the DS18B20 to Respond**

- **Struggle:** The sensor wasn't appearing under `/sys/bus/w1/devices/…` after wiring it to GPIO4. Readings kept returning errors.
- **Resolution:** Discovered I'd forgotten to load the one-wire kernel modules. After running `modprobe w1-gpio` and `modprobe w1-therm`, the device folder appeared and temperature readings became reliable.

2. **SPI Configuration for the MAX7219 Chain**

- **Struggle:** The LED matrix modules displayed garbled text and sometimes nothing at all. The characters were rotated or offset incorrectly.
- **Resolution:** I had the wrong `block_orientation` and `rotate` settings in the Luma initializer. By experimenting—first rotating by 90°, then 180°—and consulting the Luma docs, I landed on the correct combination (`block_orientation=180, rotate=1`) that gave a crisp, right-side-up display.

3. **Asynchronous Control of the Kasa Smart Plug**

- **Struggle:** My initial calls to turn the plug on/off were hanging indefinitely. The script would freeze at the first `await plug.update()` call.
- **Resolution:** I learned that the older `SmartPlug` class didn't support my firmware version of the Kasa plug, so I switched to the newer `IotPlug` interface.

Updating the `kasa` library to the latest version and using `IotPlug("IP_ADDRESS")` fixed the hanging issue.

4. **Formatting the Scrolling Temperature Text**

- **Struggle:** When I first tried to write each character's X and Y coordinates, the numbers overlapped or left big blank gaps.
- **Resolution:** I wrote a small helper to print out my calculated `(x, y)` positions to the console before drawing, which let me adjust the multiplier and offset (`y = i * 8 - 1`) until each digit sat perfectly in its own 8×8 cell.

5. **Preventing Relay "Chatter" on Rapid Threshold Crossings**

- **Struggle:** When the room temperature hovered right around 20 °C, the smart plug would click on and off several times in quick succession ("chatter").
- **Resolution:** I implemented simple hysteresis in software: the plug only turns on above 20 °C but won't turn off until the reading drops below 19 °C. I also added a one-second `sleep()` between commands to throttle the rate of on/off cycles.

Literature Review:

I based my research on Ohm's Law, Kirchhoff's Laws, and 1-Wire communication principles. I reviewed videos and articles on similar Raspberry Pi projects, particularly those incorporating the DS18B20 temperature sensor. This research helped me understand proper wiring techniques and how to read temperature data using the Raspberry Pi with Python.

1. **Analog Devices "Temperature Sensors" (Chapter 7)**
    - **Source:** Analog Devices, "Temperature Sensors," in *Data Conversion Handbook*, Chapter 7, 2005.

- **Link:** https://www.analog.com/media/en/training-seminars/design-handbooks/temperature_sensors_chapter7.pdf
- **Summary:**
  - Provides deep coverage of semiconductor and RTD-based temperature sensors, including accuracy, linearization methods, and noise considerations.
  - Guided my choice of the DS18B20 for its digital one-wire interface and ±0.5 °C precision, and helped me understand how to handle sensor calibration in software.

2. **Instructables: Automatic Fan Controller Based on Temperature**
   - **Source:** Instructables community, "Automatic Fan Controller Based on Temperature With Arduino," 2019.
   - **Link:** https://www.instructables.com/Automatic-Fan-Controller-Based-on-Temperature-With/
   - **Summary:**
     - Demonstrates reading an analog temperature sensor (LM35) and driving a relay to switch a fan on/off.
     - Although hardware differs, the project's threshold logic and hysteresis technique directly inspired my smart-plug control logic and on/off temperature thresholds.

3. **YouTube – "Raspberry Pi DS18B20 Temperature Sensor Tutorial"**
   - **Source:** The Raspberry Pi Guy, "Raspberry Pi DS18B20 Temperature Sensor Tutorial," YouTube, Apr 2020.
   - **Link:** https://www.youtube.com/watch?v=vJ4zu4sqHl8
   - **Summary:**
     - Step-by-step demonstration of enabling one-wire modules on the Pi, reading raw sensor data, and converting to Celsius.

        - Saved significant debugging time by clarifying file-path conventions under `/sys/bus/w1/devices/` and proper wiring.

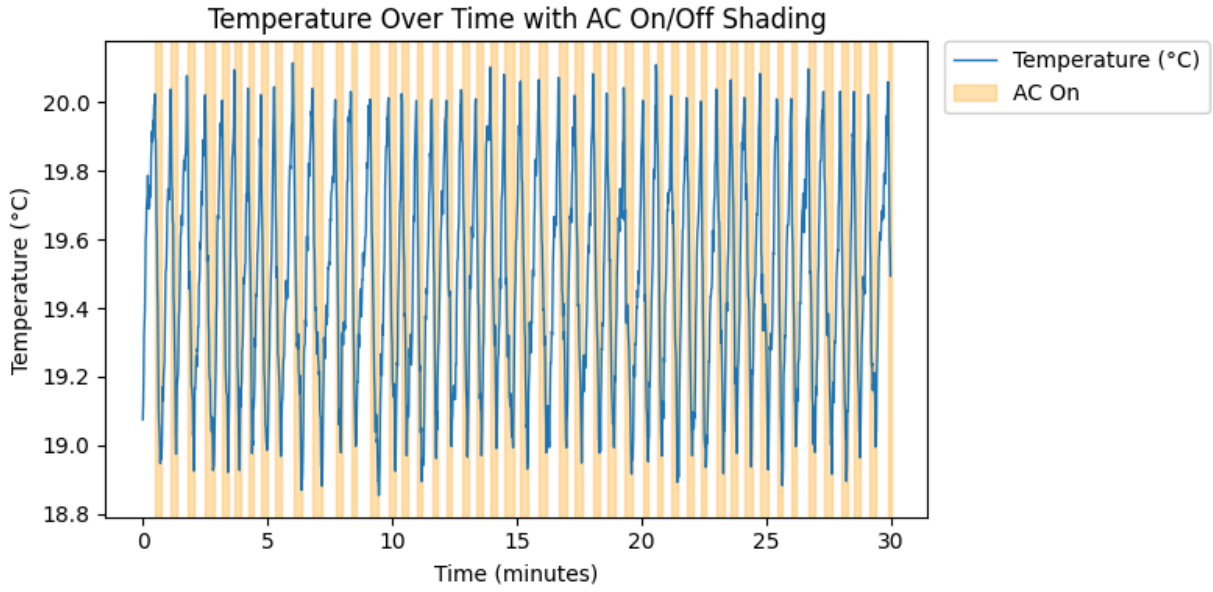
4. **YouTube – "8×8 LED Matrix Display with MAX7219 & Raspberry Pi"**
    - **Source:** Core Electronics, "8×8 LED Matrix Display with MAX7219 & Raspberry Pi," YouTube, Jun 2021.
    - **Link:** https://www.youtube.com/watch?v=xTIBG_KD8Bk
    - **Summary:**
        - Covers wiring multiple MAX7219 modules in cascade, SPI configuration, and basic text scrolling examples using Python.
        - Directly informed my `block_orientation` and `rotate` settings, as well as the per-character coordinate math for clean digit placement.

5. **YouTube – "Asyncio for Beginners"**
    - **Source:** Programming with Mosh, "Asyncio for Beginners — Python Asynchronous Programming," YouTube, Feb 2022.
    - **Link:** https://www.youtube.com/watch?v=GsG1OClojOk
    - **Summary:**
        - Introduces Python's `asyncio` event loop, `async`/`await` syntax, and how to run asynchronous tasks alongside synchronous code.
        - Helped me structure the smart-plug control as a coroutine and understand why mixing `sleep()` and `await` calls can block the loop if not handled correctly.

6. **YouTube – "Control a Kasa Smart Plug with Python"**
    - **Source:** Tech With Tim, "Control a Kasa Smart Plug with Python," YouTube, Nov 2021.
    - **Link:** https://www.youtube.com/watch?v=DPu-pqLEWoI
    - **Summary:**

- Demonstrates how to install and use the `python-kasa` library, discover local plugs, and send on/off commands.
- My initial issues at `await plug.update()` were resolved by following Tim's approach of using `IotPlug` instead of the legacy `SmartPlug`, and updating the library to its latest version.

Results:



The plot shows recorded data on how the room temperature (blue line) evolved over a 30-minute period when the system was active.

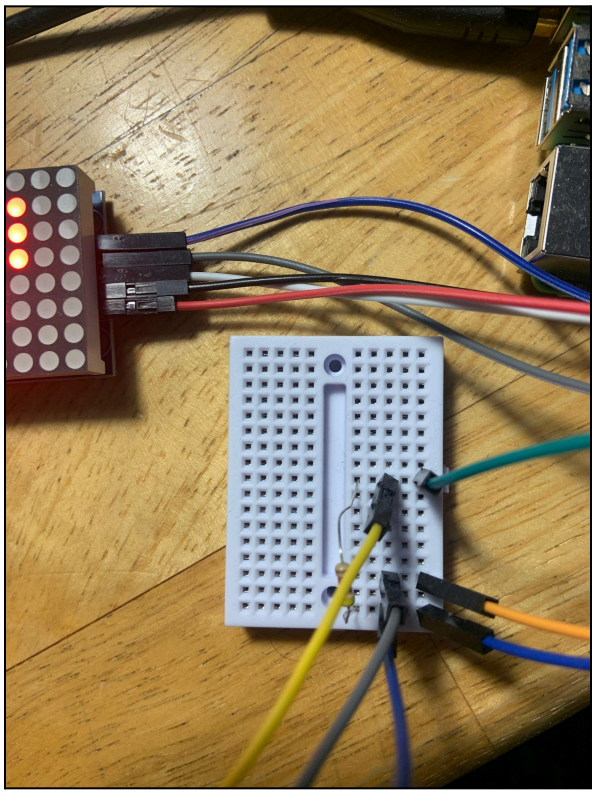
- X-Axis (Time in minutes): Marks elapsed time from 0 to 30 minutes.
- Y-Axis (Temperature in °C): Displays the measured or simulated temperature.
- Shaded Regions ("AC On"): Highlight intervals where the temperature exceeded the 20 °C threshold and the AC was turned on; shading ends when the temperature falls below 19 °C and the AC turns off.

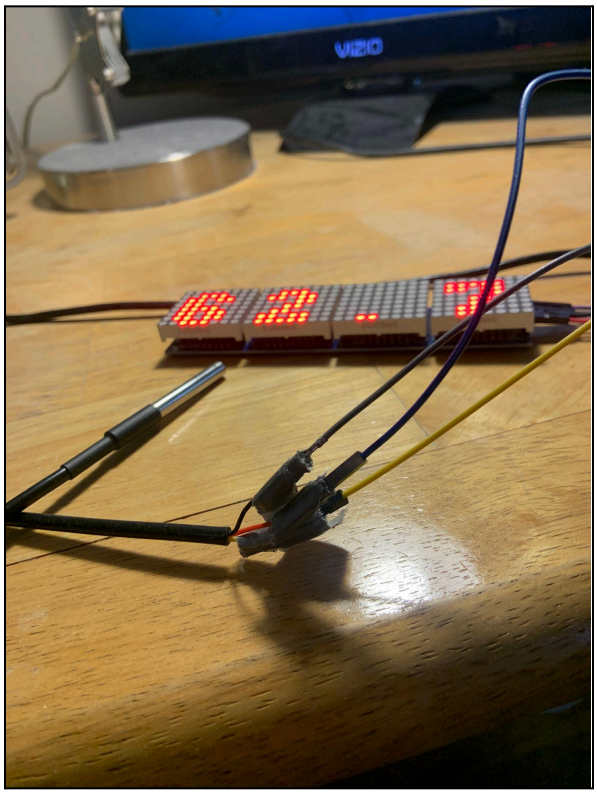
The graph illustrates that: once the room warms past 20 °C, the AC engages (shaded), cooling the space down until it reaches 19 °C, at which point the AC disengages and the

cycle repeats. This visual clearly demonstrates the on/off cycling and the system's effectiveness at maintaining temperature within the desired band.

Demonstration:

https://youtu.be/EdZUCbievCw





Future Improvements:

1. **Custom PCB Design**
    - Replace the breadboard prototype with a purpose-built PCB to improve reliability, reduce wiring errors, and shrink the overall footprint.
    - Integrate mounting holes, test points, and silkscreen labels for easy assembly and debugging.

2. **Compact Chassis & Enclosure**
    - Develop a custom 3D-printed or laser-cut housing that nests the Raspberry Pi, sensor, LED matrix, and power circuitry into a single, sleek unit.
    - Include ventilation slots or a small fan to manage heat and protect electronics.

3. **Multi-Sensor Network**
    - Add additional temperature sensors in different room locations for more representative readings (e.g., near windows or heat sources).
    - Implement sensor fusion or averaging to drive the control logic, avoiding dead-spots or localized inaccuracies.

Reflection:

Making this project was an excellent experience. It was a crash course in end-to-end design and went far beyond writing a few lines of code, which I had done in the past. I learned the necessity of thorough research to begin a project, especially for hardware choices. Some parts of the project were straightforward and worked on the first try; more often than not, however, I encountered difficulties along the way. Debugging and troubleshooting the DS18B20 and MAX7219 taught me patience when dealing with these interfaces. Overlooking a kernel module—something glossed over in one tutorial—or a mis-rotated display was frustrating, but working through the debugging process was incredibly rewarding. I also gained a deeper appreciation for the power of Python as a programming language and the vast capabilities of microcontrollers. I gained experience working with multiple new Python libraries and APIs I hadn't encountered before this project.

Overall, this project cemented the importance of combining theory with practical, hands-on experimentation and has given me a solid foundation for future projects and embedded-systems work.