Heater Control System – Part 2: Embedded Implementation

Platform

- **Simulation**: Wokwi (ESP32 board with I^2C device probe + firmware temperature scripting)
- **Real Hardware** (optional): ESP32 + TMP117 Digital Temperature Sensor (I^2C)

Language & Framework

- C (ESP-IDF 5.x)
- FreeRTOS (1 Hz periodic task for temperature monitoring & control)

Requirements Implementation

1) Temperature-Based State Tracker

States implemented:
- **Idle** — Ambient temperature 18–32 °C, stable for ≥3 readings → turn heater ON and enter Heating.
- **Heating** — Heater ON until temperature ≥48 °C.
- **Stabilizing** — Toggle at inclusive edges: heater ON at ≤48 °C, OFF at ≥52 °C. Stay in Stabilizing unless temp drops <46 °C (then return to Heating).
- **Target Reached** — 50 °C held for 5 s, heater OFF.
- **Overheat** — ≥60 °C → heater OFF (latched).
- **Fault** — Sensor read error or unrealistic temp (<18 °C or >80 °C). Auto-recovery restores a sensible state once readings normalize.

2) Continuous Temperature Reading

- **Real hardware**: TMP117 over I^2C.
- **Wokwi**: scripted temperature engine in firmware (keeps I^2C stack active by probing an I^2C device).

3) Heater ON/OFF Control

- **Heater ON**: GPIO output HIGH.
- **Heater OFF**: GPIO output LOW.
- In Wokwi, behavior is observed via logs.

4) Serial Logging (UART)

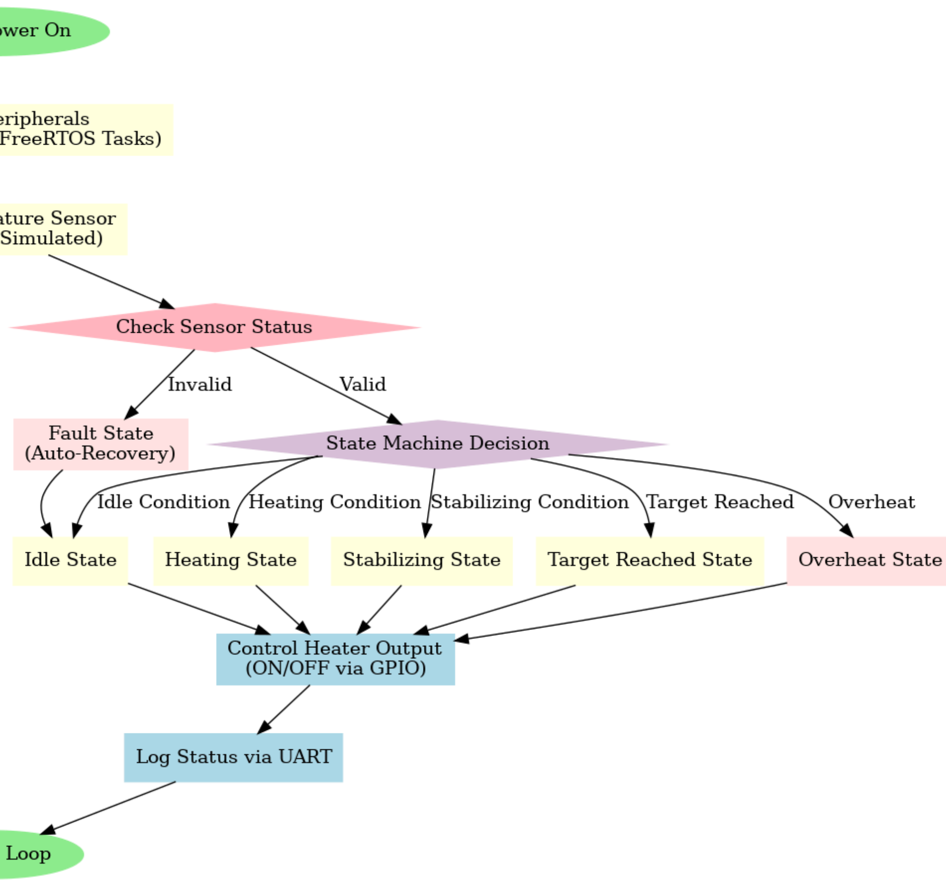
We log:
- **Current temperature (°C)**

- **Current state** (Idle / Heating / Stabilizing / Target Reached / Overheat / Fault)
- **Heater status** (ON/OFF)

Bonus Features Implemented

- **LED indicator** — ON while heating; OFF in other states.
- **FreeRTOS periodic task** — 1 Hz control loop using vTaskDelay().
- *Two simulation scripts (Wokwi):*
  - **Mode 1**: Functional system → stabilize → target (50 °C for 5 s).
  - **Mode 2**: Functional system with overheat triggered on the second upper 52 °C peak during the Stabilizing oscillation (then jumps to ~62 °C → Overheat).

Block Diagram



State Machine Implementation

Future Roadmap

1. Dual-sensor via I^2C for redundancy and spatial temperature awareness.
2. Multiple heating profiles (different setpoints, bands, and hold times).
3. Overheat interrupt using TMP117 ALERT → ESP32 GPIO ISR for faster cut-off.

Note (Simulation on Wokwi)

- **Temperature source**: No TMP117 model exists in Wokwi.
- **I²C activity**: We use an **MPU6050 accelerometer/gyroscope** as a placeholder I²C device, so the ESP-IDF I²C driver initializes and communicates with real I²C hardware in simulation.
- **Temperature generation**: Actual temperature values are scripted in firmware (mock engine) to follow required test patterns.
- **Runtime mode selection**: (Mode 1/Mode 2) done via UART prompt in Wokwi.
- **Real hardware**: Replace MPU6050 with TMP117, disable WOKWI_SIM in CMake, rebuild, and flash.

Simulation and Deployment Instructions

Running the Simulation:

1. Open the Wokwi project link: https://wokwi.com/projects/439005910473186305

2. Upload the heater_control.elf file from the /build directory of your ESP-IDF project.

https://github.com/vivekanandaramanu/heater_control_system/blob/main/build%2Fheater_control.elf

3. Click "**Start Simulation**".

4. Monitor the **UART logs** in the Wokwi Serial Monitor to observe system states.

5. To run ***Mode 1*** or ***Mode 2*** in simulation, choose the option via **UART prompt** when simulation starts.

Source Code RepositoryGitHub:
https://github.com/vivekanandaramanu/heater_control_systemELF file path:
https://github.com/vivekanandaramanu/heater_control_system/blob/main/build%2Fheater_control.elf

Testing Modes:

Mode	Description	Trigger Condition
1	Functional system with normal stabilization to target	Heater stabilizes at 50°C for 5 seconds
2	Functional system with overheat on second upper stabilization cycle	Heater jumps to ~62°C after 2nd peak at 52°C

Running on Real Hardware

1. Connect ESP32 to TMP117 via I²C.

2. In CMakeLists.txt, remove the WOKWI_SIM definition.

3. Rebuild and flash using ESP-IDF: idf.py build flash monitor

Note:

Why an I^2C Gyroscope (MPU6050) Appears in the Wokwi Schematic

Context

On real hardware, the ESP32 reads temperature from a TMP117 digital sensor over I^2C and controls the heater via GPIO. In Wokwi, the TMP117 part isn't available, so we cannot place the real temperature sensor in the simulator.

The Problem in Simulation

If we remove/skip I^2C completely and just "fake" temperatures in code, we never exercise:

• ESP-IDF I^2C driver initialization

• The I^2C controller configuration and pins

• The code paths that would normally deal with an I^2C device on the bus

That means the simulation wouldn't represent the real firmware conditions — risking surprises on actual hardware.

The Practical Workaround

We place an MPU6050 (accelerometer/gyroscope) on the I^2C bus in Wokwi as a placeholder I^2C device because:

• It's available in Wokwi's part library and easy to wire (default address 0x68)

• It gives the ESP32 a real I^2C target so driver init and (optional) probing work

• It keeps the I^2C stack active during simulation

Note: We do not use MPU6050 data for control. Temperature values in Wokwi are generated by our firmware mock engine under the WOKWI_SIM build flag.

How It Fits the Build

• Simulation build: WOKWI_SIM is enabled.

- The code initializes I^2C and (depending on your version) either probes a known device or simply returns OK for probe calls.

- Temperature data comes from the mock temperature script (Mode 1 / Mode 2).

• Real hardware build: WOKWI_SIM is disabled.

- The ESP32 talks to TMP117 at 0x48, reads real temperature over I^2C, and controls the heater.

Toggle example (CMakeLists.txt):

```
# For Wokwi simulation
 add_compile_definitions(WOKWI_SIM=1)

 # For real hardware (comment out the line above or remove WOKWI_SIM)
```