PROJECT :

PID based Temperature Control System

System Requirements Recap:

1. Controller: ESP32
2. Sensor: DHT11 (Temperature + Humidity, but we will use Temperature only for PID)
3. Display: 16x2 HD44780 LCD in **4-bit mode**
4. Actuators:
    - **220V incandescent bulb** (Heating element) via **electromechanical relay**
    - **12V DC fan** (Cooling element) via **electromechanical relay**
5. Control Algorithm: **PID**
6. Software Environment: **ESP-IDF (Espressif IDE) with FreeRTOS**
7. Mode: Automatic temperature regulation to a setpoint (e.g., 40°C)
8. Objective: Achieve proper thermal control and stability

SECTION 1 — SYSTEM OVERVIEW

We want the system to maintain a target temperature. When temperature < setpoint, heater ON. When temperature > setpoint, fan ON. Instead of bang-bang control, we use PID so that the heating power is modulated in time to reduce overshoot and oscillations.

However, since actuators are relays (ON/OFF), PID output cannot directly drive analog power. We need a strategy:

Recommended Control Strategy: Time-Proportioning Control (TPC)

PID output → duty cycle (0–100%) → relay ON/OFF based on time window

Example:

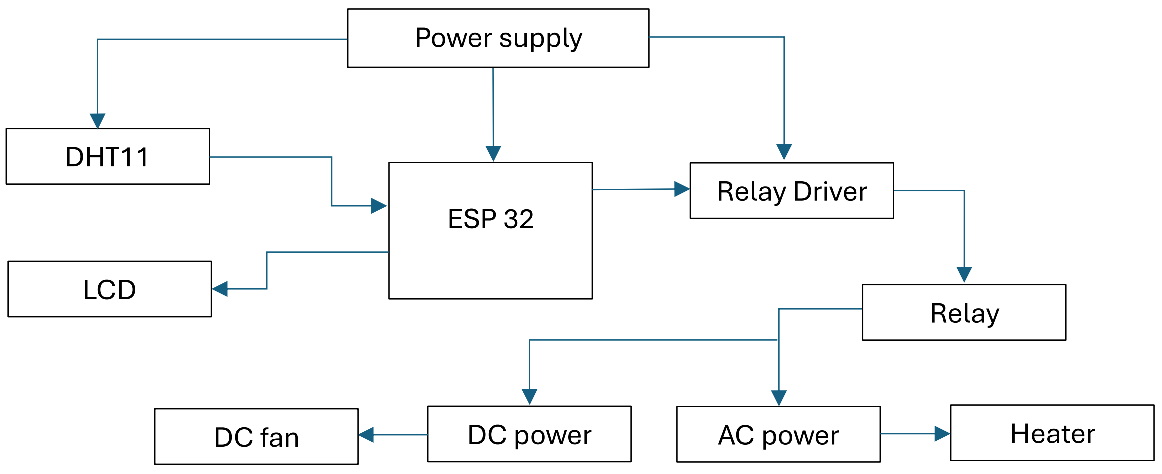
- Compute PID output every second
- Convert PID output to heating window ON time
- Use a 5-second window:
  e.g., PID output = 60% → Heater ON for 3 seconds, OFF for 2 seconds

Fan control is simpler:

- If temperature is above setpoint + threshold, run fan full

SECTION 2 — HARDWARE BLOCK DIAGRAM

Components and Signals:



[DHT11] → GPIO (Input)
[LCD 16x2] → GPIO (4-bit data + RS + EN)
[Heater Relay] → GPIO (Output)
[Fan Relay] → GPIO (Output)
[ESP32] → Central Controller running FreeRTOS + PID

Power Levels:

- DHT11: 5V or 3.3V depending model
- LCD: typically 5V logic
- Relays: Coil may require 5V or 12V relay module
- Bulb: 220V AC through relay
- Fan: 12V DC through relay

SECTION 3 — HARDWARE WIRING DETAILS

3.1 DHT11 Wiring

- VCC → 5V
- GND → GND
- Data → GPIO 4 (example)
- 10k pull-up from Data → VCC

3.2 LCD 16x2 in 4-bit Mode

Assuming the following mapping:

Pins:
LCD RS → GPIO 14
LCD EN → GPIO 27
LCD D4 → GPIO 26
LCD D5 → GPIO 25
LCD D6 → GPIO 33
LCD D7 → GPIO 32

Plus:

- VSS → GND
- VDD → 5V
- V0 → middle pin of 10k pot for contrast
- RW → GND (write mode)
- LED+ → 5V
- LED- → GND

3.3 Relays

Use relay modules with optocoupler isolation if possible.

Heater Relay:

- Relay IN → GPIO 18
- COM → AC Live input
- NO → Bulb Live input
- Neutral directly to bulb neutral
- Coil supply → 5V relay supply
- GND common with ESP32

Fan Relay:

- Relay IN → GPIO 19
- COM → 12V supply +
- NO → Fan +
- Fan – → 12V supply –
- Coil supply → 5V relay supply
- GND common with ESP32

Important Safety Notes:

1. Never handle 220V AC while power is on.
2. Use insulated connectors.

3. Use proper fuse and MCB.
4. Relays must be rated ≥ 230VAC 5A for bulbs.

SECTION 4 — CONTROL ALGORITHM & LOGIC

4.1 PID Formulation

Error = Setpoint – Measured_Temperature

PID Output = Kp*error + Ki*∫error dt + Kd*d(error)/dt

Output range selected: 0–100%

Mapping output:
0% → Heater OFF
100% → Heater ON full

4.2 Time-Proportioning Window

Choose window = 5000 ms (5 seconds)

Heater ON time = (PID_Output/100)*5000 ms

Relay behavior in each window:

- Measure, compute output
- Start window timer
- If t < ON_time → Heater ON
- Else → Heater OFF

4.3 Fan Logic

If current_temp > setpoint + 2°C:
Fan ON
Else:
Fan OFF

(Deadband avoids relay chatter)

SECTION 5 — FREERTOS TASK DESIGN

Task Definitions

1. Task_DHT_Read (Priority 4)
    - Reads sensor periodically (2 sec)
    - Stores value in global struct (protected via mutex)
2. Task_PID_Control (Priority 5)
    - Runs every 1 sec
    - Reads global temperature
    - Computes PID output
    - Stores PID output & ON/OFF times
3. Task_Heater_Control (Priority 3)
    - Time proportioning relay drive
    - Executes 100ms loop
4. Task_Fan_Control (Priority 2)
    - Simple ON/OFF control based on temperature
5. Task_LCD_Update (Priority 1)
    - Updates display every 1 sec
6. Watchdog (Optional)

Inter-Task Communication

- Use a global data structure + mutex
- No queues needed for basic build

SECTION 6 — SOFTWARE IMPLEMENTATION (ESP-IDF)

Folder Structure Example:

```
project/
├── main/
│   ├── main.c
│   ├── dht11.c / dht11.h
│   ├── lcd.c / lcd.h
│   ├── pid.c / pid.h
│   └── relay.c / relay.h
└── CMakeLists.txt
```

SECTION 7 — CODE IMPLEMENTATION (REFERENCE)

7.1 Global Structures and Mutex

```
typedef struct {
   float temperature;
   float humidity;
} sensor_data_t;

static sensor_data_t g_sensor;
static float g_setpoint = 40.0;
static float g_pid_output = 0.0;
static SemaphoreHandle_t xSensorMutex;
```

7.2 PID Logic (pid.c)

```
typedef struct {
   float Kp, Ki, Kd;
   float prev_error;
   float integral;
} pid_t;

void pid_init(pid_t *pid, float Kp, float Ki, float Kd) {
   pid->Kp = Kp;
   pid->Ki = Ki;
   pid->Kd = Kd;
   pid->prev_error = 0;
   pid->integral = 0;
}

float pid_compute(pid_t *pid, float setpoint, float measured, float dt) {
   float error = setpoint - measured;
   pid->integral += error * dt;
   float derivative = (error - pid->prev_error) / dt;
   pid->prev_error = error;

   float output = pid->Kp*error + pid->Ki*pid->integral + pid->Kd*derivative;
   if(output > 100) output = 100;
   if(output < 0) output = 0;
   return output;
}
```

Recommended initial PID tuning:

- Kp = 3.0
- Ki = 0.5
- Kd = 1.0

Adjust after thermal testing.

7.3 DHT Task

```
void task_dht(void *pvParameters) {
  while(1) {
    float t, h;
    if(dht11_read(&t, &h) == ESP_OK) {
      xSemaphoreTake(xSensorMutex, portMAX_DELAY);
      g_sensor.temperature = t;
      g_sensor.humidity = h;
      xSemaphoreGive(xSensorMutex);
    }
    vTaskDelay(pdMS_TO_TICKS(2000));
  }
}
```

7.4 PID Task

```
void task_pid(void *pvParameters) {
  static pid_t pid;
  pid_init(&pid, 3.0, 0.5, 1.0);
  TickType_t last_wake = xTaskGetTickCount();

  while(1) {
    xSemaphoreTake(xSensorMutex, portMAX_DELAY);
    float temp = g_sensor.temperature;
    xSemaphoreGive(xSensorMutex);

    g_pid_output = pid_compute(&pid, g_setpoint, temp, 1.0);
    vTaskDelayUntil(&last_wake, pdMS_TO_TICKS(1000));
  }
}
```

7.5 Heater (Relay) Control Task with Time Window

```
void task_heater(void *pvParameters) {
  const int window_ms = 5000;
  TickType_t last_wake = xTaskGetTickCount();

  while(1) {
    float on_time = (g_pid_output/100.0) * window_ms;
    TickType_t start = xTaskGetTickCount();

    if(on_time > 0)
      gpio_set_level(HEATER_GPIO, 1);

    while((xTaskGetTickCount() - start) < pdMS_TO_TICKS(on_time)) {
      vTaskDelay(pdMS_TO_TICKS(100));
    }

    gpio_set_level(HEATER_GPIO, 0);

    while((xTaskGetTickCount() - start) < pdMS_TO_TICKS(window_ms)) {
      vTaskDelay(pdMS_TO_TICKS(100));
    }
```

```
        vTaskDelayUntil(&last_wake, pdMS_TO_TICKS(0));
    }
}
```

7.6 Fan Task

```
void task_fan(void *pvParameters) {
    while(1) {
        xSemaphoreTake(xSensorMutex, portMAX_DELAY);
        float temp = g_sensor.temperature;
        xSemaphoreGive(xSensorMutex);

        if(temp > g_setpoint + 2) {
            gpio_set_level(FAN_GPIO, 1);
        } else {
            gpio_set_level(FAN_GPIO, 0);
        }
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

7.7 LCD Task (4-bit mode updates)

Typical approach:

- Show Temperature, Setpoint, PID%

Example:

```
Line1: T:30.2C SP:40C
Line2: OUT: 65%
```

```
void task_lcd(void *pvParameters) {
    char buf[16];
    while(1) {
        float t;
        xSemaphoreTake(xSensorMutex, portMAX_DELAY);
        t = g_sensor.temperature;
        xSemaphoreGive(xSensorMutex);

        lcd_clear();
        snprintf(buf, 16, "T:%.1f SP:%.0f", t, g_setpoint);
        lcd_puts(0,0, buf);

        snprintf(buf, 16, "OUT:%3.0f%%", g_pid_output);
        lcd_puts(0,1, buf);

        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

SECTION 8 — SYSTEM BRING-UP PROCEDURE

Step-by-step hardware bring-up:

1. Verify ESP32 logic powering.
2. Verify LCD power + contrast adjust.
3. Run basic LCD “Hello” test.
4. Run DHT11 read-only test.
5. Add relay GPIO test using dummy loads (e.g., LED lamp).
6. Integrate PID code but log only, do not drive relays yet.
7. Connect bulb and fan through relays.
8. Test heater only for setpoint < ambient.
9. Tune PID parameters to reduce overshoot.

SECTION 9 — PID TUNING GUIDE (THERMAL SYSTEM)

Thermal systems are slow → Derivative helps reduce overshoot.

Basic method:

1. Set Ki=0, Kd=0 → Increase Kp until slight overshoot observed.
2. Add Ki until steady-state error eliminated.
3. Add small Kd to suppress overshoot.

Example good stable values:

- Kp: 3.0 to 7.0
- Ki: 0.3 to 0.7
- Kd: 1.0 to 3.0
