★ **Code for Creating own library for implementing temperature control sensor**

OneWireCustom.h - Header File

- **Import Library**
  - Include `Arduino.h` to access basic Arduino functions.
- **Set Up Constants**
  - `ONE_WIRE_PIN`: The pin used for OneWire communication.
  - `DEVICE_DISCONNECTED_C`: Error code used if the temperature sensor is disconnected.
- **Declare Functions**
  - `reset()`: Resets the OneWire bus to start communication.
  - `writeBit(bit)`: Sends a single bit (1 or 0) to the device.
  - `readBit() -> bit`: Reads a single bit from the device.
  - `writeByte(byte)`: Sends a full byte (8 bits) of data to the device.
  - `readByte() -> byte`: Reads a full byte (8 bits) of data from the device.
  - `readTemperature() -> temperature`: Gets and returns the temperature from the sensor.

OneWireCustom.cpp - Implementation File

- **Function: reset()**
  - Set the `ONE_WIRE_PIN` to output mode and pull it low, starting the reset.
  - Switch `ONE_WIRE_PIN` back to input mode to release the line.
  - Wait briefly to allow connected devices to respond.
- **Function: writeBit(bit)**
  - Pull `ONE_WIRE_PIN` low to begin sending a bit.
  - If `bit` is 1:
    - Quickly release the line to indicate a '1'.
    - Wait to complete the rest of the bit's timing.
  - If `bit` is 0:
    - Keep `ONE_WIRE_PIN` low longer to indicate a '0'.
    - Then release the line.
- **Function: readBit() -> bit**
  - Pull `ONE_WIRE_PIN` low briefly to start reading a bit.
  - Release the line and wait a moment before reading the bit's value.
  - Return the read bit (either 1 or 0).
- **Function: writeByte(byte)**

- For each of the 8 bits in `byte`:
  - Send each bit using `writeBit()`.
  - Shift `byte` to get the next bit ready to send.

- **Function: readByte() -> byte**
  - Start with `data` set to 0.
  - For each of the 8 bits:
    - Read a bit and set it in `data`.
  - Return the byte (8 bits) stored in `data`.
- **Function: readTemperature() -> temperature**
  - Reset the bus with `reset()`.
  - Send commands to start temperature measurement.
  - Wait while the measurement completes.
  - Reset the bus again and send commands to read the temperature.
  - Read two bytes representing the temperature.
  - Combine these bytes, convert to Celsius, and return the result.

Main Sketch (main.ino)

- **setup()**
  - Start serial communication (for debugging) at 9600 baud rate.
  - Set `ONE_WIRE_PIN` as `INPUT_PULLUP` to prepare the sensor.
- **loop()**
  - Call `readTemperature()` to get the temperature.
  - If temperature reading is valid (not equal to `DEVICE_DISCONNECTED_C`):
    - Print the temperature on the serial monitor.
  - If the reading fails:
    - Print an error message.
  - Wait 2 seconds before the next reading to avoid excessive polling.

★ Final Code (Pseudo code)

Import Necessary Libraries

- Import `CustomOneWireLibrary` for handling temperature sensor data.
- Import `PIDControlLibrary` to apply PID control for maintaining desired temperature.
- Import `DisplayLibrary` (like `Adafruit_SSD1306` and `Adafruit_GFX`) to control the OLED display and show temperature readings.
- Import `SPI.h` and `Wire.h` for communication protocols that enable OLED display operation.

Define Pin Connections and Initial Settings

- **Temperature Sensor Pin**: Define the digital pin that connects to the data line of the temperature sensor.
- **Rotary Switch Pins**: Define the pins for each of the 4 rotary switch positions (each position represents a specific temperature setting).
- **OLED Display Dimensions**: Set the width and height of the display; also define the reset pin for the OLED.
- **Peltier Control Pin**: Define the pin to control the Peltier module, which manages heating and cooling.
- **Pump Control Pin**: Define the pin for the pump, which will activate if the temperature differs significantly from the setpoint.

Set PID Constants and Initial Target Temperature

- **PID Constants**: Define `kp`, `ki`, and `kd` constants to tune the PID control.
- **Initial Target Temperature**: Set a default target temperature, like `25°C`, which can be adjusted by the rotary switch.

Setup Function (Runs Once at the Start)

- **Initialize PID Settings**: Use `PIDControlLibrary` to set up PID parameters with the initial values.
- **Start Serial Communication**: Begin serial communication for debugging purposes (to check outputs on the serial monitor).
- **Configure Rotary Switch Pins**:
  - Set each rotary switch pin as an input to read the user's selected temperature setpoint.
- **Initialize OLED Display**:
  - Begin the display with `DisplayLibrary`.
  - Check if the display is connected and working.
  - Clear the screen so it's ready for new data.
- **Configure Control Pins**:
  - Set the Peltier pin as an output and initially turn it off.
  - Set the pump pin as an output, also starting in the off position.

Main Loop (Repeats Continuously)

- **Read Rotary Switch to Set Target Temperature**:
  - Check each rotary switch position pin to see if it's active.
  - If position 1 is active, set the temperature target (`setpoint`) to 10°C.
  - If position 2 is active, set the temperature target (`setpoint`) to 15°C.
  - If position 3 is active, set the temperature target (`setpoint`) to 20°C.
  - If position 4 is active, set the temperature target (`setpoint`) to 25°C.

- Print the selected target temperature to the serial monitor for verification.

- **Read Temperature from Sensor**:
  - Use `CustomOneWireLibrary` to get the current temperature from the sensor and store it in a variable `T`.
  - Print the current temperature to the serial monitor for debugging.
- **Display Temperature on OLED**:
  - Clear the OLED display to remove previous data.
  - Display the text "Temp:" followed by the current temperature (`T`) on the screen.
- **Calculate PID Output for Temperature Control**:
  - Record the current time and calculate the time difference since the last reading.
  - Map the actual temperature `T` and target temperature `setpoint` to a 0–100 scale (for consistent PID control).
  - Calculate the error between the target temperature and the actual temperature.
  - Use the `pid()` function from `PIDControlLibrary` to compute the output based on the error.
  - Map the PID output to a range suitable for controlling the Peltier module.
  - Apply the PID output to the Peltier module using `analogWrite`.
- **Show Debug Information**:
  - Print the setpoint, actual temperature, and error values to the serial monitor to help verify the system's performance.
- **Control Pump Based on Temperature Error**:
  - If the temperature error is non-zero, turn the pump on for 5 seconds to help adjust temperature.
  - If the temperature is close to the target, turn the pump off.
  - Wait for 5 seconds between each pump operation to avoid rapid cycling.

Helper Functions

- **PID Calculation Function (`pid`)**:
  - Calculate each part of the PID output:
    - **Proportional**: This is the current error.
    - **Integral**: Sum of all past errors (helps eliminate steady-state errors).
    - **Derivative**: Rate of change of the error (predicts future error).
  - Return the combined PID output.
- **Temperature Mapping Function (`mapT`)**:
  - Map temperature range from -55°C to 125°C to a standard 0–100 scale for consistency in calculations.
  - Return the mapped value to use in the PID calculations.

Codes used in this Project

Testing of Components

★ Code for reading Temperature Sensor Output in Serial Monitor

```
#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is connected to GPIO pin (e.g., PA0)
#define ONE_WIRE_BUS PA0

// Setup a oneWire instance to communicate with DS18B20
OneWire oneWire(ONE_WIRE_BUS);

// Pass the oneWire reference to DallasTemperature library
DallasTemperature sensors(&oneWire);

void setup() {
  // Start the Serial communication
  Serial.begin(115200);  // Set the baud rate
  while (!Serial);  // Wait for the Serial Monitor to open

  // Start the DS18B20 sensor
  sensors.begin();
}

void loop() {
  // Request temperature from the sensor
  sensors.requestTemperatures();

  // Fetch the temperature in Celsius
  float temperatureC = sensors.getTempCByIndex(0);

  // Print the temperature to the Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temperatureC);
```

```
  Serial.println(" °C");

  // Delay before next reading
  delay(1000);  // Read every second
}
```

★ Code for displaying Temperature Sensor in OLED Display

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <OneWire.h>
#include <DallasTemperature.h>
// Data wire is connect to the Arduino digital pin 4
#define ONE_WIRE_BUS 4
// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(ONE_WIRE_BUS);
// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL
pins)
#define OLED_RESET     4 // Reset pin # (or -1 if sharing Arduino
reset pin)
Adafruit_SSD1306 _display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

void setup() {
  Serial.begin(9600);
  sensors.begin();
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V
internally
  if(!_display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C
for 128x32
    Serial.println(F("SSD1306 allocation failed"));
```

```
    for(;;); // Don't proceed, loop forever
  }

  _display.clearDisplay();
}

void loop() {
  // Call sensors.requestTemperatures() to issue a global
temperature and Requests to all devices on the bus
  sensors.requestTemperatures();

  Serial.println("Celsius temperature: ");
  // Why "byIndex"? You can have more than one IC on the same bus. 0
refers to the first IC on the wire
  Serial.print(sensors.getTempCByIndex(0));

  delay(1000);
  float T = sensors.getTempCByIndex(0);  // let T be temperature in
degC from sensor
          // floating-point number, with a decimal point
  // On each loop, we'll want to clear the display so we're not
writing over
  // previously drawn data
  _display.clearDisplay();

  int16_t x, y;
  uint16_t textWidth, textHeight;
  const char strHello[] = "Hello Vaish!";

  // Setup text rendering parameters
  _display.setTextSize(1);
  _display.setTextColor(WHITE, BLACK);

  // Measure the text with those parameters
  _display.getTextBounds(strHello, 0, 0, &x, &y, &textWidth,
&textHeight);

  // Center the text on the display
  _display.setCursor(_display.width() / 2 - textWidth / 2,
_display.height() / 2 - textHeight / 2);
```

```
  _display.print("Temp:");
  _display.print(T);

  // Render the graphics buffer to screen
  _display.display();

  delay(500);
}
```

★ Code for Rotary Switch

```
#define POSITION_1_PIN 2
#define POSITION_2_PIN 3
#define POSITION_1_PIN 5
#define POSITION_2_PIN 6

void setup()  {
  pinMode(POSITION_1_PIN,INPUT_PULLUP);
  pinMode(POSITION_2_PIN,INPUT_PULLUP);
  pinMode(POSITION_3_PIN,INPUT_PULLUP);
  pinMode(POSITION_4_PIN,INPUT_PULLUP);
  pinMode(13,OUTPUT);
  digitalWrite(13,LOW);

  Serial.begin(9600);
}

void loop()  {
  int switch_position_1_status=digitalRead(POSITION_1_PIN);
  int switch_position_2_status=digitalRead(POSITION_2_PIN);
  int switch_position_3_status=digitalRead(POSITION_3_PIN);
  int switch_position_4_status=digitalRead(POSITION_4_PIN);

  if (switch_position_1_status==LOW)  {
    Serial.println("switch in position 1");
  }
  else if (switch_position_2_status==LOW)  {
    Serial.println("switch in position 2");
  }

  else if (switch_position_3_status==LOW)  {
```

```
    Serial.println("switch in position 3");
  }

  else if (switch_position_4_status==LOW)  {
    Serial.println("switch in position 4");
  }



  delay(1000);
}
```

★ Code for Peltier Module and Temperature Sensor

```
#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is plugged into pin 2 on the Arduino
#define ONE_WIRE_BUS 2
#define PELTIER_PIN 9

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

void setup() {
  // Start serial communication for debugging
  Serial.begin(9600);

  // Start the DS18B20 sensor
  sensors.begin();

  // Set the Peltier control pin as output
  pinMode(PELTIER_PIN, OUTPUT);
}

void loop() {
  // Request temperature from the DS18B20
  sensors.requestTemperatures();
  float temperatureC = sensors.getTempCByIndex(0);
```

```
  // Print the temperature to the Serial Monitor
  Serial.print("Current Temperature: ");
  Serial.print(temperatureC);
  Serial.println("°C");

  // Define a setpoint for cooling (e.g., 25°C)
  float setpoint = 25.0;

  // Control the Peltier module based on temperature
  if (temperatureC > setpoint) {
    // If temperature is above the setpoint, turn ON the Peltier
    digitalWrite(PELTIER_PIN, HIGH);
    Serial.println("Peltier ON");
  } else {
    // If temperature is below the setpoint, turn OFF the Peltier
    digitalWrite(PELTIER_PIN, LOW);
    Serial.println("Peltier OFF");
  }

  // Wait for 1 second before the next reading
  delay(1000);
}
```

★ Code for Circulation Pump

```
#define PUMP_PIN 9

void setup() {
  // Initialize the pump control pin as an output
  pinMode(PUMP_PIN, OUTPUT);
}

void loop() {
  // Turn the pump ON
  digitalWrite(PUMP_PIN, HIGH);
  delay(5000);  // Pump stays on for 5 seconds

  // Turn the pump OFF
  digitalWrite(PUMP_PIN, LOW);
```

```
        delay(5000);  // Pump stays off for 5 seconds
      }
```

➔ Final Code

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <OneWire.h>
#include <DallasTemperature.h>


// Data wire is connected to the Arduino digital pin 4
#define ONE_WIRE_BUS 4
// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(ONE_WIRE_BUS);
// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL
pins)
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino
reset pin)
Adafruit_SSD1306 _display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

#define POSITION_1_PIN 2
#define POSITION_2_PIN 3
#define POSITION_3_PIN 5
#define POSITION_4_PIN 6
#define PELTIER_PIN A0

// constants for pid control
double dt, last_time;
double integral, previous, output = 0;
double kp, ki, kd;
```

```
double setpoint=25;

void setup()
{
  //Defining constants for PID
  kp = 0.8;
  ki = 0.20;
  kd = 0.001;
  last_time = 0;

  Serial.begin(9600);

  // Code to get setpoint input from the rotary switch
    pinMode(POSITION_1_PIN,INPUT_PULLUP); //
    pinMode(POSITION_2_PIN,INPUT_PULLUP);
    pinMode(POSITION_3_PIN,INPUT_PULLUP);
    pinMode(POSITION_4_PIN,INPUT_PULLUP);

  // code for OLED display
  sensors.begin();
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V
internally
  if(!_display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C
for 128x32
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  _display.clearDisplay();

  // Set the Peltier control pin as output
  pinMode(PELTIER_PIN, OUTPUT);
  digitalWrite(PELTIER_PIN, 0);
  pinMode(PUMP_PIN, OUTPUT);

}

void loop()
{
```

```
  // Code to get setpoint input from the rotary switch
    int switch_position_1_status=digitalRead(POSITION_1_PIN);
    int switch_position_2_status=digitalRead(POSITION_2_PIN);
    int switch_position_3_status=digitalRead(POSITION_3_PIN);
    int switch_position_4_status=digitalRead(POSITION_4_PIN);

    if (switch_position_1_status==LOW)  {
      Serial.println("Set point 10 celsius");
      float setpoint=10;
    }
    else if (switch_position_2_status==LOW)  {
      Serial.println("Set point 15 celsius");
      float setpoint=15;
    }
    else if (switch_position_3_status==LOW)  {
      Serial.println("Set point 20 celsius");
      float setpoint=20;
    }
    else if (switch_position_4_status==LOW)  {
      Serial.println("Set point 25 celsius");
      float setpoint=25;
    }
  // Call sensors.requestTemperatures() to issue a global
temperature and Requests to all devices on the bus
  sensors.requestTemperatures();

  Serial.println("Celsius temperature: ");
  // Why "byIndex"? You can have more than one IC on the same bus. 0
refers to the first IC on the wire
    float T=sensors.getTempCByIndex(0);
    Serial.println("Temp:");
  Serial.println(T);

  // let T be temperature in degC from sensor
  // floating-point number, with a decimal point
  // On each loop, we'll want to clear the display so we're not
writing over
  // previously drawn data
  _display.clearDisplay();
```

```
  int16_t x, y;
  uint16_t textWidth, textHeight;
  const char strHello[] = "Hello Vaishnavi!";

  // Setup text rendering parameters
  _display.setTextSize(1);
  _display.setTextColor(WHITE, BLACK);

  // Measure the text with those parameters
  _display.getTextBounds(strHello, 0, 0, &x, &y, &textWidth,
&textHeight);

  // Center the text on the display
  _display.setCursor(_display.width() / 2 - textWidth / 2,
_display.height() / 2 - textHeight / 2);
  _display.print("Temp:");
  _display.print(T);

  // Render the graphics buffer to screen
  _display.display();

// Code for PID
  double now = millis();
  dt = (now - last_time)/1000.00;
  last_time = now;

  //Mapping is done to convert all inputs to standard refercence of
0-100.
  double actual=mapT(T);
  double set=map(setpoint,10,25,0,100);
  double error = set - actual;
  output = pid(error);
  out=map(output,0,100,0,255)
  analogWrite(PELTIER_PIN, out);

  // Setpoint VS Actual
  Serial.print(set);
  Serial.print(",");
  Serial.println(actual);
  // Error
```

```
  Serial.println(error);
  delay(1000);
  if(err0r!=0){
     digitalWrite(PUMP_PIN, HIGH);
    delay(5000);  // Pump stays on for 5 seconds
  }
  else if{
     digitalWrite(PUMP_PIN, LOW);
    delay(5000);  // Pump stays on for 5 seconds
  }
}

//FUNCTIONS
double pid(double error)
{
  double proportional = error;
  integral += error * dt;
  double derivative = (error - previous) / dt;
  previous = error;
  double output = (kp * proportional) + (ki * integral) + (kd *
derivative);
  return output;
}
double mapT(double Temperature) {
  // Map -55 to 125 Celsius to 1 to 100
  return (Temperature - (-55)) * (100.0 - 1) / (125 - (-55)) + 1;
}
```