

# DEVELOPMENT AND SIMULATION OF TEMPERATURE CONTROL SYSTEM FOR LASER SYSTEM ON STM32 MCU

by:

SAYANDEEP PRAMANIK

Jadavpur University, Kolkata

Department of Electronics and Telecommunications  
Engineering

Roll: 002210701116

Year: 2025

## ABSTRACT

This report presents the design and simulation of a temperature control system on STM32 MCU, intending to refine the laser system of an atomic clock to ensure precise temperature and energy levels for the accurate functioning of the clock.

The report begins with an overview of the theoretical background related to the laser system, including the factors contributing to instability, such as temperature fluctuations, mechanical vibrations, and ageing of components. A comprehensive explanation of the PDH technique is provided, detailing its operational principles and advantages over the stabilization methods.

The design process focuses on the study and aims at controlling the temperature fluctuations of the laser and the vapor cell oven temperature. These fluctuations play a very significant role in the system's efficiency, as minor drifts and errors cause the system to be unstable and give undesirable results.

The control system is designed on an STM MCU board, precisely STM32U575I-EV. The IDE and the simulator used to program and simulate the system is STM32CUBEIDE, STM32CUBEMX, and STM32CUBEPROGRAMMER.

Results from the simulation demonstrate the effectiveness of the program and the pre-defined parameters along with it. The control system focuses on the error signal between the required temperature and the current temperature of the system and aims at controlling with a PID/PWM control system.

In conclusion, the project successfully illustrates the feasibility of the MCU-based control system at controlling the laser system of an atomic clock. Future work may explore advanced control algorithms and real-world experimental validation to further improve the system robustness.

## INDEX

S. No	Title	Page No.
1.	Introduction	
2.	Theory- 2.1 Laser system temperature 2.2 Factors affecting the system temperature 2.3 Need for regulation	
3.	Design of Temperature Control System- 3.1 Ideology of the control system 3.2 Boundaries and Specifications 3.3 Scope of improvement	
4.	Simulation Setup- 4.1 Simulation Environment 4.2 Components Defined 4.3 Simulation parameters 4.4 Simulation Procedure	
5.	Simulation Result and Analysis- 5.1 Simulation Results and Analysis 5.2 Discussion	
6.	Discussion- 6.1 Limitations of Findings 6.2 Design Limitations 6.3 Future Work	
7.	Conclusion	
8.	References	

## 1. INTRODUCTION

Lasers have revolutionized numerous fields, including telecommunications, medicines, and scientific research, due to their ability to produce coherent and monochromatic light. However, the performance of the laser systems can be significantly affected by fluctuations in temperature which can lead to inaccuracies in the applications that require high precision. Frequency drift refers to the gradual change in the output generated which includes wavelengths, frequency, and intensities.

In high-precision applications, such as optical frequency standards, gravitational wave detection, and quantum computing, maintaining a stable laser temperature is crucial. Even minute variations in temperatures can lead to significant and major errors in the measurement of parameters. Therefore, effective laser temperature regulation is essential for ensuring reliability and accuracy of the laser-based systems.

Implementation of a regulation system revolves around the development of an efficient control system. A stable control system refers to a system able to regulate the errors in the system to the desired value for the system, for this purpose we aim at using a closed loop control system to eliminate the variation in temperature values of the laser system, furthermore the benefits of a closed loop control system is that it prevents significant overshoot and also keeps the temperature values regulated at an acceptable range via controlling the system parameters.

The temperature control system used in this project is a closed PID/PWM control system.

Compared to normal control systems, this kind of system has a significant upper hand in controlling the system parameters, as it is not only a closed-loop system but also it has the ability to integrate, differentiate the errors, along with proportionalize to further smooth the response. The smoothing of the temperature variation is crucial to the stability of the laser system.

The project focuses on designing a closed-loop PID/PWM-based control system to eliminate fluctuations in the laser systems. The control system is designed on an STM32 evaluation board where the system is programmed in the board, the board reads the temperature of the laser system via analog sensors and uses ADC to read it digitally and send it to the control system programmed in it, the regulation signal is generated in the board and send to an output pin which regulates the heater control its heating efficiency thus, maintaining temperature. There are several inspirations for future development some of which include even more refined and smoother outputs, ultra-small-scale integration of the control system to be internally associated with the laser-based systems.

## 2. THEORY

This section provides the theoretical foundation necessary to understand the principles behind laser temperature fluctuations/drifts, the factors affecting it, and how they affect it.

### **2.1 Temperature fluctuations/drifts**

**Temperature fluctuations** refer to the variations or changes in the thermal state of the laser system components over time. These fluctuations represent deviations from the desired or stable operating temperature and can have significant effects on the laser's performance, stability, and output quality. Many components—such as laser diodes, pump sources, and optical elements—are sensitive to temperature changes in a laser system. Even small variations can cause shifts in emission wavelength, changes in beam quality, or fluctuations in output power. For example, the refractive index of laser media and optical coatings can vary with temperature, leading to thermal lensing or misalignment within the cavity. This ultimately affects the precision and reliability of the laser output.

Maintaining **precise temperature regulation** is therefore essential to ensure the laser operates within its optimal thermal range. Proper thermal control enhances the longevity of the laser components by reducing thermal stress and preventing overheating. It also improves the consistency and reproducibility of laser output, which is critical for applications requiring high precision such as spectroscopy, telecommunications, or medical procedures.

Understanding the **primary causes** of temperature fluctuations is crucial for designing effective thermal management strategies. By identifying and addressing the factors that lead to thermal instability—such as ambient environmental changes, heat generation during operation, power supply variations, cooling inefficiencies, material thermal properties, and mechanical influences—engineers can implement appropriate solutions. These may include active cooling systems, thermal insulation, stabilized power supplies, or vibration damping.

In summary, controlling temperature fluctuations not only preserves the quality and performance of laser systems but also enhances their reliability and operational lifespan, making it a foundational aspect of laser system design and maintenance.

## 2.2 Factors Affecting the System Temperature:

Primary Causes of Temperature Fluctuations in Laser-Based Systems are listed below.

- **Ambient Environment Changes:**

The surrounding environment plays a significant role in the thermal stability of a laser system. Variations in room temperature, humidity, and airflow can introduce external thermal changes. For example, an increase in room temperature or sudden drafts from air conditioning can cause the laser components to heat up or cool down unevenly, leading to fluctuations in the system's overall temperature.

- **Laser Operation Heat Generation:**

During operation, laser diodes and pumping mechanisms inherently produce heat as a byproduct of converting electrical energy into light. This heat accumulates in the laser cavity and adjacent components, causing localized temperature rises. If not managed properly, this can lead to thermal lensing, wavelength shifts, and reduced output power stability.

- **Electrical Power Supply Variations:**

Fluctuations or noise in the electrical power supplied to the laser can cause changes in the laser's internal heat generation. Sudden increases or drops in power can affect the laser's thermal load and consequently alter its temperature. Consistent power delivery is crucial to maintain steady thermal conditions.

- **Cooling System Inefficiencies:**

Cooling mechanisms such as fans, thermoelectric coolers (TECs), or liquid cooling systems are designed to dissipate heat and stabilize temperature. If these systems are inadequate in capacity, poorly maintained, or malfunctioning, they fail to remove excess heat effectively. This results in temperature buildup and fluctuations that degrade laser performance.

- **Thermal Conductivity of Materials:**

The laser system is composed of various materials with differing thermal conductivities. Components with low thermal conductivity can trap heat, while others with high conductivity dissipate heat rapidly. This mismatch can cause uneven heat distribution, creating localized hot spots or cold spots, which negatively impact temperature uniformity.

- **Mechanical Stress and Vibration:**

Mechanical forces, including vibrations or physical stress on the laser system, can affect the thermal contacts between components. Such disturbances may cause micro-movements or misalignments, disrupting heat flow pathways. This leads to irregular temperature changes and challenges in maintaining thermal equilibrium.

## 2.3 Need for Regulation

Temperature regulation is essential in laser systems to maintain stable and efficient operation. The key reasons for regulating temperature include:

- **Ensuring Performance Stability:**  
Maintains consistent output power and beam quality by preventing thermal-induced changes.
- **Preserving Output Quality:**  
Prevents wavelength shifts, mode hopping, and beam distortion caused by temperature fluctuations.
- **Protecting Components from Thermal Damage:**  
Avoid overheating that can degrade or permanently damage sensitive laser parts.
- **Extending System Longevity:**  
Minimises thermal stress and mechanical fatigue from repeated heating and cooling cycles.
- **Improving Energy Efficiency:**  
Maintains optimal operating conditions, reducing unnecessary power consumption.
- **Supporting Precision Applications:**  
Ensures reproducibility and accuracy needed in medical, telecommunications, and scientific uses.
- **Meeting Safety Standards:**  
Prevents hazardous overheating and aligns with safety regulations.



### 3. DESIGN OF TEMPERATURE CONTROL SYSTEM

#### **3.1 Ideology of the Control System**

##### **1. Closed-Loop Control System:**

The system is designed as a closed-loop control system, specifically employing a PID (Proportional–Integral–Derivative) control mechanism with Pulse Width Modulation (PWM) output. The principle of operation is based on feedback control: the system continuously monitors the output temperature, compares it with a predefined reference (setpoint), and adjusts the control signal to minimise the error.

Closed-loop control is essential in precision systems because it allows dynamic correction of deviations due to external disturbances, internal noise, or thermal drifts.

##### **2. Error-Based Regulation:**

The STM32 microcontroller reads the current temperature from an analog temperature sensor (such as the LM35), converts it into a digital value using the ADC (Analog-to-Digital Converter), and calculates the error using the formula:

$$\text{Error} = \text{Reference Temperature} - \text{Measured Temperature}$$

##### **3. PID Control Logic:**

The PID controller refines the control by considering:

- Proportional (P): Direct response to the current error. Larger errors produce a stronger control signal.
- Integral (I): Considers accumulated past errors to eliminate long-term deviations from the setpoint.
- Derivative (D): Responds to the rate of change of the error to dampen rapid fluctuations and overshoots.

The overall control signal is computed as:

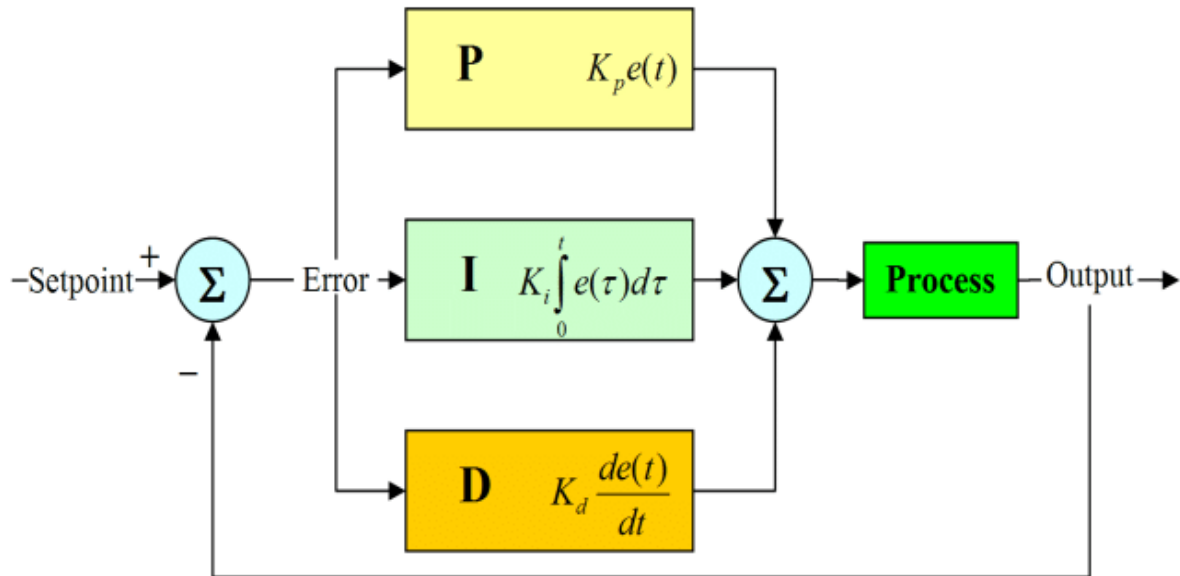
$$\text{Control Output} = K_p \cdot \text{error} + K_i \cdot \int \text{error} + K_d \cdot \frac{d(\text{error})}{dt}$$

This control signal is used to generate a PWM output which adjusts the power supplied to the heater, thereby regulating the system temperature.

##### **4. Hardware Integration:**

The control system is implemented on an STM32U575I-EV microcontroller unit. It handles:

- Reading temperature through ADC.
- Performing PID calculations in firmware.
- Generating PWM signals to control the heater or heating element.
- Optional UART communication to monitor system performance or change the reference temperature in real time.



### 3.2 Boundaries and Specifications

This section outlines the physical and functional limitations, operating conditions, and precision requirements that define the design and implementation of the temperature control system. Understanding these boundaries is critical to ensuring the system operates effectively within the desired performance envelope and avoids instability, thermal overshoot, or hardware limitations.

#### 1. Hardware Integration:

The temperature control system is implemented on the STM32U575I-EV microcontroller unit, chosen for its high-performance processing capabilities, integrated ADCs, and peripheral support suited for real-time control applications. The system integrates the following hardware functionalities:

- **Temperature Acquisition via ADC:**  
Analog temperature sensors (such as the LM35) sense the laser system's thermal state. The STM32's ADC channels sample these analogue signals to convert them into digital values for processing.
- **PID Control Execution:**  
The digital temperature readings are fed into a PID control algorithm implemented in firmware. The algorithm calculates the required control

effort based on the error between the current and target temperature values.

- **PWM Signal Generation for Heater Control:**  
The output of the PID controller is mapped to a PWM signal, which modulates the power delivered to the heating element. This allows for fine-grained control over the heating rate and thermal stability of the system.
- **UART Communication (Optional):**  
A UART interface is incorporated to provide serial communication with a host terminal (such as PuTTY). Through this interface, users can:
  - Monitor real-time temperature values and control signals,
  - Dynamically set or change the reference temperature,
  - Observe system behavior for tuning and debugging.

## 2. Simulation-Based Development:

Before deploying the system in a physical setup, simulation-based verification is employed. This stage ensures that both the control logic and the hardware-software interface operate as expected. The simulation serves the following purposes:

- **Gain Tuning:**  
PID gain constants ( $K_p$ ,  $K_i$ , and  $K_d$ ) are tested and adjusted for optimal response. Poorly tuned gains can result in excessive overshoot, slow settling times, or oscillations.
- **Thermal Response Testing:**  
The simulation models the time constants associated with the heating system. This includes understanding how quickly the heater responds to control signals and how the system retains or loses heat over time.
- **Boundary Condition Testing:**  
The system is tested under simulated external disturbances and initial conditions to ensure that it behaves reliably in a variety of operating scenarios.

Simulation provides a safe, cost-effective, and efficient platform to iterate on design before implementation on physical hardware.

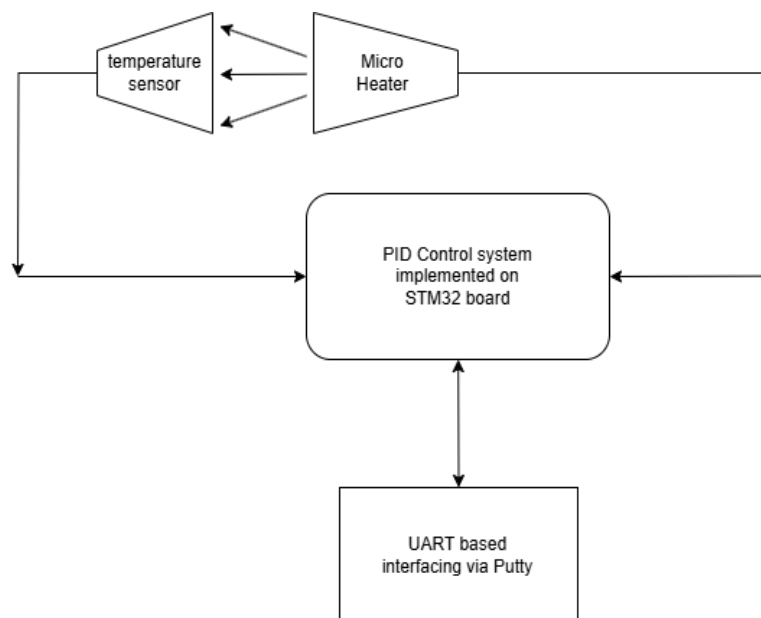
## 3. Design Constraints:

To ensure the control system performs effectively and safely in the context of laser temperature regulation, several design constraints and operational boundaries must be respected:

- **Temperature Operating Range:**  
The target application involves controlling the temperature of laser components, which typically require a tightly regulated thermal range of 30°C to 50°C. The system must prevent temperatures from exceeding

these limits to avoid damaging sensitive optical and electronic components.

- **Thermal Inertia Considerations:**  
Heating elements and the surrounding environment possess thermal inertia, meaning changes in temperature occur gradually. The control algorithm must account for this lag to avoid overcompensating or undershooting the target.
- **Sensor Resolution and Response Time:**  
The analog temperature sensor (e.g., LM35) and the microcontroller's ADC introduce limitations on resolution and response time. The system must operate within the limits of ADC precision (typically 12–16 bits) and sampling frequency to ensure accurate and timely feedback.
- **Power Regulation and Efficiency:**  
The control system must efficiently modulate power to the heater to avoid unnecessary energy consumption, overheating, or thermal runaway. PWM duty cycle limits are enforced to maintain a balance between responsiveness and safety.
- **Environmental and Mechanical Factors:**  
External factors such as ambient room temperature, airflow, and mechanical vibrations may affect the thermal behavior of the laser system. The control algorithm must be robust enough to tolerate such disturbances and maintain stability.



### 3.3 Scope of Improvement

While the current temperature control system demonstrates effective performance in simulation and is tailored for precision thermal regulation in laser-based applications, particularly atomic clocks, there remains considerable scope for improvement to enhance robustness, adaptability, and integration for real-world deployment.

#### 1. Real-World Experimental Validation

The current system has been primarily validated through simulations.

Transitioning to physical hardware experimentation will provide critical insights into:

- The actual thermal response of materials and heaters,
- Environmental interference such as vibration, airflow, or electromagnetic noise,
- Sensor drift and real-time control stability.

Implementing and testing the control system on a fully functional hardware prototype will allow fine-tuning of PID gains under real conditions and help assess long-term reliability.

#### 2. Adaptive and Intelligent Control Algorithms

The system presently uses a classic PID control approach with fixed gains. In dynamic or unpredictable thermal environments, this may lead to sub-optimal control. Potential enhancements include:

- Auto-tuning PID algorithms that adjust gain parameters in real time based on system behavior,
- Model Predictive Control (MPC) or fuzzy logic control for better handling of non-linearities and varying thermal loads,
- Machine learning-based control to predict temperature drifts and proactively adjust outputs.

These intelligent methods can improve control accuracy, adaptability, and fault tolerance.

#### 3. Enhanced Sensor and Data Acquisition Capabilities

The current design uses a basic analog temperature sensor (e.g., LM35).

Enhancements in this area could involve:

- High-resolution digital temperature sensors with faster response and noise immunity,
- Use of multiple sensors to capture spatial temperature variations,
- Implementing digital filtering techniques to smooth noisy data before PID processing.

This would ensure more accurate temperature feedback, essential for high-precision applications like laser frequency stabilization.

#### 4. Power Management and Efficiency Optimization

PWM-based control, while effective, can be further optimized:

- Implementing variable-frequency PWM to match heater characteristics,
- Incorporating low-power modes for idle periods to improve energy efficiency,
- Feedback-based control on power draw to prevent overshoot and improve thermal ramp-up behavior.

Efficient power modulation is especially crucial in embedded or portable systems.

#### 5. System Miniaturization and Integration

Currently, the system exists as a separate control unit interfaced with the laser setup. Future development can focus on:

- Miniaturizing the control system using compact microcontrollers or ASICs,
- Direct integration with laser modules in an embedded form factor,
- Thermal shielding and mechanical packaging to isolate and protect the control electronics from environmental factors.

Such integration is vital for scalable deployment in commercial or defense-grade atomic clocks.

#### 6. Remote Monitoring and Control Features

Introducing network interfaces such as Bluetooth, Wi-Fi, or RS-485 can enable:

- Remote setting of reference temperature,
- Real-time monitoring of thermal performance,
- Logging and diagnostic data acquisition for long-term analysis.

This would enhance usability in distributed or inaccessible systems.

## 4. SIMULATION SETUP

### **4.1 Simulation Environment**

The development and simulation of the temperature control system were conducted using an integrated toolchain tailored for STM32 microcontrollers, specifically the STM32U575I-EV evaluation board. This environment provided the necessary framework to design, implement, simulate, and debug the control algorithm in a systematic and structured manner.

#### **1. Development Board: STM32U575I-EV**

The target hardware platform used for implementation was the STM32U575I-EV, an evaluation board based on the STM32U575 microcontroller. This board features advanced peripherals suitable for high-precision control applications, including ADC modules, PWM-capable timers, and UART communication interfaces. The onboard features facilitated sensor interfacing, real-time signal processing, and heater control, forming the core of the temperature regulation system.





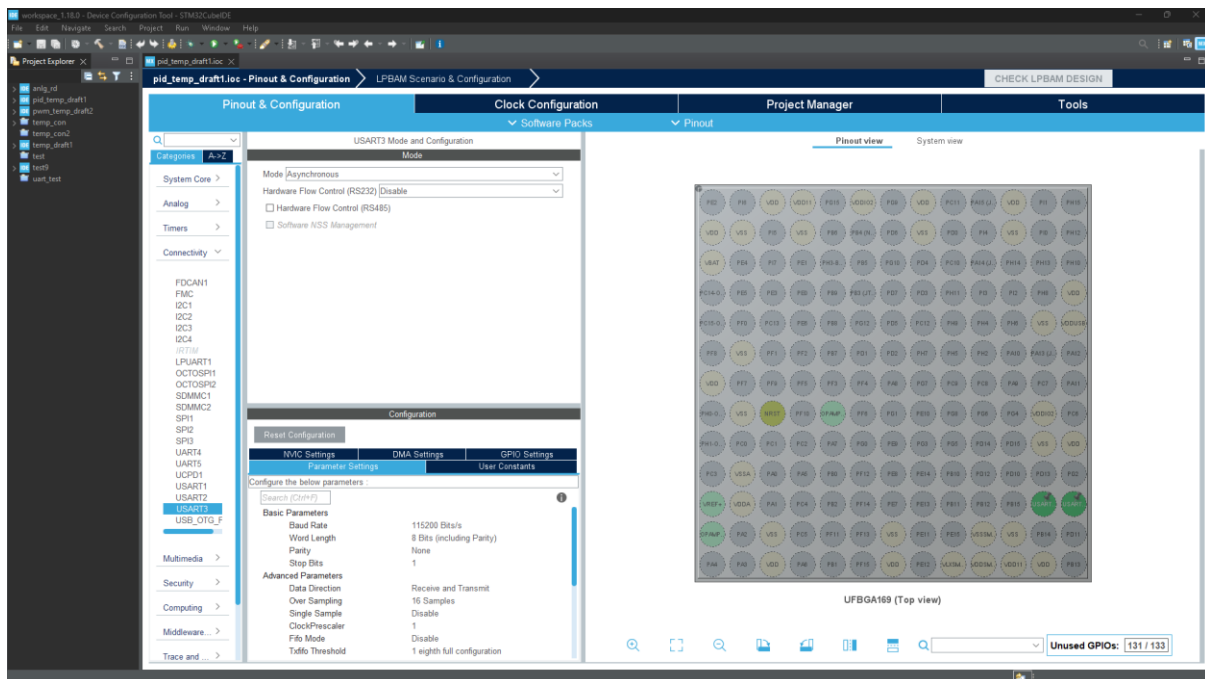
STM32U575I-EV Board



## 2. Code Development and Compilation: STM32CubeIDE

The primary software environment used for writing, compiling, and flashing code was STM32CubeIDE. It served as the central development platform by integrating the following capabilities:

- **Code Editor and Project Management**  
C-based firmware for the STM32 was written and managed within the IDE using CMSIS and HAL libraries for hardware abstraction.
- **Compilation and Build System**  
The STM32CubeIDE incorporates a built-in ARM GCC toolchain (GNU Arm Embedded Toolchain) to compile the source code into a binary image (.hex/.elf) ready for flashing onto the microcontroller.
- **Flashing and Debugging**  
The IDE allows flashing the compiled firmware directly onto the STM32U575I-EV via the onboard ST-Link debugger interface, as well as providing live debugging features.



## 3. Pin Configuration: STM32CubeMX (Integrated)

STM32CubeMX, which is integrated into STM32CubeIDE, was used to:

- Configure GPIO ports for analog sensor input, PWM output, and UART communication,
- Set up internal peripherals such as ADC channels, timers, and USART modules,
- Generate initialization code automatically, reducing development time and human error.

This graphical tool simplified the process of mapping the hardware resources to their respective firmware components.

#### 4. Device Connection and Flashing: STM32CubeProgrammer

To verify hardware connectivity and perform low-level programming tasks, the STM32CubeProgrammer tool was used. It served the following purposes:

- **Board Detection and Connection Check**  
Ensured that the STM32U575I-EV board was properly connected to the host system via USB.
- **Memory Erase and Flashing (Alternative to IDE)**  
Allowed manual erasing, flashing, and verification of the firmware image, especially useful when STM32CubeIDE flashing was unavailable or needed redundancy.

#### 5. Serial Communication Interface: PuTTY

To interact with the control system and monitor its real-time performance, the PuTTY application was used as the UART serial terminal. Its roles included:

- Displaying real-time temperature readings and control signals sent from the STM32,
- Allowing manual input of reference temperature values via serial commands,
- Observing the effect of control actions on temperature regulation in real time.

PuTTY was configured to communicate over the COM port assigned to the STM32U575I-EV's virtual serial interface, with standard baud rate and data format settings.

## 4.2 Components Defined

This section outlines the key hardware and software components configured within the STM32-based simulation environment to implement the temperature control system. Each component serves a distinct purpose in ensuring accurate sensing, control computation, actuation, and communication.

Depending on the model used and the level of complexity of the system, several simulation models are created using several components. A brief description of the models developed throughout the internship is stated below.

### 1. System Clock:

We are using the STM32 boards Internal 16MHz clock for our project. We can configure clock for better efficiency but we need to configure it externally and will add on to the complexity.

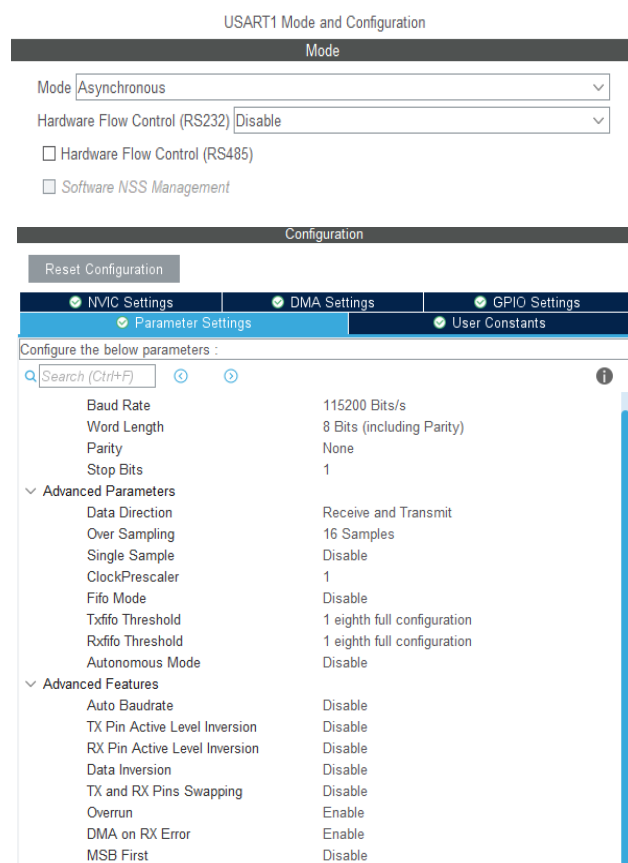
### 2. BANG-BANG Control system (Virtual system):

This type of control system has only two control state which in this case is heater on/off, this adds to simplicity of the system and the components. But, the controlling efficiency is very less as there is no regulation for any overshoot thus it's almost impossible to reach a significantly ignorable error value.

Components:

- UART: Only UART is used for the virtual system to interface the heater outcomes.

The parameter settings for the displayed in the image attached.



### UART configuration

#### 3. Simple PWM Control system (Virtual system):

With a bit more complex architecture of the system compared to BANG-BANG this implements PWM to regulate the heater efficiency to control the overshoot and increase regulation to some extent.

Components:

- UART: Only UART is used for the virtual system to interface the heater outcomes.
- PWM Timer: with the built in feature of stm32 board we can implement PWM using internal pins of the board to control the heater efficiency.

The parameter settings for the displayed in the image attached.

TIM3 Mode and Configuration

Mode

Slave Mode Disable ▼  
 Trigger Source Disable ▼  
 Clock Source Internal Clock ▼  
 Channel1 Disable ▼  
 Channel2 Disable ▼  
 Channel3 PWM Generation CH3 ▼  
 Channel4 Disable ▼  
 Combined Channels Disable ▼  
 Use ETR as Clearing Source Disable ▼  
☐ XOR activation  
☐ One Pulse Mode

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :
 

i

▼ Counter Settings

Prescaler (PSC - 16 bits value) 1599  
 Counter Mode Up  
 Dithering Disable  
 Counter Period (AutoReload Register) 99  
 Internal Clock Division (CKD) No Division  
 auto-reload preload Enable

▼ Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit) Disable (Trigger input effect not delayed)  
 Trigger Event Selection TRGO Reset (UG bit from TIMx\_EGR)

▼ Clear Input

Clear Input Source Disable

▼ PWM Generation Channel 3

Mode PWM mode 1

### PWM Configuration

#### 4. PID-PWM Control system (Virtual system):

Stepping up the complexity to increase the control efficiency this system used PID-PWM system to smoothen the regulation and establish an accurate control system.

Components:

- UART: Only UART is used for the virtual system to interface the heater outcomes.
- PWM Timer: with the built-in feature of stm32 board we can implement PWM using internal pins of the board to control the heater efficiency.

The parameter settings for the displayed in the image attached.

#### 5. PID-PWM Control system (Sensors Configured):

This uses the same ideology and coding as the previous but it's the real-life implementable system with sensors configured to take actual heat/temperature reading

Components:

- UART: Only UART is used for the virtual system to interface the heater outcomes.
- PWM Timer: with the built-in feature of stm32 board we can implement PWM using internal pins of the board to control the heater efficiency.
- ADC: Using the internal ADC module of the board we can configure analog sensors (LM35 in this case) and take analog temperature readings.

The parameter settings for the displayed in the image attached.

ADC1 Mode and Configuration

Mode

IN1

Disable

IN2

Disable

IN3

Disable

IN4

Disable

IN5

IN5 Single-ended

IN6

Disable

IN7

Disable

IN8

Disable

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :

Search (Ctrl+F)

ADCs\_Common\_Settings

Mode

Independent mode

ADC\_Settings

Clock Prescaler

Asynchronous clock mode divided by 1

Resolution

ADC 14-bit resolution

Gain Compensation

0

Scan Conversion Mode

Disabled

Continuous Conversion Mode

Disabled

Discontinuous Conversion Mode

Disabled

Trigger Frequency

High frequency

End Of Conversion Selection

End of single conversion

Overrun behaviour

Overrun data preserved

Left Bit Shift

No bit shift

Conversion Data Management Mode

Regular Conversion data stored in DR register ...

Low Power Auto Wait

Disabled

ADC\_Regular\_ConversionMode

Enable Regular Conversions

Disable

Oversampling Ratio

1

## ADC Configuration

23

### 4.3 Simulation parameters

The simulation of the temperature control system was configured using a set of defined parameters that represent the behavior of both the physical components and the control logic. These parameters are crucial to mimic real-world operation and ensure that the control algorithm responds accurately to changes in system conditions. The simulation was executed within STM32CubeIDE using code-level simulations and real-time debugging tools, complemented by external monitoring through UART.

1. Temperature Setpoint:
  - Reference: set at 30°C (test value, can be set as per system requirements)
  - For PID-PWM lower threshold and upper threshold is set to control overshoot and oscillations at 29.5°C and 30.5°C respectively (also flexible to system requirements)
2. Temperature initialization:
  - For virtual systems UART is configured to take manual entry from user
  - For sensor configured system the analog sensors take the actual temperature of the target system and uses internal ADC module for configuring and displays via UART.
3. Control Parameters:
  - The Bang-bang system is set to simulate temperature rise/fall at a steady rate of 0.01°C.
  - For PWM control parameter is set using the below stated equations
    - **Error Calculation:**  
$$\text{Error} = \text{Reference Temperature} - \text{Current Temperature}$$
    - **Heater Control Signal:**  
$$\text{heater\_percent} = 10 \times \text{error}$$

This is a Proportional-only controller. The proportional gain = 10. It means for every 1°C error, the heater gets 10% more power.
  - For PID-PWM system the compiled simulation parameters are stated below in details
    - **Proportional Gain (Kp):**  
Set to **20.0**. This term increases the heater power proportionally to the instantaneous error between the reference temperature and the measured temperature.



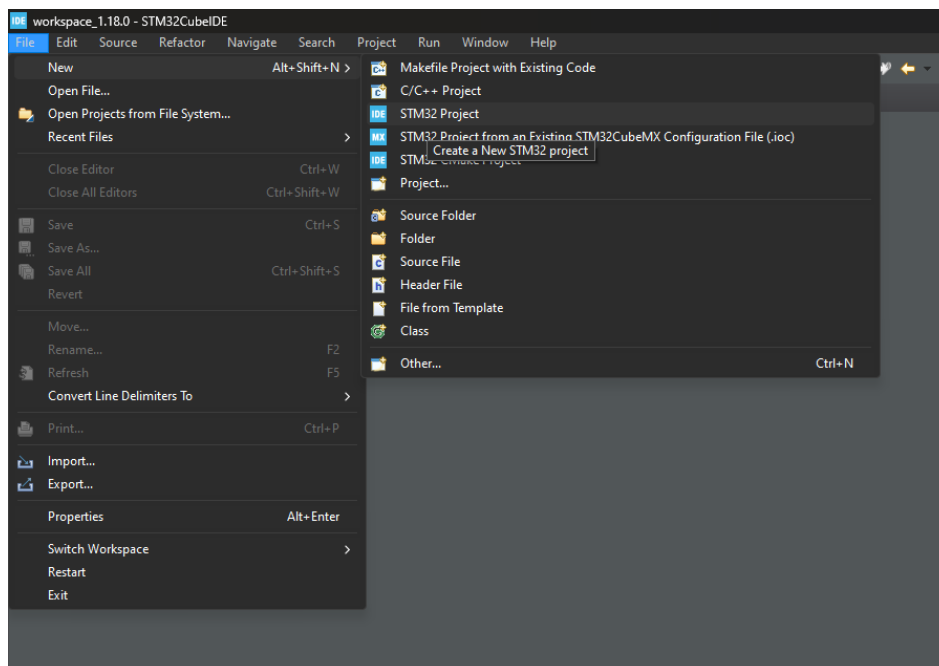
- **Integral Gain (Ki):**  
Set to **2.0**. The integral term accumulates the error over time and helps eliminate steady-state offset. It is updated every 0.5 seconds.
- **Derivative Gain (Kd):**  
Set to **10.0**. The derivative term responds to the rate of change of error, helping to reduce overshoot and improve response smoothness.
- **Sampling Interval:**  
The PID control loop updates every **500 milliseconds** (`HAL_Delay(500)`), which sets the effective sample time for integral and derivative calculations.
- **Integral Windup Protection:**  
The integral term is limited to the range of **-50.0 to +50.0** to avoid excessive accumulation, which can destabilize the system.
- **Output Clamping:**  
The computed control output (heater duty cycle) is constrained between **0% and 100%** to prevent invalid or unsafe power levels.
- **Temperature Range Control:**  
When temperature  $\geq 30.0\text{ }^{\circ}\text{C}$  → Heater is turned **off** and PID terms are reset. When temperature  $\leq 29.8\text{ }^{\circ}\text{C}$  → PID controller is **activated** to increase temperature. Between  $29.8\text{ }^{\circ}\text{C}$  and  $30.0\text{ }^{\circ}\text{C}$  → Control is **inhibited** to avoid rapid toggling.
- **Control Output Application:**  
The result of the PID computation is used to set the PWM duty cycle (via TIM3 Channel 4) which controls the heating element.

## 4.4 Simulation Procedure

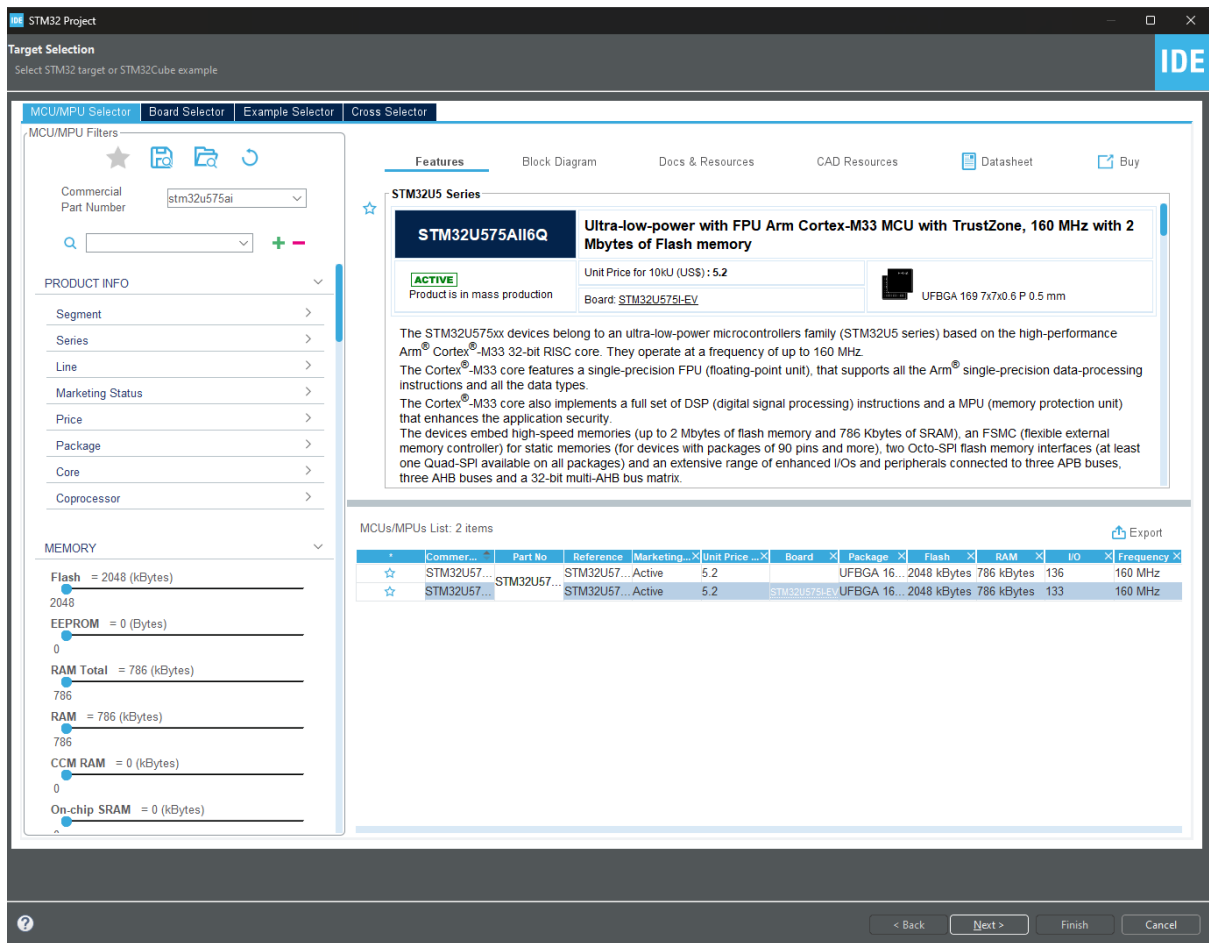
This section is dedicated to give a brief walkthrough on how to work on the STM32CUBEIDE environment. The step-by-step walkthrough along with interface reference is given in details below.

- **To create a New Project first:** To start creating a new STM32 project, first **open STM32CubeIDE** and wait for it to fully load the workspace. Once the IDE is ready, navigate to the top menu and click **File** → **New** → **STM32 Project**. This will open the **Target Selection** window, where you must choose the STM32 device or board you're working with. If you know the exact board you're using—such as STM32U575I-EV—click on the **Board Selector** tab → type the board name in the search bar → select it from the results → then click **Next**. Alternatively, if you want to choose a specific microcontroller, switch to the **MCU Selector** tab → type the part number (e.g., STM32U575AI) → select the matching device from the list → then click **Next**.

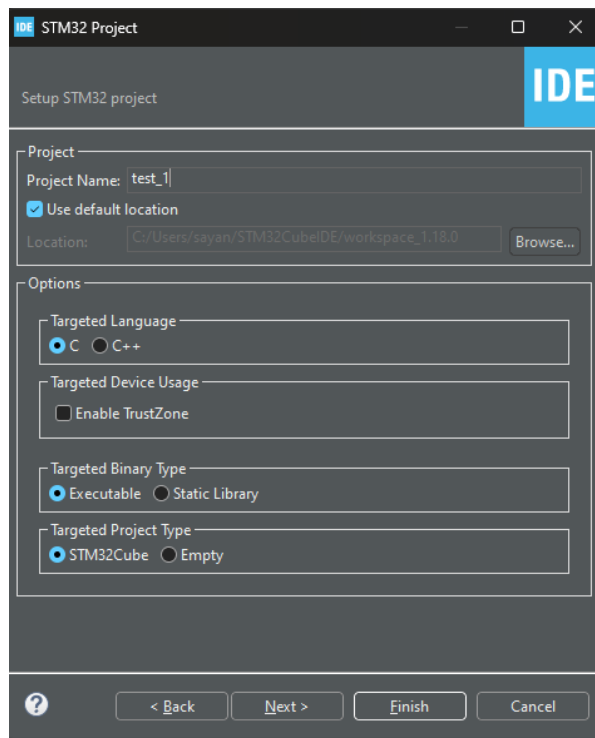
In the next window titled **Project Setup**, enter a **Project Name** (e.g., Test\_1), ensure the **Targeted Project Type** is set to “STM32Cube” and the **Toolchain/IDE** is set to “STM32CubeIDE”, then click **Finish**. The IDE will now create the project directory structure and automatically open the **.ioc** file, launching the **STM32CubeMX graphical configuration interface**. Here, you'll land on the **pinout and configuration view**, where you see a **schematic diagram of the MCU/board pins**. This is where you begin assigning peripheral functions to the pins based on your application requirements.



Project Creation

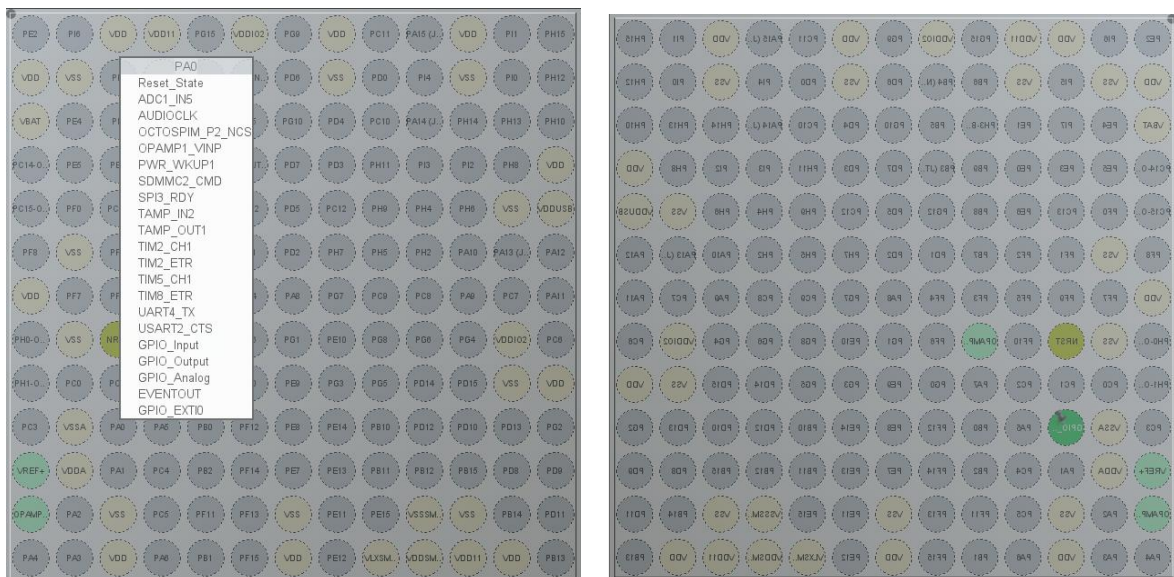


## Target Selector



## Project Naming

- Configuring Pins in CUBEMX Interface:** After opening the .ioc file in STM32CubeIDE, you'll see the MCU pinout view. To configure a pin, click on the desired pin on the MCU diagram, then select the required function from the drop-down (e.g., GPIO, ADC, UART). The pin will be assigned and listed under the corresponding peripheral in the left panel. Click the peripheral name in the left panel to adjust settings like mode, pull-up/pull-down, speed, and optional labels. Repeat this for all required pins. Depending on the Pins and the required function define the pin parameters take reference from the parameter setting mentioned in earlier sections. Once done, click Save, then Generate Code to auto-generate initialization code for all pin and peripheral configurations.



Configuring Pin



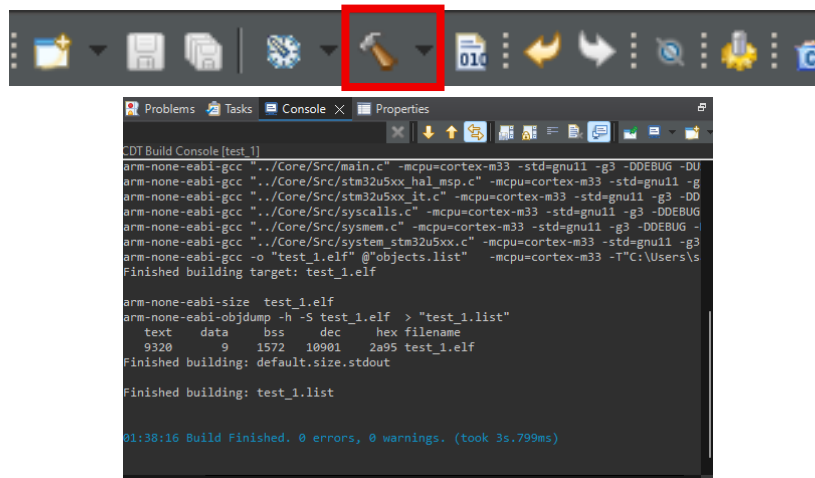
Generating Code

- **Writing Code:** After Generating the code it lands you on a coding window with basic main.h library initiated along with pre coded Pin configuration which came from defining pins in the CubeMX interface.

Initialize the required libraries as per code (Check if the required library have their required Driver files present in the project directory, if not add to avoid errors, simple codes usually don't require external/3<sup>rd</sup> party driver files). After that write the system code in usual stm syntax get a grip on it by practicing some example projects.

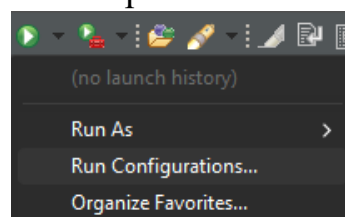
- **Building Project:** Save the code once written, after that we need to build it in order to make it uploadable to the board. For this check the syntax and then click on the build button in debug mode. The build window pops and displays the building process wait for it to finish.

Check if errors are shown in the window, fix them if present and build again. The code is ready to be flashed in the board.

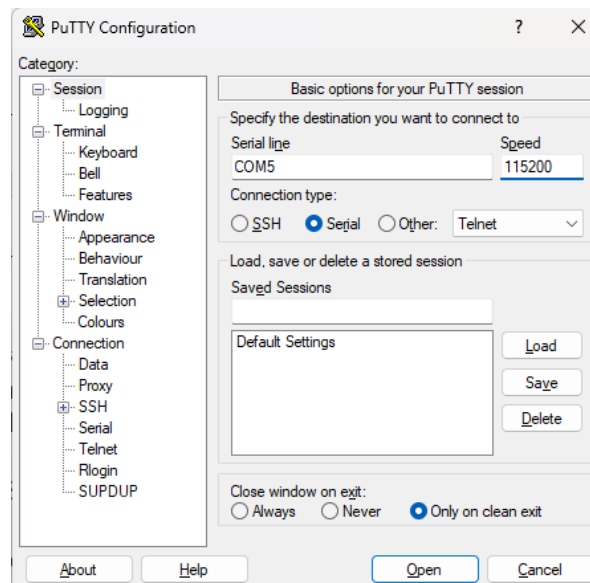


### Building Code

- **Flashing Code:** To flash the built code firstly connect the board to the PC, check if STLINK is established, once STLINK is setup then select the **Run Configuration** option under the run button, the configuration window is displayed select required setting then click on the **debug** button to upload the code on the board.



- **Viewing On Putty:** Once the code is uploaded successfully it starts to run, to view the output via UART on Putty we need to open the application select the **Serial** button define the COM port (check active STM port in device manager), define the Baud Rate (depends on what is set in UART while configuring in CUBEMX), then click on **Open** button, this opens the display interface, if nothing is displayed run the code again with the Putty session running. Thus, you ran your code on the board and viewed data via UART on Putty.



Putty Interface

## 5. SIMULATION RESULTS AND ANALYSIS

### 5.1 Simulation Results and Analysis

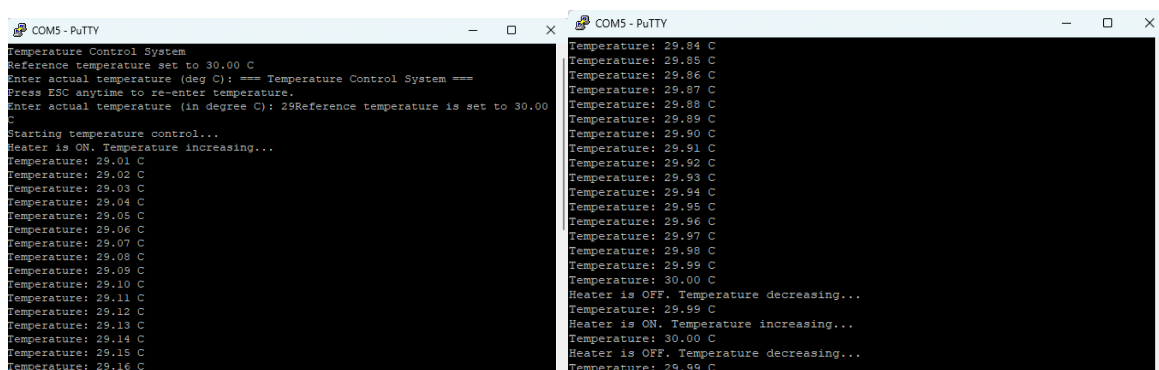
This section focuses on the simulation results of the previously discussed system components and control strategies. It presents the behavior of the temperature control system under defined parameters, including sensor response, PWM modulation, and PID control performance. The aim is to evaluate the effectiveness of the system in maintaining thermal stability around the reference temperature. Observations are made based on UART output data and system response over time, providing insight into both the strengths and potential limitations of the current implementation.

#### 1. Bang-Bang Type Control system:

The simulation output demonstrates a basic **bang-bang control behavior with fixed-rate adjustment**. The system begins with an initial user-defined temperature input (e.g., **29.00 °C**) and compares it to a reference temperature of **30.00 °C**. Upon detecting that the actual temperature is below the reference, the system activates the **heater** and enters a loop where it **increases the temperature incrementally by 0.01 °C per step**, as defined by TEMP\_STEP.

As observed in the terminal output, the temperature rises steadily, confirming that the system responds correctly to the control logic. Each new temperature is printed in real-time, indicating that the control loop is functioning and that the UART interface is reliably transmitting data. This controlled and gradual heating eliminates abrupt temperature jumps, providing **safe and predictable behavior**—a key enhancement over traditional on-off bang-bang systems. The system also listens for user input during operation (via ESC or newline), allowing for live updates to the setpoint or reinitialization, which is useful for manual override or dynamic reference control.

While effective in its simplicity, this method lacks dynamic responsiveness to changing rates of error (no proportional or derivative scaling), which could lead to **slow settling** or **longer time to reach setpoint** in real-world scenarios with thermal lag.



```
COM5 - PuTTY
Temperature Control System
Reference temperature set to 30.00 C
Enter actual temperature (deg C): == Temperature Control System ==
Press ESC anytime to re-enter temperature.
Enter actual temperature (in degree C): 29Reference temperature is set to 30.00
C
Starting temperature control...
Heater is ON. Temperature increasing...
Temperature: 29.01 C
Temperature: 29.02 C
Temperature: 29.03 C
Temperature: 29.04 C
Temperature: 29.05 C
Temperature: 29.06 C
Temperature: 29.07 C
Temperature: 29.08 C
Temperature: 29.09 C
Temperature: 29.10 C
Temperature: 29.11 C
Temperature: 29.12 C
Temperature: 29.13 C
Temperature: 29.14 C
Temperature: 29.15 C
Temperature: 29.16 C

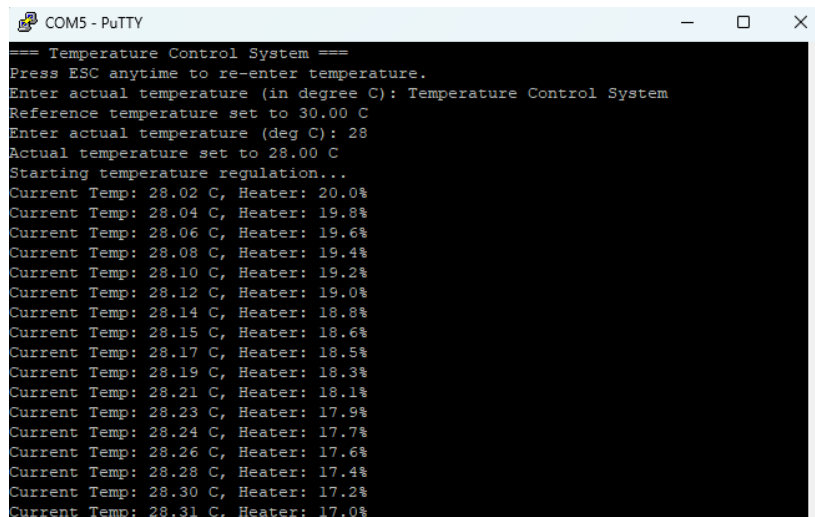
COM5 - PuTTY
Temperature: 29.84 C
Temperature: 29.85 C
Temperature: 29.86 C
Temperature: 29.87 C
Temperature: 29.88 C
Temperature: 29.89 C
Temperature: 29.90 C
Temperature: 29.91 C
Temperature: 29.92 C
Temperature: 29.93 C
Temperature: 29.94 C
Temperature: 29.95 C
Temperature: 29.96 C
Temperature: 29.97 C
Temperature: 29.98 C
Temperature: 29.99 C
Temperature: 30.00 C
Heater is OFF. Temperature decreasing...
Temperature: 29.99 C
Heater is ON. Temperature increasing...
Temperature: 30.00 C
Heater is OFF. Temperature decreasing...
Temperature: 29.99 C
```



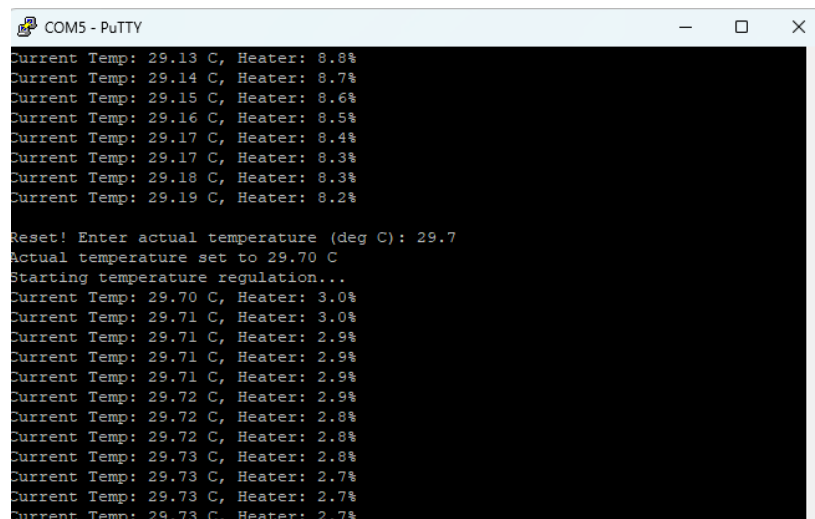
## 2. PWM Based Virtual system:

The simulation demonstrates a proportional control system that uses PWM (Pulse Width Modulation) to manage temperature with smooth, adaptive behavior. Starting from an initial input of 28.00 °C, the system calculates the error relative to the fixed reference temperature of 30.00 °C. This error is then scaled by a proportional gain to determine the heater output, expressed as a PWM duty cycle. In this case, the system begins with a 20% duty cycle corresponding to a 2 °C difference, showing immediate activation of the heating process.

As the temperature gradually rises, the error decreases, causing the system to reduce the heater power step by step—from 20.0% to 17.0%—as seen in the UART output. This real-time modulation ensures that the system slows down heating as it approaches the setpoint, preventing abrupt overshoot. Compared to basic bang-bang control, this method offers significantly smoother and more efficient regulation, making it ideal for applications requiring controlled thermal ramp-up and greater stability near the target temperature.



```
COM5 - PuTTY
=== Temperature Control System ===
Press ESC anytime to re-enter temperature.
Enter actual temperature (in degree C): Temperature Control System
Reference temperature set to 30.00 C
Enter actual temperature (deg C): 28
Actual temperature set to 28.00 C
Starting temperature regulation...
Current Temp: 28.02 C, Heater: 20.0%
Current Temp: 28.04 C, Heater: 19.8%
Current Temp: 28.06 C, Heater: 19.6%
Current Temp: 28.08 C, Heater: 19.4%
Current Temp: 28.10 C, Heater: 19.2%
Current Temp: 28.12 C, Heater: 19.0%
Current Temp: 28.14 C, Heater: 18.8%
Current Temp: 28.15 C, Heater: 18.6%
Current Temp: 28.17 C, Heater: 18.5%
Current Temp: 28.19 C, Heater: 18.3%
Current Temp: 28.21 C, Heater: 18.1%
Current Temp: 28.23 C, Heater: 17.9%
Current Temp: 28.24 C, Heater: 17.7%
Current Temp: 28.26 C, Heater: 17.6%
Current Temp: 28.28 C, Heater: 17.4%
Current Temp: 28.30 C, Heater: 17.2%
Current Temp: 28.31 C, Heater: 17.0%
```



```
COM5 - PuTTY
Current Temp: 29.13 C, Heater: 8.8%
Current Temp: 29.14 C, Heater: 8.7%
Current Temp: 29.15 C, Heater: 8.6%
Current Temp: 29.16 C, Heater: 8.5%
Current Temp: 29.17 C, Heater: 8.4%
Current Temp: 29.17 C, Heater: 8.3%
Current Temp: 29.18 C, Heater: 8.3%
Current Temp: 29.19 C, Heater: 8.2%

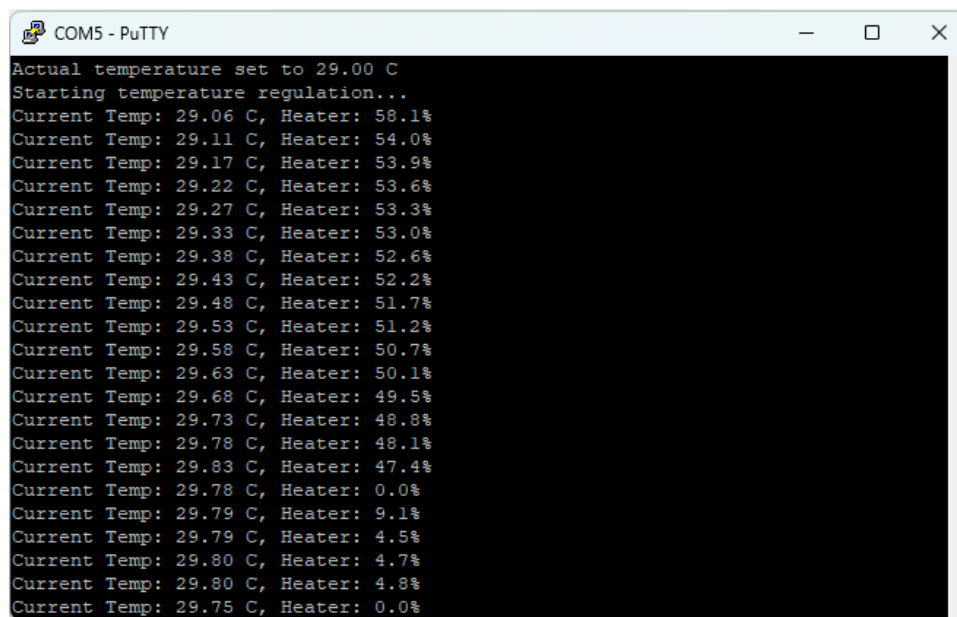
Reset! Enter actual temperature (deg C): 29.7
Actual temperature set to 29.70 C
Starting temperature regulation...
Current Temp: 29.70 C, Heater: 3.0%
Current Temp: 29.71 C, Heater: 3.0%
Current Temp: 29.71 C, Heater: 2.9%
Current Temp: 29.71 C, Heater: 2.9%
Current Temp: 29.71 C, Heater: 2.9%
Current Temp: 29.72 C, Heater: 2.9%
Current Temp: 29.72 C, Heater: 2.8%
Current Temp: 29.72 C, Heater: 2.8%
Current Temp: 29.73 C, Heater: 2.8%
Current Temp: 29.73 C, Heater: 2.7%
Current Temp: 29.73 C, Heater: 2.7%
Current Temp: 29.73 C, Heater: 2.7%
```



### 3. PID-PWM Based Virtual System:

The simulation showcases an improved temperature control system using PID logic with hysteresis to achieve fast, stable, and accurate regulation. Starting at 29.00 °C, the system calculates the error from the 30.00 °C setpoint and applies proportional, integral, and derivative corrections to control the heater's PWM output. Heater power starts at 58.1% and gradually decreases as the temperature rises. A hysteresis band between 29.8 °C and 30.0 °C prevents frequent toggling near the setpoint, allowing the system to settle smoothly.

In comparison to the earlier proportional-only PWM system—which adjusted heater power based solely on current error—this PID-based design provides better accuracy and stability. The integral term removes long-term error, while the derivative term limits overshoot. Combined with hysteresis, the system avoids rapid fluctuations and maintains tighter temperature control. This results in quicker settling, smoother operation, and greater reliability for precision thermal applications.



```
COM5 - PuTTY
Actual temperature set to 29.00 C
Starting temperature regulation...
Current Temp: 29.06 C, Heater: 58.1%
Current Temp: 29.11 C, Heater: 54.0%
Current Temp: 29.17 C, Heater: 53.9%
Current Temp: 29.22 C, Heater: 53.6%
Current Temp: 29.27 C, Heater: 53.3%
Current Temp: 29.33 C, Heater: 53.0%
Current Temp: 29.38 C, Heater: 52.6%
Current Temp: 29.43 C, Heater: 52.2%
Current Temp: 29.48 C, Heater: 51.7%
Current Temp: 29.53 C, Heater: 51.2%
Current Temp: 29.58 C, Heater: 50.7%
Current Temp: 29.63 C, Heater: 50.1%
Current Temp: 29.68 C, Heater: 49.5%
Current Temp: 29.73 C, Heater: 48.8%
Current Temp: 29.78 C, Heater: 48.1%
Current Temp: 29.83 C, Heater: 47.4%
Current Temp: 29.78 C, Heater: 0.0%
Current Temp: 29.79 C, Heater: 9.1%
Current Temp: 29.79 C, Heater: 4.5%
Current Temp: 29.80 C, Heater: 4.7%
Current Temp: 29.80 C, Heater: 4.8%
Current Temp: 29.75 C, Heater: 0.0%
```

#### 4. PID-PWM Based System with Sensors Configured:

The simulation demonstrates a fully autonomous, sensor-driven temperature control system integrating **PID control, real-time ADC sensing using an LM35 sensor**, and a manual **pause/resume function via UART**. The system begins by reading the temperature directly from ADC input and compares it to a fixed reference of 30.00 °C. PID logic is applied using tuned gains ( $K_p = 20$ ,  $K_i = 2$ ,  $K_d = 10$ ) to compute heater output as a PWM signal. When the ESC key is pressed, the system pauses operation—turning the heater off and freezing control until ESC is pressed again to resume. This allows safe, interactive control during live operation, as reflected in the simulation where the system pauses, then resumes with a heater output of 100% at low temperatures (21.76 °C to 21.84 °C).

Compared to previous implementations—like the manual-input-based PID system—this design significantly improves practicality and autonomy. By using the LM35 sensor and ADC for temperature measurement, it eliminates reliance on user-fed values and mimics real-world feedback. Additionally, the pause/resume mechanism enhances usability, safety, and testing convenience. The PID logic combined with hysteresis ensures stable and efficient heating, while real-time sensing ensures accurate, closed-loop regulation. Overall, this version is the most realistic and robust, providing a complete embedded temperature control system with responsive behavior, user interaction, and adaptive control logic.

```
System Paused... Press ESC to resume
System Paused... Press ESC to resume
System Paused... Press ESC to resume
System Paused... Press ESC to resume
System Paused... Press ESC to resume
System Resumed
Current Temp: 21.76 C
Heater Output: 100.0%
Current Temp: 21.76 C
Heater Output: 100.0%
Current Temp: 21.84 C
Heater Output: 100.0%
Current Temp: 21.84 C
Heater Output: 100.0%
Current Temp: 21.84 C
Heater Output: 100.0%
Current Temp: 21.84 C
Heater Output: 100.0%
```

```
Press ESC to pause/resume the system
Current Temp: 40.94 C
Heater Output: 0.0%
Temperature Control System with LM35 Sensor
Reference temperature set to 30.00 C
Press ESC to pause/resume the system
Current Temp: 40.53 C
Heater Output: 0.0%
Current Temp: 40.37 C
Heater Output: 0.0%
Current Temp: 40.29 C
Heater Output: 0.0%
Current Temp: 40.13 C
Heater Output: 0.0%
Current Temp: 40.05 C
Heater Output: 0.0%
Current Temp: 39.97 C
```

## 5.2 Discussion

Across the four simulated systems, we observe a clear evolution in control strategy, complexity, and performance. The initial bang-bang system provided a basic on-off mechanism with fixed temperature stepping, offering simplicity but limited precision and efficiency. The subsequent proportional-only PWM system introduced smoother control by varying heater power based on instantaneous error, improving stability but still lacking memory and predictive capability.

The third system, implementing PID control with hysteresis, significantly enhanced performance by combining real-time proportional, integral, and derivative terms with a thermal buffer zone. This enabled faster convergence and reduced oscillations near the setpoint. Finally, the most advanced system integrated ADC-based PID control with live LM35 sensor feedback and pause/resume functionality, offering autonomous operation, accurate real-world response, and user interactivity. This version best represents a practical and adaptable closed-loop control system. Together, these simulations highlight the trade-offs between simplicity and performance. While each system has its application domain, the final ADC-PID model demonstrates the most balanced and robust approach for precise temperature regulation in embedded environments.

## 6. DISCUSSION

### **6.1 Limitations of Findings**

While the progressive simulation and analysis of the four control systems provided valuable insights into embedded temperature regulation techniques, several limitations were observed across their implementation, design assumptions, and simulation fidelity:

- 1. Lack of Real-World Thermal Dynamics Modeling**

All systems used simplified temperature increment/decrement logic (e.g., linear steps or proportional changes) to simulate heating and cooling behavior. These do not account for real-world thermal inertia, sensor lag, or non-linear heat transfer, which may lead to different behavior in physical deployment.

- 2. Limited Environmental Disturbance Handling**

The systems were tested in idealized conditions without modeling external factors such as ambient temperature variations, noise in ADC readings, or power supply fluctuations. This restricts the generalizability of results to more complex environments.

- 3. Absence of Full PID Tuning Strategy**

Although the PID gains were manually selected and produced acceptable responses, no formal tuning method (e.g., Ziegler–Nichols or Cohen–Coon) was applied. This may result in suboptimal performance, particularly under varying load or thermal conditions.

- 4. Fixed Setpoint and Thresholds**

The reference temperature and hysteresis bounds were statically defined in code, limiting flexibility. Dynamic adjustment or adaptive thresholding would be more suitable in real-world applications requiring varied thermal profiles.

- 5. No Long-Term Stability or Overshoot Testing**

Simulations were run over short durations, mostly focusing on initial convergence. Long-term drift, oscillation behavior, and overshoot beyond setpoint were not evaluated in depth, particularly under PID control.

- 6. Manual Input and Limited User Interface**

Systems requiring manual temperature entry via UART may not reflect true autonomous operation. Although the final system integrated sensor input, the control logic still depends on discrete UART interactions, which may not scale for continuous or complex tasks.

- 7. Simplified Hardware Representation**

The heater was represented virtually through PWM logic without real

power electronics, and the LM35 sensor was used in a controlled setting. This omits real hardware constraints like actuator non-linearity, saturation, or thermal feedback delay.

## **6.2 Design Limitations**

In this topic Design Limitations refers to the drawbacks and flaws related to mainly the hardware and the simulation environment linked to the systems. These plays a crucial role in making a system stable and efficient, also these limitations can also be the core reason for a system to be failing completely. Checking the limitations in a system is good practice to fix flaws to create an even more robust and reliable system. The benefit of using an STM board is that even though its based on a comparatively complex architecture compared to easily available MCU, it's an industry grade board which means it has heavy versatility and robust core system and features. The Design Limitations of the project are stated in details below for cautions and future work.

### **1. Simplified Heater and Sensor Simulation**

Most systems used virtual temperature changes and lacked interaction with real thermal elements, leading to unrealistic response dynamics and oversimplified feedback behavior.

### **2. No Real-Time Operating System (RTOS)**

All control loops relied on blocking delays (`HAL_Delay`), limiting timing precision and preventing multitasking—unsuitable for real-time or safety-critical applications.

### **3. Idealized ADC and Sensor Handling**

Even in the final system using the LM35 sensor, no noise filtering, calibration, or averaging was applied, reducing measurement reliability under real-world conditions.

### **4. Limited Fault Tolerance**

The systems did not simulate or handle critical events like sensor failure, ADC errors, or communication faults, making them fragile under unexpected runtime conditions.

### **5. UART-Based Manual Control Interface**

Dependence on UART for entering temperature or pausing the system limits automation and scalability, as this approach is not viable in autonomous or deployed environments.

## 6.3 Future Work

Building on the insights and limitations identified throughout this study, several avenues for future development and refinement are proposed to enhance the accuracy, robustness, and real-world applicability of the temperature control systems:

1. **Hardware-Integrated Testing with Real Heating Elements**

Future implementations should interface with actual heaters or thermal actuators to capture real thermal dynamics, delays, and power control behavior. This will validate control logic under realistic load conditions and reveal practical response constraints.

2. **Advanced Sensor Handling and Calibration**

To improve measurement accuracy and noise immunity, future systems should incorporate calibrated sensors with filtering, averaging, and error detection mechanisms.

3. **Implementation of RTOS-Based Control**

Replacing blocking delays with timer-interrupt-driven task scheduling would allow more precise control timing, better resource management, and true multitasking—essential for complex embedded systems.

4. **Robust Fault Detection and Safety Features**

Enhancing fault tolerance through sensor disconnection detection, ADC error handling, and watchdog timers will make the system more reliable under unpredictable real-world conditions and better suited for industrial use.

5. **Closed-Loop Adaptive Control and Auto-Tuning**

Future versions can incorporate adaptive PID control or machine-learning-assisted tuning to dynamically adjust control parameters based on system behavior, improving performance across varying thermal environments and load profiles.

6. **Scalable and Autonomous Interface Design**

Moving beyond UART-based manual input, systems should include touch-screen interfaces, remote control via Bluetooth/Wi-Fi, or integration with supervisory systems for autonomous or networked operation.

7. **Comprehensive Long-Term Testing**

Extended simulation and deployment testing should be conducted to evaluate long-term stability, overshoot behavior, and response to disturbances—crucial for systems expected to operate continuously or in mission-critical roles.

## 7. CONCLUSION

This project implemented and compared four STM32-based temperature control systems, from basic bang-bang to an ADC-driven PID controller with real-time sensor feedback. Each system showed progressive improvement in accuracy and stability, with the final design offering practical closed-loop control using PWM and UART interaction. Despite effective results, limitations like blocking delays, simplified modeling, and lack of fault handling remain. Future enhancements should include real hardware integration, improved sensor accuracy, and interrupt-driven control for more precise and reliable embedded performance.

The key findings and summary of the project can be summarized as follows:

1. The project successfully developed and analyzed four temperature control systems on an STM32 microcontroller: **Bang-Bang, Proportional PWM, PID with Hysteresis**, and an **ADC-based PID system using an LM35 sensor**.
2. Each version showed improved **control precision, thermal stability**, and **automation**, progressively enhancing system responsiveness and convergence to the setpoint.
3. The final system demonstrated the use of **real-time sensor feedback, PWM modulation**, and **user-interactive control** through UART-based pause/resume commands, making it the most complete and realistic model.
4. Simulation results confirmed that **PID-based systems significantly outperformed simpler logic** in terms of overshoot control, steady-state accuracy, and smoother transitions.
5. The study identified key limitations, such as **simplified heater modeling, blocking delay-based loops**, and **absence of real actuator dynamics**.
6. Future work should focus on incorporating **real heating hardware**, replacing delay-based loops with **interrupt-driven control**, and improving **sensor accuracy and fault tolerance** for more reliable embedded deployment.

## 7. REFERENCES

1. Jongcheol Park, Hyun Gue Hong- 3D Integrated Physics Package using MEMS Alkali Vapor Cell for Miniature Atomic Clocks
2. Tianyu Liu, Duo Pan, Jingbiao Chen - Optical Pumped Cesium Atomic Clock With Multi-pole Magnet
3. Yuanhong Cao, Xingwen Zhao, Lin Yang - Progress in the Development of Commercial Optically pumped Cesium Atomic Clock
4. STM32U575I-EV DataBrief - <https://www.st.com/en/evaluation-tools/stm32u575i-ev.html>
5. Youtube – STM32 tutorials, Atomic Clock understanding
6. ChatGPT – Syntax fixes, simple queries