Juan Esteban Navarro Camacho
Proposed solution for: Temperature control problem
navarrocamje@gmail.com

Solution: Temperature control problem

Overview

The solution achieved will cover the abstraction of a master loop that will control the executions of child loops. These child loops will control the abstractions defined for every sensor and the memory management.
In order to visualize the execution of the device simulated, the entry point of the executable will contain a simulator that represents the external interactions that will impact the operation of the main loop.

Requirements definition

This section will cover the requirements coded from the problem description. To achieve a proper syntax of requirement encoding the standard selected was: IEEE 830 1998 *for system*

Juan Esteban Navarro Camacho
Proposed solution for: Temperature control problem
navarrocamje@gmail.com

requirement design (The Institute of Electrical and Electronics Engineers, Inc., 2009) that consists of requirement descriptions for use cases that are close to the hardware implementation or are related to low level software implementation.

The following section will outline sections extracted from the recommended standard for system requirement specification.

System context

The system developed represents a simulation where abstracted sensors define the state of a water container. This water container will reach a certain target temperature. The system will execute tasks that will verify the state of every physical variable defined and set the status of the device accordingly.

Two physical variables are considered to define the state of the system.

1. Temperature in Celsius with a precision of +- 5 C.
2. Level of water that is mapped to a percentage value, from 0 to 100%.

The simulation assumes that the container is attached to any appropriate energy source. Energy disconnections or device turn off scenarios are discarded for this exercise.

System Modes

These states condense the operation of the device simulated:

1. HEATER_ON: Apply energy to elevate the water temperature to a target value.
2. HEATER_TURNED_OFF: Turn off the device to avoid water temperature elevation.

These states will be considered as part of the scope of the solution.

Operational Scenarios

- The system will be in HEATER_TURNED_OFF during the following conditions:
    - The level of water must reach a certain level.
    - The temperature of the water should be less than 100 Celsius.
        - Discrepancies from precision measurement are discarded.

Juan Esteban Navarro Camacho
Proposed solution for: Temperature control problem
navarrocamje@gmail.com

System capabilities, conditions, and constraints

- Detection of temperature:
    - The device should simulate the read operation of a temperature sensor.
    - The device will map the value to a specific reserved memory.
- Detection of water level:
    - The device should simulate the read operation of a water level sensor.
    - The device will map the value to a specific reserved memory.
- Power applied to elevate the water temperature:
    - The device should simulate the write/read operations to a memory reserved location to describe how much “energy is applied” to raise the temperature water.
    - The power of the heater will be:
        - 0%, when the water level is under the 51%
        - or the water level is above the 51% and the temperature has reached the target value.
        - The power applied and the temperature variation are simulated for this exercise to activate the different modes.
        - The power applied to the container will uniformly increase the temperature of the water at a rate defined by the percentage value.

Scope of the proposed solution

- CMAKE is used to compile the project.
    - Two main libraries are developed and installed to be used for this simulation:
        - Memory: Access: to abstract memory management access.
        - Scheduler: to control and schedule every task defined.
    - The location of the installed libraries is defined by the platform where this executable is compiled.
        - Under Ubuntu Systems: **/usr/local/bin** (*Install — CMake 3.30.1 Documentation*, n.d.)
    - These libraries are compiled, installed and linked into the entry file to execute the application.
- The language used to implement the simulation will be C++.

- **Scheduled Task pattern** is used in order to abstract the sub processes that will handle the state and behavior of the simulated device.
    - Scheduled task pattern is inspired by Real Time Operating systems for embedded devices: (Richard Barr, 2016).
    - Every task must has access of the resources:
        - Execution (Run time).
        - Memory Access.

Value of the solution

The strategy to follow to implement the simulation is by the usage of Heisenberg chart (guide, 2021) will describe the organizations of the items based on its priority and the value that it offers to the whole solution.

	More Value	Less Value
Highest effort	• Abstract a general purpose task to be calendarized. • Define the periodicity of a task based on the system requirements.	• Simulate explicitly the memory permissions. • Abstract memory registers to use the Memory address described in the solution. • Simulate water temperature reduction.
Less effort	• Encapsulate the memory access to a class. • Pass parameters to a task to be executed or customized. • Simulate the water level changes. • Simulate the temperature variation of the device. • Control scenarios of water level. • Control scenarios for temperature level.	• Simulate the 16 bit and 8 bit data manipulation. • Define a library to interact with memory management. • Define a library to schedule tasks.

Juan Esteban Navarro Camacho
Proposed solution for: Temperature control problem
navarrocamje@gmail.com

Priority order of implementation: The order to start adding features to the solution will be based on the previous table.

1. Must Do: These requirements/features must be implemented. *Less effort and Highest Value.*
2. Should Do: These should be implemented and documented in cases that are not implementable. *highest effort and heights value.*
3. Nice to have: Features that will not cause any interruption or unexpected behavior if they are not implemented. *lowest effort and lowest value.*
4. Evaluate for other versions: Features that won´t provide further value than the current value committed. *highest effort and lowest value.*

Project configuration

The project is composed by two libraries:

- Memory management: to manipulate data write and read operations.
- Scheduler: To schedule user defined functions.

These libraries are compiled, built and installed using CMAKE (CMAKE, n.d.).
The whole project is configured using CMAKE to simplify the building system of the simulation.

⚠️ Refer to the readme file attached in the repository to compile and run the simulation. ⚠️

The entry point is the file located in <project_folder>/src/main.cpp, this file is compiled along with the libraries in the folder <project_folder>/build.

The executable generated is called <project_folder>/build/TEMPERATURE.

The libraries must be compiled and installed before the main.cpp compilation.

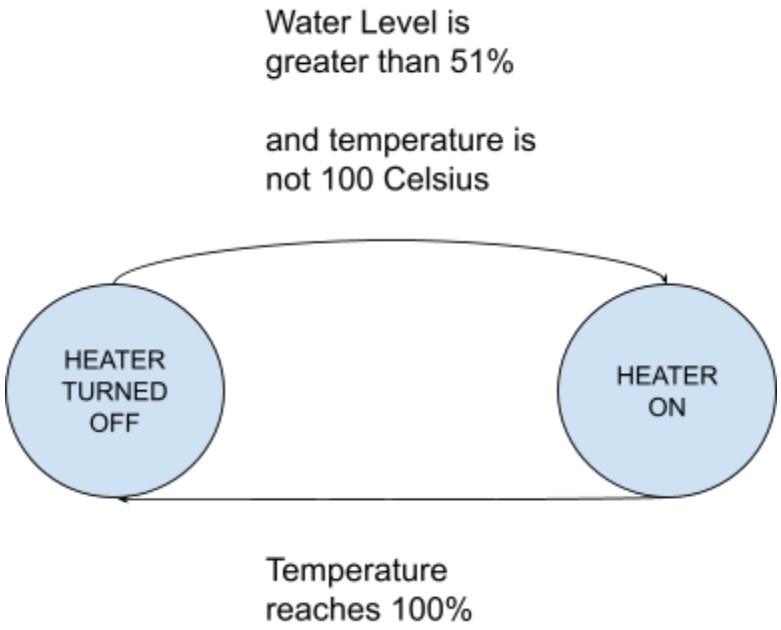
- <project_folder>/memory
- <project_folder>/scheduler

⚠️These libraries are developed and implemented for this simulation. ⚠️

Juan Esteban Navarro Camacho
Proposed solution for: Temperature control problem
navarrocamje@gmail.com

Solution design

Diagram of states of the simulation, this diagram represents the interactions defined for this simulation.



Abstracting tasks scheduling

The following list of tasks are created to simulated the device behavior:

- **Water Level.** Task that will read the value of water level. Runs every 2 seconds
- **Heater Controller.** Task that controls the heater. Runs every 6 seconds.
- **D.** A dashboard to print the interaction of the variables. (Out of the scope of the solution), but created for simulation purposes. Runs every second.
- **E.** A task that will represent the external interactions, to activate the states of the container. (Out of the scope of the solution), but created for simulation purposes. Runs every second.

The following table will describe the distribution of the scheduled tasks in time cost.

Juan Esteban Navarro Camacho
Proposed solution for: Temperature control problem
navarrocamje@gmail.com

<table>
<tr><td rowspan="2">Order of calendarization</td><td colspan="12">Distribution of calendarized tasks (1 second minum block)</td></tr>
<tr><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td><td>D</td></tr>
<tr><td rowspan="4">↓</td><td>E</td><td>E</td><td>E</td><td>E</td><td>E</td><td>E</td><td>E</td><td>E</td><td>E</td><td>E</td><td>E</td><td>E</td></tr>
<tr><td colspan="2">Water Level</td><td colspan="2">Water Level</td><td colspan="2">Water Level</td><td colspan="2">Water Level</td><td colspan="2">Water Level</td><td colspan="2">Water Level</td></tr>
<tr><td colspan="6">Heater Controller</td><td colspan="6">Heater Controller</td></tr>
<tr><td></td><td colspan="12">Total time mapped 12 seconds</td></tr>
</table>

Every task is abstracted by a Task class that receives the following parameters:

- std::function<void(void*)> func : (*Std::unique_lock - Cppreference.com*, 2024) The user customized function to be calendarized, every function must have the following structure:
    - Void return
    - Recieves a void* pointer that is casted based on the user needs.
- void* data: pointer that contains user defined parameters.
- int period: how many seconds this task is going to be executed.

To schedule every task, a queue (*Std::queue - Cppreference.com*, 2024) is used to organize the order of execution and a mutex lock (*Std::unique_lock - Cppreference.com*, 2024) to prevent collisions during the concurrent execution of every task abstracted as a thread.

Represent memory management

- A Memory class is created to abstract read and write operations over an array of Registers.
- A Register struct will be composed by blocks of 8 bits that are mapped to unsigned int of 8 bit size.
- To represent a 16 bits block, the Memory class will make bitwise operations to combine to 8 bit Registers.

The following table will describe the memory mapping set for this simulation.

Juan Esteban Navarro Camacho
Proposed solution for: Temperature control problem
navarrocamje@gmail.com

Memory Direction	Memory Map	Usage
0xFF00	Memory.registers[0]	Register that holds the temperature.
0xFF01	Memory.registers[1]	
0xFF02	Memory.registers[2]	Water level register.
0xFF03	Memory.registers[3]	Heater Power level register.
0xFF04	Memory.registers[4]	Status register

Assumptions

- Energy disconnections or device turn off scenarios are discarded for this exercise.
- Discrepancies from precision measurement are discarded.
- The power applied and the temperature variation are simulated for this exercise to activate the different modes.
- The power applied to the container will uniformly increase the temperature of the water at a rate defined by the percentage value.

Testing

After building the console will display a table like the following image:

The status column will describe the state of the simulated device.
The heater level is mapped to 25%, this value is divided by 5 and added to the current temperature to simulate the temperature variation.
When the temperature is closed to 100, the heater indicator will mark: **Heater OFF**

Juan Esteban Navarro Camacho
Proposed solution for: Temperature control problem
navarrocamje@gmail.com

Bibliography

CMAKE. (n.d.). *Step 2: Adding a Library — CMake 3.30.1 Documentation*. CMake. Retrieved July 22, 2024, from https://cmake.org/cmake/help/latest/guide/tutorial/Adding%20a%20Library.html

guide, s. (2021, October 22). *The Eisenhower Matrix: How to Prioritize Your To-Do List [2024] • Asana*. Asana. Retrieved July 22, 2024, from https://asana.com/resources/eisenhower-matrix

install — CMake 3.30.1 Documentation. (n.d.). CMake. Retrieved July 22, 2024, from https://cmake.org/cmake/help/latest/command/install.html

The Institute of Electrical and Electronics Engineers, Inc. (Ed.). (1998, December 8 December). IEEE Guide for Developing System Requirements Specifications [Guidance for the development of the set of requirements, System Requirements Specification (SyRS), that will satisfy an expressed need is provided. Developing an SyRS includes the identification, organization, presentation, and modification of the require]. In *IEEE Guide for Developing System Requirements Specifications*. https://ranger.uta.edu/~huber/cse4316/Docs/IEEEStd1233-1998.pdf.

Richard Barr. (2016). *Mastering the FreeRTOS™ Real Time Kernel* [FreeRTOS is ideally suited to deeply embedded real-time applications that use microcontrollers or small microprocessors. This type of application normally includes a mix of both hard and soft real-time requirements.]. Real Time Engineers Ltd.

https://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

std::function - cppreference.com. (2024, July 4). C++ Reference. Retrieved July 22, 2024, from https://en.cppreference.com/w/cpp/utility/functional/function

std::queue - cppreference.com. (2024, May 20). C++ Reference. Retrieved July 22, 2024, from https://en.cppreference.com/w/cpp/container/queue

std::unique_lock - cppreference.com. (2024, February 28). C++ Reference. Retrieved July 22, 2024, from https://en.cppreference.com/w/cpp/thread/unique_lock