**Task Scheduler Technical and Operational Documentation**

**Introduction**

This documentation offers an in-depth analysis of the smart thermostat and its task scheduler designed for the CC3220SF LaunchPad-XL microcontroller. It efficiently orchestrates tasks involving button state monitoring, temperature regulation, and server communication in a GPI interrupt-based system.

**System Requirements**

- **Microcontroller:** Texas Instruments CC3220SF LaunchPad XL, featuring advanced GPIO, UART, and Timer functionalities.

- **Software Environment:** Code Composer Studio with SimpleLink CC32xx SDK 7.10.00.13, offering tailored support for the CC3220SF's capabilities.

**Scheduler Algorithm**

**Task Scheduler Function (`taskScheduler`)**

- **Functionality:** This function is the cornerstone of the scheduler, methodically iterating over a series of tasks. Each task is checked against its elapsed time relative to its predefined execution period. The scheduler ensures that tasks are executed only when their specific timing conditions are met, maintaining an orderly and efficient execution sequence.

- **Efficiency:** The scheduler employs a Greatest Common Divisor (GCD) method for calculating the execution periods of tasks. This approach minimizes the number of unnecessary checks and wake-ups, thereby conserving CPU cycles and energy, which is especially crucial in IoT devices like smart thermostats where power efficiency is a key concern. In this case where the tasks fire at 200,500 and 1000 ms respectively the GCD is 100 ms.

**Task Initialization (`createTasks`)**

- **Configuration:** Each task is distinctly configured with a state, period, and function pointer and put into the tasks array

- **Specific Tasks:**

- **Button Check Task:** triggered every 200ms, this task is pivotal for user interaction. It monitors the state of buttons to adjust the thermostat's set\_point. If the increase button is pressed, the set\_point temperature is incremented, allowing the user to dictate at what temperature the heater activates. Conversely, pressing the decrease button lowers the set\_point, giving users control over the heating threshold.

- **Temperature Monitoring Task:** Executing every 500ms, this task reads the current temperature and compares it to the set\_point. Based on this comparison, it controls the heater and the LED indicator. If the room temperature is at or below the set\_point, the heater is turned on (indicated by the LED), to bring the room temperature up to the desired level. Conversely, if the room temperature is above the set\_point, the heater is turned off to maintain the desired temperature without overheating the space.

- **Server Communication Task:** Operating every 1000ms, this task is responsible for assembling and sending data to a server. The data transmission is carried out over UART (Universal Asynchronous Receiver-Transmitter), a standard communication protocol in embedded systems. It formats the data as a string in the structure <%02d, %02d, %d, %04d>\n\r, where:

**%02d for room temperature:** Represents the room temperature in a two-digit ASCII decimal format (00 - 99 degrees Celsius).

**%02d for set\_point temperature:** The target temperature, also in two-digit ASCII decimal format (00-99 degrees Celsius).

**%d for heater status:** A single digit indicating the heater's state, ‘0’ for off and ‘1’ for on.

**%04d for uptime:** A four-digit count of seconds since the system was last reset, providing a timestamp of system operation.

**System Operations**

**Input Handling**

- **Button Processing:** The system employs callback functions for button presses. Each callback function is designed to respond to a specific button. When a button is pressed, the corresponding callback function sets either **increase\_set\_point** or **decrease\_set\_point** to true. This adjustment allows the user to change the set-point temperature, effectively controlling when the heater will turn on or off.

- **Temperature Sensing:** The temperature is read only when the Temperature Monitoring Task is called, which occurs every 500ms. This task reads the current temperature from the sensor and compares it to the set-point. Based on this comparison, it decides whether to activate or deactivate the heater and the LED, maintaining the desired temperature in the environment. To read the temperature a TMP 116 temperature sensor is used

**Output Management**

- **Heater and LED Control:** Adapts the heating system and LED indicators according to real-time temperature data when the Temperature Monitoring Task is called

- **Server Communication:** Sends structured data packets (`<%02d, %02d, %d, %04d>\n\r`) to the server, detailing room temperature, set\_point, heater status, and system uptime.

**State Machines**

**Button State Machine**

Below is written documentation, however a state-machine diagram is also available in the task-scheduler-and-state-machine-diagrams file.

**Detailed States, Actions, and Transitions**

1. **Button\_State\_Init:**

**- Action:** Initializes system variables, preparing for button input monitoring. Also resets both flags increase\_set\_point and decrease\_set\_point

- **Transition:** Proceeds to `**Button\_State\_Idle**`.

2. **Button\_State\_Idle:**

- **Action:** Awaits button input while continuously checking for simultaneous button presses. If both buttons are pressed, it maintains an idle state to avoid conflicting commands. Also resets both flags increase\_set\_point and decrease\_set\_point

- **Transition:**

- To `**Button\_State\_Increase**` if only the increase button is pressed.

- To `**Button\_State\_Decrease**` if only the decrease button is pressed.

- Remains in Idle if both or none of the buttons are pressed.

3. **Button\_State\_Increase:**

- **Action:** Incrementally increases the temperature set\_point. It also performs a check to ensure that only one button is pressed. If the decrease button is pressed, it switches states to manage the conflicting inputs. Also resets the increase\_set\_point flag.

- **Transition:**

- Remains in `**Button\_State\_Increase**` if the increase button continues to be pressed.

- Changes to `**Button\_State\_Decrease**` if the decrease button is pressed.

- Returns to `**Button\_State\_Idle**` if both buttons or none are pressed.

4. **Button\_State\_Decrease:**

- **Action:** Similarly decreases the temperature set\_point, with checks for button presses. Also resets the decrease\_set\_point flag.

- **Transition:**

- Stays in `**Button\_State\_Decrease**` if the decrease button is pressed.

- Moves to `**Button\_State\_Increase**` if the increase button is pressed.

- Reverts to `**Button\_State\_Idle**` if both or no buttons are pressed.

**Heater State Machine**

Below is written documentation, however a state-machine diagram is also available in the task-scheduler-and-state-machine-diagrams file.

**Detailed States, Actions, and Transitions**

1. **Heater\_State\_Init:**

- **Action:** Analyzes initial conditions and decides the initial state based on the current temperature relative to the set\_point.

- **Transition:**

- To `**Heater\_State\_ON**` if the temperature is at or below the set\_point.

- To `**Heater\_State\_OFF**` if the temperature is above the set\_point.

2. **Heater\_State\_ON:**

- **Action:** Activates the heater and LED to indicate heating. Continuously monitors the temperature to determine if heating should continue or cease.

- **Transition:**

- Remains at `**Heater\_State\_ON**` if the temperature is at or below the set\_point.

- To `**Heater\_State\_OFF**` if the temperature is above the set\_point.

3. **Heater\_State\_OFF:**

- **Action:** Deactivates the heater and LED, indicating no heating is needed. It checks if the temperature falls below the set\_point to potentially reactivate the heater.

- **Transition:**

- To `**Heater\_State\_ON**` if the temperature is at or below the set\_point.

- Remains at `**Heater\_State\_OFF**` if the temperature is above the set\_point.

**Timer Configuration (`initTimer`)**

- **Purpose:** The **initTimer** function is critical in setting up the timer that dictates the operation of the task scheduler. It configures the timer to match the GCD of the task periods, thereby optimizing the efficiency of task execution.

**Main Thread Function (`mainThread`)**

**Initialization and Setup:** The **mainThread** function is the starting point of the thermostat system. It initializes the necessary hardware components and drivers required for the system's operation. This includes setting up GPIO for button input and LED output, initializing UART for server communication, and configuring the temperature sensor interface.

**Task Creation and Scheduling:** After initialization, **mainThread** proceeds to create the tasks required for the thermostat's operation by calling **createTasks**. It sets up the task scheduler with the tasks' specific intervals and functionalities, ensuring that each task is ready for execution.

**Main Loop Execution:** Once the tasks are set up, the **mainThread** enters a continuous loop. In this loop, the function consistently checks for the timer flag set by the **initTimer** function. When the flag indicates that the timer period has elapsed, **mainThread** calls the **taskScheduler** function to check and execute the tasks as per their scheduling. This loop forms the core operational cycle of the thermostat, continuously monitoring, and responding to changes in button inputs, temperature readings, and server communication requirements.

**System Stability and Responsiveness:** The design of **mainThread** ensures that the thermostat system remains stable and responsive. By continuously cycling through task checks and executions, the system can promptly respond to user inputs, maintain desired temperature levels, and communicate status updates, all while efficiently managing resource utilization.

**Callback Functions**

- **Timer Callback (`timerCallback`):** This function is invoked each time the timer set by **initTimer** reaches its interval. Its primary role is to signal the task scheduler that the set period has elapsed and it is time to check and potentially execute the scheduled tasks. This callback is essential for maintaining the rhythmic execution of tasks, ensuring that actions such as temperature readings and button state checks are performed at consistent intervals. When it is called every 100 ms it sets **TimerFlag** to 1 indicating that it is true and allowing the infinite loop in **mainThread** to continue on and call **taskScheduler** (this is because in the while loop it waits for **TimerFlag** to be set to 1 with the line while(!TimerFlag){} )

- **Button Callbacks (`gpioButtonFxn0`, `gpioButtonFxn1`):** These functions are triggered when the respective buttons are pressed. Each callback function is associated with a specific button, and its execution results in changes to the thermostat settings:

**gpioButtonFxn0:** This callback is executed when the button designated for increasing the temperature set\_point is pressed. It sets a flag, increase\_set\_point, to true. This flag's state is later checked by the Button Check Task to determine whether to increment the temperature set\_point.

**gpioButtonFxn1:** Similarly, this callback responds to the button assigned for decreasing the temperature set\_point. Upon button press, it sets decrease\_set\_point to true, signaling the Button Check Task to lower the temperature set\_point.

These callbacks play a critical role in enabling user interaction with the thermostat, allowing real-time adjustments to the heating system's set-point based on user preferences.

**initHeaterVars Function**

**Functionality:** The **initHeaterVars** function is crucial for setting the initial state of the thermostat's heating control. Upon system startup, this function is called to initialize the temperature-related variables.

**Temperature Reading**: It starts by reading the current ambient temperature using **readTemp()**, a function that fetches the latest temperature data from the sensor.

**Set-Point Logic:**

If the read temperature is greater than 5 degrees Celsius, the function sets the thermostat's set-point to 5 degrees below the current temperature. This logic implies that if the room is relatively warm (above 5 degrees), the system prepares to maintain a comfortable environment by setting a lower target temperature, potentially activating the heating if the room temperature drops significantly.

Conversely, if the temperature is 5 degrees Celsius or lower, the set-point is set to the current temperature. This approach indicates that the system will activate heating to increase the temperature only if it drops further, maintaining the current level of warmth.

Initial Conditions: This function establishes the initial operating parameters for the thermostat, ensuring that the system starts with a reasonable set-point based on the existing environmental conditions. This early setup is key to the thermostat's efficiency, as it prevents unnecessary heating in already warm conditions and ensures comfort in cooler environments.

**References**

*CC3220SF-LAUNCHXL Development kit | TI.com*. (n.d.). Www.ti.com. <https://www.ti.com/tool/CC3220SF-LAUNCHXL>

‌ *CC3220 SimpleLinkTM Wi-Fi ® LaunchPadTM Development Kit Hardware User’s Guide CC3220 SimpleLinkTM Wi-Fi ® LaunchPadTM Development Kit Hardware*. (2017). <https://www.ti.com/lit/pdf/swru463>

‌ *CCSTUDIO Code Composer Studio (CCS) Integrated Development Environment (IDE) | TI.com*. (2020). Ti.com. <https://www.ti.com/tool/CCSTUDIO>

*SIMPLELINK-CC32XX-SDK Software development kit (SDK) | TI.com*. (n.d.). Www.ti.com. Retrieved November 21, 2023, from <https://www.ti.com/tool/download/SIMPLELINK-CC32XX-SDK>

*CS 350 Project Thermostat Lab Guide*. (n.d.). Retrieved November 21, 2023, from <https://learn.snhu.edu/content/enforced/1399359-CS-350-J7824-OL-TRAD-UG.23EW1/course_documents/CS%20350%20Project%20Thermostat%20Lab%20Guide.pdf?_&d2lSessionVal=izpplwWXnSGqZe7YLfzvpa2S8&ou=1399359>

*zyBooks*. (n.d.). Learn.zybooks.com. Retrieved November 21, 2023, from <https://learn.zybooks.com/zybook/CS-350-J7824-OL-TRAD-UG.23EW1/chapter/3/section/1>

*zyBooks*. (n.d.). Learn.zybooks.com. Retrieved November 21, 2023, from <https://learn.zybooks.com/zybook/CS-350-J7824-OL-TRAD-UG.23EW1/chapter/4/section/1>

‌

‌

‌