SETR 24/25 Mini-project

Prototyping a thermal process control system using the nRF52840-DK and Zephyr

Introduction

The aim of this short project is to develop a prototype of a thermal process control system. The actuator (heater) is implemented by a power resistor and the temperature sensing is carried out by a digital temperature sensor (I2C interface).

The system has two operator interfaces, one for configuration and the other one for normal operation:

- The configuration interface is supported on the UART. An engineer can use a PC terminal to set relevant parameters, such as maximum temperature allowed and the controller settings, as well as to obtain real-time data about the process.
- The operation interface allows a user to turn the system on and off, and to set the temperature. There are also some indicators about the system operation status.

Specification

The controller comprises the following inputs and outputs:

- Operation interface:
  - Button 1: toggles the system on/off
  - Button 2: increases the desired temperature by 1ºC degree
  - Button 4: decreases the desired temperature by 1ºC degree
  - Led 1: indicates if the system is on (led on) or off (led off)
  - Led 2: “normal state” indication, i.e. on when the temperature is within a vicinity of +/- 2 Celcius degrees of the target temperature
  - Led 3: warning-low, on if the temperature is more than 2 degrees below the target temperature
  - Led 4: warning-high, on if the temperature is more than 2 degrees above the target temperature
- Configuration interface:

- via UART, set to 115200, 8, n, 1
- Similar to Assignment 2. Frame oriented, full ASCII format (the user shall be able to type commands and obtain the corresponding replies using a simple terminal)
- Commands and answers:
  - General format: # CMD DATA CS ! (no spaces between fields)
  - #: one byte, Start of Frame symbol
  - !: one byte, End of Frame symbol
  - CMD: one byte, defines the command or reply
  - DATA: variable size, command/reply arguments
  - CS: modulo-256 checksum. Sum of numerical value of CMD and DATA[i] bytes.
  - Minimum set of commands to implement
    - #Mxxxyyy! - Set maximum temperature to “xxx” (in ºC). “yyy” is the checksum
    - #Cyyy! - Obtain the current process temperature. “yyy” is the checksum.
      - Reply is “#cxxxyyy!”, where “xxx” is the temperature (in ºC) and “yyy” is the checksum
    - #Sxxx...xxxyyy! - Set the controller parameters. “xxx...xxx” are the controller parameters. You have to define them according to the type of controller used (eg. Kp, Ti, Td for a PID, hysteresis for an on/off controller, ….). “yyy” is the checksum.
    - All commands should be acknowledged. Include at least:
      - #Eoyyy! . No error/Ok. “yyy” is the checksum
      - #Efyyy! . Framing error. “yyy” is the checksum
      - #Esyyy! . Checksum error. “yyy” is the checksum
      - #Eiyyy! . Invalid command. “yyy” is the checksum
      - Note that commands that have an explicit reply (e.g. #Cyyy!) don’t need a “No error/Ok” acknowledgment. The reply message is an implicit confirmation of the success of the command.

- Power & sensing interface
  - One I2C temperature sensor, TC74A0-3.3VAT
  - A power resistor (E.g. 5 W, subject to availability). Interface made via a TN0702 connected to a digital output (and eventually a PWM channel).

NOTE: The devkit supply (VDD) is 3 V. **Feeding an input pin with a voltage higher than 3 V can destroy the devkit**.

Please use the internal power supply (VDD and GND at connector P1) to power the temperature sensor (TC74A0-3.3VAT), and by doing so assuring that the voltage level at its output is adequate.

Be careful with the assembly of the TN0702 FET, because the power resistor will be connected to the bench power supply at higher voltages (maybe ~10 V), so a mistake in this connection can also destroy the devkit.

According to the real-time model, external read and write operations are carried out asynchronously. That is, when the external computing device (a PC) sends a command, its accesses a Real-Time Database (RTDB) to set/read the corresponding values. This RTDB is, concurrently, accessed by several internal real-time tasks that keep it synchronized with the I/O interfaces. Refer to Figure 1 for an overview of the overall system architecture.

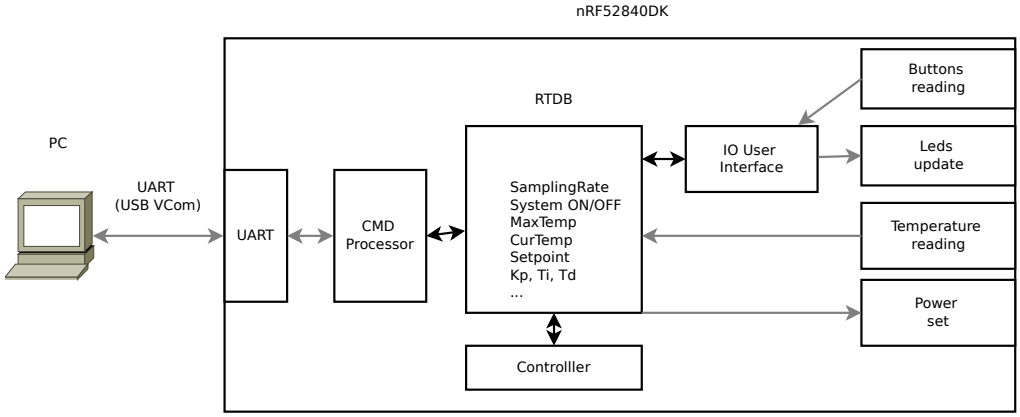

Figure 1: Temperature Controller General Architecture

Tasks should have suitable activation modes and use adequate IPC mechanisms.

Eventual race conditions should be prevented using suitable mechanisms.

Note 1: the emphasis of this work is on structuring the software according with the real-time model. A solution using one loop with all functionality inside it will be evaluated with 0 (zero), even if “it works”.

Note 2: the global quality of the solution is relevant. This includes e.g. documenting the code, using unit testing (at least for key parts), using suitable activation methods and synchronization protocols for tasks and developing a robust protocol for the UART communications, to name just a few.

Deliverables

- A three to four pages report, pdf format, submitted via eLearning, with:
  - Page 1:
    - Identification of the course and assignment
    - Identification of the group members (first name, last name and ID number)
    - Links to GitHib project repositories containing the project code and documentation
  - Pages 2-4:
    - Brief description (diagrams and text) of the tasks, its execution patterns and relevant events
    - Justification of the chosen task activation models (event vs time-triggered), IPC and synchronization mechanisms
    - **Real-time characterization of the tasks and a discussion of the system schedulability (very important)**

Schedule

- This assignment runs for classes 12 to 14;
- The application will be demonstrated during class 14;
- Each group has a slot of 10 minutes and should prepare a convincing demonstration. The objective is not to show the code, but, instead, to convince the audience (me) that the solution works, is efficient and robust;
- The last commit of the code should be made immediately before the demos start and the report can be submitted until one week after the last class. Significant modifications to the code after the demos start are not allowed.