SETR 22/23 Mini-project

Prototyping a “smart” I/O module for the Industrial Internet of Things (IIoT) using Zephyr

Introduction

The aim of this short project is to implement simple Input/Output sensor/actuator modules, which are core parts of modern IoT and IIoT applications.

The embedded I/O module is based on the Nordic nRF52840 Devkit used in the lab classes, using the Zephyr RTOS. The firmware shall be structured according to the real-time model.

The “smart” adjective appears in the (nowadays) conventional sense (i.e. programmable/flexible node), not meaning that it incorporates AI/ML/...

In a more realistic situation, the wireless radio of the DevKit would be used, e.g. via the BLE protocol. However, adding BLE is not trivial (it is not “rocket science”, but it takes a few 100’s of lines of code and a few hours of effort to learn how to use it), so the interface to the I/O module is made via UART. Note that, if the firmware is properly implemented (i.e. following a modular architecture), moving from the UART interface to BLE would be straightforward.

Specification

The smart I/O module comprises the following inputs and outputs:

- 4 digital inputs (the devkit buttons)
- 4 digital outputs (the devkit leds)
- 1 temperature sensor (TC74, connected via I2C)

> **NOTE**: The devkit supply (VDD) is 3V. Feeding an input pin with a voltage higher than 3V can destroy the devkit. Use the internal power supply (VDD and GND at connector P1) to power the TC74 CI. It is safe to do this as this device consumes a small amount of current.

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

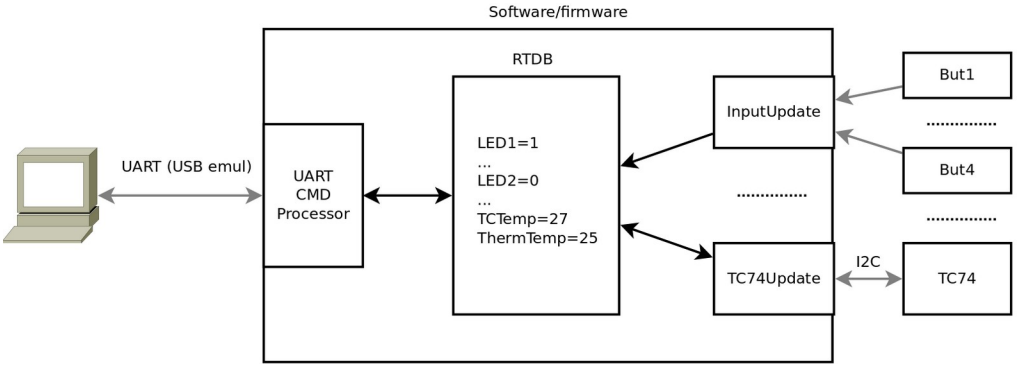

Figure 1: Smart I/O system architecture overview

The UART interface should allow configuring the relevant operational parameters of the I/O's, namely:

- Frequency of update of the input sources (by class: buttons and I2C temperature sensor set sepparately);
- Frequency of update of the digital outputs.

As mentioned above, the interface with the "outside world" is made through the UART. A set of commands should be defined to allow configuring the board, set the outputs and read the inputs.

Task should have suitable activation modes and use adequate IPC 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.

Note3: interested students can try to add the BLE interface. I can supply example code that can be "easily" adapted, and the Nordic Academy has a course on BLE. But if you want to do so, be prepared to spend several hours … and learn a lot in the process.

Deliverables

- A two to three 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
  - Pages 2/3:
    - Brief description (diagrams and text) of the tasks, its execution patterns and relevant events
    - Real-time characterization of the tasks and a discussion of the system schedulability

Schedule

- This assignment runs for classes 11 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. E.g., if your UART protocol allows sending invalid commands, you should include in your demo a scenario that shows it
- 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. No modifications to the code will be allowed after the demos start.